Sensor Web Geoprocessing on the Grid

Aengus Robert McCullough BSc. (Hons.)

Thesis Submitted for the Degree of

Doctor of Philosophy

Newcastle University

School of Civil Engineering & Geosciences

2011

Abstract

Recent standardisation initiatives in the fields of grid computing and geospatial sensor middleware provide an exciting opportunity for the composition of large scale geospatial monitoring and prediction systems from existing components. Sensor middleware standards are paving the way for the emerging sensor web which is envisioned to make millions of geospatial sensors and their data publicly accessible by providing discovery, task and query functionality over the internet. In a similar fashion, concurrent development is taking place in the field of grid computing whereby the virtualisation of computational and data storage resources using middleware abstraction provides a framework to share computing resources. Sensor web and grid computing share a common vision of world-wide connectivity and in their current form they are both realised using web services as the underlying technological framework. The integration of sensor web and grid computing middleware using open standards is expected to facilitate interoperability and scalability in near real-time geoprocessing systems.

The aim of this thesis is to develop an appropriate conceptual and practical framework in which open standards in grid computing, sensor web and geospatial web services can be combined as a technological basis for the monitoring and prediction of geospatial phenomena in the earth systems domain, to facilitate real-time decision support. The primary topic of interest is how real-time sensor data can be processed on a grid computing architecture. This is addressed by creating a simple typology of real-time geoprocessing operations with respect to grid computing architectures. A geoprocessing system exemplar of each geoprocessing operation in the typology is implemented using contemporary tools and techniques which provides a basis from which to validate the standards frameworks and highlight issues of scalability and interoperability.

It was found that it is possible to combine standardised web services from each of these aforementioned domains despite issues of interoperability resulting from differences in web service style and security between specifications. A novel integration method for the continuous processing of a sensor observation stream is suggested in which a perpetual processing job is submitted as a single continuous compute job. Although this method was found to be successful two key challenges remain; a mechanism for consistently scheduling real-time jobs within an acceptable time-frame must be devised and the trade-off between efficient grid resource utilisation and processing latency must be balanced.

The lack of actual implementations of distributed geoprocessing systems built using sensor web and grid computing has hindered the development of standards, tools and frameworks in this area. This work provides a contribution to the small number of existing implementations in this field by identifying potential workflow bottlenecks in such systems and gaps in the existing specifications. Furthermore it sets out a typology of real-time geoprocessing operations that are anticipated to facilitate the development of real-time geoprocessing software.

Acknowledgements

I would like to express my gratitude to everyone that has helped and supported me while undertaking this research. First and foremost, thanks to my supervisors Philip James and Stuart Barr, for their years of invaluable support and guidance without which this would not have been possible. Thanks also to my Geomatics colleagues and countless others in the School that have provided useful support and advice at one time or another.

I have received a great deal of technical assistance via email that has been critical to the success of this research and that deserves acknowledgement here. Alain Roy at University of Wisconsin-Madison has provided invaluable assistance with Globus Toolkit woes. At the National Grid Service Matteo Turilli has helped with GridSAM on numerous occasions, and Simon Collins has provided significant help with the Oracle service. Alex Mckeown at CSIRO provided much appreciated help debugging an SES filter, thank you. I have also received significant email support from 52 North developers Christoph Stasch, Johannes Echterhoff and Thomas Everding. Furthermore, Trevor Arkless at Newcastle City Council has been very helpful in providing access to road traffic travel time data.

My family have been extremely tolerant and understanding throughout the course of this research, particularly my parents Hugh and Kate who have helped me keep body and soul together during the final stages of this research period. Paulette thanks for doing an excellent job of distracting me from my studies when necessary, for buying me motivational cupcakes and for being there one way or another.

This research was funded by EPSRC and the School of Civil Engineering & Geosciences at Newcastle University.

Publications from this Research

The following publications have been produced from the research presented in this thesis:

McCullough, A., James, P., Barr, S. L. (2011) ‘A Typology of Real-Time Parallel Geoprocessing for the Sensor Web Era’ *Proceedings of the Workshop on Integrating Sensor Web and Web-based Geoprocessing, AGILE 2011.* Utrecht, CEUR pp.712-1.

McCullough, A., James, P., Barr, S. L. (2011) ‘A Service Oriented Geoprocessing System for Real-Time Road Traffic Monitoring’ *Transactions in GIS (in press)*

Table of Contents

List of Figures

List of Tables

List of Code Listings

List of Equations

Glossary

ACID: Atomicity, Consistency, Isolation, Durability

AMI: Amazon Machine Image

API: Application Programming Interface

BASE: Basically Available, Soft-state, Eventually Consistent

BPEL: Business Process Execution Language

CA: Cellular Automata

CEP: Complex Event Processing

CFD: Computational Fluid Dynamics

CGSG: Coarse Grained Snapshot Geoprocessing

CSW: Web Catalogue Service

DEM: Digital Elevation Model

DSG: Data Stream Geoprocessing

DSMS: Data Stream Management System

DSP: Data Stream Processing

EOS: Earth Observing System (NASA)

FCD: Floating Car Data

FE: Finite Element

FGSG: Fine Grained Snapshot Geoprocessing

FTP: File Transfer Protocol

GIS: Geographic Information System

GPS: Global Positioning System

HDFS: Hadoop File System

HPC: High Performance Computing

HPF: High Performance Fortran

IaaS: Infrastructure as a Service

IDW: Inverse Distance Weighted

InSAR: Interferometric Synthetic Aperture RADAR

ISO: International Standards Organisation

IT: Information Technology

ITN: Integrated Transport Network (Ordnance Survey)

LOD: Load on Demand

MIMD: Multiple Instruction Multiple Data

MISD: Multiple Instruction Single Data

MNMP: Multiple Node Multiple Processors

MNSP: Multiple Node Single Processor

MPI: Message Passing Interface

MPP: Massive Parallel Processing

MPTS: Moving Point Time Series

NCC: Newcastle City Council

NOSQL: Not Only SQL

NOW: Network of Workstations

O&M: Observations and Measurements

OASIS: Organisation for the Advancement of Structured Information Standards

OCCI: Open Cloud Computing Interface

OGC: Open Geospatial Consortium

OGF: Open Grid Forum

OGSA: Open Grid Services Architecture

OGSA-BES: OGSA Basic Execution Service

OGSA-DAI: Open Grid Services Architecture Data Access and Integration

ORCHESTRA: Open Architecture and Spatial Data Infrastructure for Risk Management

OSWA: Open Sensor Web Architecture

OWS: OpenGIS Web Services

PaaS: Platform as a Service

RAC: Real Application Clusters

REIS: Real-time Earthquake Information System

REST: Representational State Transfer

RM-ODP: Reference Model for Open Distributed Processing

SAAJ: Soap with Attachments API for Java

SaaS: Software as a Service

SensorML: Sensor Model Language

SFTS: Spatial Field of Time Series

SIMD: Single Instruction Multiple Data

SISD: Single Instruction Single Data

SLA: Service Level Agreement

SNMP: Single Node Multiple Processors

SOA: Service Oriented Architecture

SOAP: Simple Object Access Protocol

SOS: Sensor Observation Service

SPARK: Spatial Reclassification Kernel

SQL:Structured Query Language

SWE: Sensor Web Enablement

TFSS: Time Field of Spatial Series

TIN: Triangulated Irregular Network

TML: Transducer Model Language

TOID: Topographic Identifier (Ordnance Survey)

UDDI: Universal Description Discovery and Integration

UML: Unified Modelling Language

URI: Universal Resource Identifier

VO: Virtual Organisation

WCS: Web Coverage Service

WFS: Web Feature Service

WMS: Web Mapping Service

WMTS: Web Map Tile Service

WPS: Web Processing Service

WSDL: Web Services Description Language

WS-I BP 1.1: WS-I’s Basic Profile 1.1

WS-I: Web Services Interoperability

WSN: Web Services Notification

WSRF: Web Services Resource Framework

Chapter 1 Introduction

1.1 Context

Recent technological advancements in the acquisition and distribution of spatial data are set to have a profound impact on Geographic Information Systems (GIS). Traditional methods of spatial data acquisition are rapidly being augmented with a new generation of digital sensors that are capable of capturing spatial phenomena in real-time and without human intervention. Furthermore, the widespread proliferation of the internet has created an opportunity to make this information available to a wider range of users than ever before. The term ‘sensor web’ has been coined to describe the vision of numerous inter-connected digital sensors across the globe that can be discovered and accessed through the internet (Reichardt, 2005). Although this vision is not yet a reality it has the potential to make a significant impact on the field of GIS, particularly for applications such as environmental monitoring, disaster management, climate change prediction, logistics and the management of utilities. The sensor web vision is probably best exemplified by the European funded Global Earth Observation System of Systems (GEOSS) project which is described as a “comprehensive, near real-time information system that will coordinate present and future observation systems, monitor the entire Earth, track changes in all of its physical, chemical, and biological systems, and serve as an essential decision support tool for a vast range of issues and user groups” (Acache, 2007).

The sensor web vision has coincided with a more general evolution of the GIS landscape; monolithic software packages are gradually being replaced by collections of distributed services (Section 1.2.1). Rather than storing and processing spatial data on a local desktop workstation, data is stored in web accessible repositories and processed remotely. This client-server approach has three advantages (Abel *et al.*, 1999); firstly, less investment in hardware and software is required by end-users as data and processing resources can be accessed remotely. Secondly, the ability to maintain a central data repository and access it as a service facilitates the integration of disparate data sources and allows them to be easily updated. Thirdly, voluminous geospatial data is not easily portable and the ability to analyze it remotely is therefore desirable. Significant work has been undertaken to standardise interfaces to geospatial services across the industry to promote data sharing and interoperability between disparate organisations (Lee and Percivall, 2008). This evolution has provided an opportunity to integrate the sensor web vision and GIS, because in a distributed architecture sensors and their data can be discovered, described and accessed through well defined service interfaces in much the same way as other data sources.

More recently another trend referred to as grid computing has emerged in the Information Technology (IT) sector that has been hailed as the third information technology wave (Sun *et al.*, 2005). Grid computing is defined by Foster (2002) as a computing infrastructure that enables the sharing of heterogeneous computing resources across organisational boundaries, without centralised control, using standard, open and general purpose protocols and interfaces. It provides a framework in which access to heterogeneous computing resources such as processor cycles and data storage devices can be federated, thus facilitating geographically dispersed collaboration, permitting inexpensive access to high end computational capabilities and enabling increased use of idle computing capacity (Foster and Kesselman, 1999). From a GIS perspective grid computing presents an exciting opportunity; it provides an extension to the client-server approach whereby spatial analysis can be outsourced on a massive scale to a large cluster of computers rather than to a single server. Furthermore, the ability to task processors on demand is likely to prove useful for sensor web applications that exhibit temporal variation in the amount of computational power they require.

Another distributed computing infrastructure known as cloud computing has in the last few years become popular which shares many similarities with the grid computing concept. Cloud computing has already had a significant impact on the mainstream IT market (Armbrust *et al.*, 2009, Buyya *et al.*, 2008) and is increasingly being used as a platform for geospatial applications (Baranski *et al.*, 2009, Blunck *et al.*, 2010, Blower, 2010, Chen *et al.*, 2008).

1.2 Background

1.2.1 Service Oriented Architecture (SOA) for GIS

GIS and grid computing conform to a distributed software design referred to as a Service Oriented Architecture (SOA). SOA software is composed of a set of disparate components referred to as services, each of which encapsulates some functionality and a description specifying its purpose and how to interact with it. Web services are a technological implementation of SOA principles that have become the de-facto communication platform for distributed systems. Web services are defined by Curbera *et al* (2002) as a platform neutral, vendor independent framework based on open XML standards that specifies communication protocols, service descriptions and service discovery mechanisms to allow application to application interaction.

Using web services, a number of application specific frameworks have been defined to facilitate the sharing and availability of resources such as hardware, software, instruments and data. OpenGIS Web Services (OWS) and Sensor Web Enablement (SWE) are frameworks defined by the Open Geospatial Consortium (OGC) which is the leading standards body for geospatial services. OWS represent a domain specific effort towards making heterogeneous geospatial datasets and processing functions widely accessible through standard service interfaces. Likewise, SWE specifications provide an interface to task heterogeneous sensor collections and retrieve their observations. In contrast, the Open Grid Services Architecture (OGSA) framework, as originally proposed by Foster *et al.* (2002) and managed by the Open Grid Forum (OGF), represents a broader effort towards sharing resources such as computational power, data storage and sensors, using a different set of service interfaces (Chen *et al.*, 2006).

1.2.2 Application Specific Frameworks

The OWS framework fulfils the perceived need for a distinctive set of web services that enable users to meaningfully interact with spatial data. For example, the ability to perform spatial queries on data repositories enables geographic features to be selected based on their spatial relationships such as ‘distance to’, ‘contains’, ‘within’ and ‘intersects’. This ability to retrieve precisely the features that are required is necessary in a SOA as it minimises network communication cost; the alternative being to download an entire dataset and query it locally (Scharl and Tochtermann, 2007). OWS incorporates the Web Feature Service (WFS), Web Coverage Service (WCS), Web Mapping Service (WMS), Web Processing Service (WPS) and Web Catalogue Service (CSW). WFS and WCS define interfaces to deliver vector and raster data respectively, and the WMS enables both raster and vector data to be combined into a visual map document. WPS enables geo-processing operations to be published as a service and CSW defines a registry service that enables other OWS services to be discovered (Hobona *et al.*, 2007).

The SWE framework has been designed to facilitate the emerging sensor web and is comprised of a complete and structured set of XML based languages for describing sensor models, sensors and their observations. It also includes a set of service interfaces to perform sensor discovery, observation delivery and dynamic tasking of sensor systems (Botts *et al.*, 2006).

1.2.3 Scalability and Performance in Sensor Web Geoprocessing

The sensor web promises the ability to integrate remote, in-situ, fixed and mobile sensors of every kind and communicate with them in a uniform manner via a set of services; this is envisioned to greatly facilitate data fusion and to enable software applications containing mashups of live environmental data to be easily created (Botts *et al.*, 2006). However, as noted by Chen *et al* (2005), monitoring events and entities and making predictions about their future state carries a large computational burden. Furthermore, uncertainty in the behaviour of real world phenomena makes it difficult to predict the timing and the magnitude of computational power required (Hingne *et al.*, 2003). Consequently, for applications that only require occasional access to high-end computational capabilities there is a need for a system that can react to fluctuations in demand and recruit computational resources as necessary (Foster and Kesselman, 1998). Grid computing has been proposed as a potential solution to the sensor web data deluge.

1.3 Problem Statement

The development of scalable grid and cloud based sensor web geoprocessing applications is currently a difficult process. Due to the significant variation exhibited by geoprocessing tasks in their algorithmic and data properties there is no single solution to scale an application through gridification as different tasks are suited to different techniques (Werder and Krüger, 2009). The recent proliferation of standards in GIS and grid computing provide an important step towards interoperability. However, industry wide disarray in web service specifications makes it difficult to leverage grid computing to improve performance and scalability in sensor web monitoring and prediction applications. Furthermore, given the diversity that sensor web scenarios and their associated geoprocessing algorithms exhibit, there is no “one size fits all solution” to improve the scalability or performance of sensor web processing applications. A lack of a cohesive framework to relate real-time geoprocessing operations with parallel processing techniques has hindered the development of generic software tools and solutions thus far. Consequently there is a perceived need to consolidate existing parallel geoprocessing techniques, and to align web service based standards, so that sensor web geoprocessing applications can easily leverage the scalability and performance advantages of distributed computing.

1.4 Scope of the Thesis

While there are numerous issues surrounding the integration of grid computing and sensor web into GIS workflows this thesis focuses only on interoperability, scalability and performance in relation to monitoring and prediction systems. This thesis attempts to identify common design patterns in distributed sensor web geoprocessing systems and attempts to solve the interoperability, scalability and performance issues that frequently occur in such designs. Specifically, the suitability of existing and proposed interface and encoding standards are explored in order to identify areas in which they could be augmented or improved. Additionally an attempt is made to identify commonly occurring workflow bottlenecks in these designs and to suggest alternative approaches. It is anticipated that the outcomes from this research will facilitate the development of distributed monitoring and prediction systems using sensor web and grid computing technology by providing a framework from which standard development tools can be created.

1.5 Organisation of the Thesis

The remaining Chapters in this thesis are organised as follows:

Chapter Two reviews standards, tools and techniques for geoprocessing on the grid. Firstly, the suitability of grid computing for geospatial monitoring and prediction systems is established. Secondly, the current state of the art in sensor web, grid computing and geospatial web services are set out and parallel geoprocessing tools and techniques are reviewed. Finally, existing efforts to integrate grid computing into geospatial workflows are examined and a research agenda for real-time geoprocessing on the grid is set out.

Chapter Three details existing efforts to classify geoprocessing operations and explores the effect of introducing real-time data into distributed geoprocessing workflows. The main content of this Chapter is the presentation of a prototypical typology of real-time geoprocessing operations and an attempt to classify common geoprocessing operations in the context of this typology. In addition, an evaluation and critique of the typology is conducted.

Chapter Four provides details of the design, implementation and testing of a scalable real-time geoprocessing system that conforms to the Data Stream Geoprocessing (DSG) category of real-time geoprocessing operation outlined in Chapter 3. The system in question uses grid computing to perform a map-matching operation for a fleet of vehicles in near real-time.

Chapter Five details the design, implementation and testing of another geoprocessing system. In relation to the typology presented in Chapter 3 this system incorporates elements of Fine-grained Snapshot Geoprocessing (FGSG) and DSG. This prototypical system performs road traffic monitoring by using Floating Car Data (FCD) to estimate travel times along different road stretches; the information is subsequently used to plan the quickest route between two locations in a city.

Chapter Six explores the utility of cloud computing by presenting the design, implementation and testing of a system that conforms to the Coarse-grained Snapshot Geoprocessing (CGSG) class of geoprocessing operation. Amazon's Elastic MapReduce service is used to increase the performance of an image processing algorithm known as the Spatial Reclassification Kernel (SPARK).

Chapter Seven discusses the main findings of this research and highlights the overall research contribution of this work.

Chapter Eight concludes the thesis and details the possibilities for future work in this field.

Chapter 2 Geoprocessing on the Grid: A Review of Standards, Tools and Techniques

2.1 Introduction

Sensor web, grid computing and geospatial web services have been identified as key technology areas that are well placed to deal with the problems of scalability and interoperability in real-time geoprocessing systems. In this Chapter the suitability of these technologies to solve the computational and architectural challenges inherent in monitoring real-world phenomena and predicting their future state are reviewed from a geospatial perspective. The major objectives of this literature review are set out as follows:

1. Identify the design characteristics of geospatial monitoring and prediction applications and review the case for a distributed approach to the design of geospatial monitoring and prediction applications.
2. Describe the current state of the art in each of the following key technology areas: sensor web, grid computing and geospatial web services.
3. Review contemporary tools and techniques for geoprocessing in parallel.
4. Examine existing efforts to integrate grid computing into geospatial workflows.
5. Set out a research agenda for real-time geoprocessing on the grid.

The remainder of this Chapter is divided into three logically distinct sections. Section 2.2 reviews geospatial monitoring and prediction applications and examines their suitability for integration with grid computing, thus fulfilling objective 1 above. Section 2.3 considers the array of web service based middleware in GIS and grid computing that enables geoprocessing to take place in a distributed environment, fulfilling objective 2 above. Section 2.4 presents a review of the parallel geoprocessing strategies and data architectures that are outlined in the literature, thus fulfilling objectives 3 and 4 above. The key findings of this review are presented in the summary in Section 2.5 which fulfils objective 5 above. The aims, objectives and research questions of this thesis, are then set out in Section 2.6.

2.2 Characteristics of Geospatial Monitoring & Prediction Applications

Our ability to remotely measure and record real world phenomena pertaining to ourselves and our environment has rapidly increased over recent years due to technological advancements in communication systems (Liang *et al.*, 2005) wireless sensor networks (Culler *et al.*, 2004, Martinez *et al.*, 2004), satellite imaging (Plaza *et al.*, 2009) and satellite positioning systems (Liang *et al.*, 2003). This access to timely information about our environment has enabled us to make better, more informed decisions and to react to changing circumstances as they happen (Aloisio, 2003). Notably, fields such as geohazard monitoring and structure monitoring have allowed us to improve the safety of our environment. Furthermore, our ability to monitor moving entities such as people, vehicles and animals has enabled us to improve logistics and security. The purpose of this Section is to outline the utility and scope of geospatial monitoring and prediction, and to highlight the compute and data characteristics of such systems in order to rationalise the case for a sensor web / grid computing approach to system design.

2.2.1 Real-Time Geohazard Monitoring and Mitigation

Mitigating the effects of disasters relating to geo-hazards is becoming an increasingly important priority. There is a rising trend in the number of extreme weather events and in the cost of such events in terms of lives and economic damage; trends that are attributed to a changing climate and to increasing concentrations of the world’s population in vulnerable areas (Freeman *et al.*, 2003). To highlight the importance placed on geohazard monitoring and mitigation, and the perceived role of SOA and geospatial web services it is worth referring to the European funded Open Architecture and Spatial Data Infrastructure for Risk Management (ORCHESTRA) project. ORCHESTRA has attempted to improve interoperability between risk management organisations by defining a common abstract specification framework, the Reference Model for the ORCHESTRA Architecture, which sets out the building blocks for risk management systems based on OGC, ISO, W3C and OASIS standards (Klopfer and Kanellopoulos, 2008).

According to the International Centre for Geohazards, strategies for mitigating the effects of geohazards fall into six categories (Solheim *et al.*, 2005); careful land-use planning, adherence to good construction practice, physical protection barriers, community preparedness, early warning systems and evacuation routes. The first three strategies enumerated here are required to be in place long before the occurrence of a disaster but the latter three could conceivably benefit from real-time monitoring and observation of real world phenomena.

Technological solutions can often assist communities in preparing for geo-hazard events by coupling sensor networks with computationally intensive models. For example, although earthquake early warning systems currently provide a maximum of seventy seconds warning (Böse *et al.*, 2007), damage limitation can still be achieved using Real-time Earthquake Information Systems (REIS) that give rapid notification of earthquake parameters such as time, location and magnitude (Kanamori, 1997). Such notification enables emergency services to allocate their resources more effectively in the aftermath of an earthquake event. Nakamuru *et al* (2009) describe a REIS in Japan that utilises an 800 node seismometer network that has been deployed throughout the country. Observations are taken from each sensor node every second and transferred to a central processor that maintains three minutes of observation data for the entire network and is updated every second. The processor scans the observation data held in shared memory for evidence of an earthquake by comparing the signal to noise ratio of 1 second and 30 second averages of ground acceleration and maximum amplitude. If an earthquake is detected the system ceases scanning and starts attempting to determine the earthquakes hypocentre (Horiuchi *et al.*, 2005). This real-time system runs on a single dual core Linux machine (Xeon 2.8 Ghz) with 8GB RAM.

Tralli *et al* (2004) argue that the widespread deployment of seismometer networks is not economically viable, and that space based sensing should be used to augment data collected from the ground. Interferometric Synthetic

Aperture Radar (InSAR) provides a spatially continuous dataset showing ground movement. InSAR data can assist in the understanding of earthquake processes, which is likely to improve forecasting, and is also useful for post-earthquake damage assessment (Rejaie and Shinozuka, 2004). However, due to the limited temporal resolution of satellite data, it is unlikely to replace ground based solutions for seismic monitoring.

Wildfire prediction systems have concentrated on predicting fire pre-cursors such as lightning risk and fuel loads (Grasso and Singh, 2008). For example de Groot *et al* (2006) have developed a global wildfire early warning system that is based on weather forecast information and local historic data on fire and weather events. To account for uncertainty in prediction of atmospheric conditions, the same model is run several times using different parameters to provide distributions of possible outcomes. Such an approach is ideally suited to a distributed computing architecture in which each model run can be executed on a different processor. When a fire does break out, it is now possible to model its spread and the effect it has on structures due to recent advances in Computational Fluid Dynamics (CFD) and Finite Element (FE) analysis (Han *et al.*, 2010). The FireGrid project (Han *et al.*, 2010) has demonstrated how grid computing, high-performance computing, command and control systems and wireless sensor networks can be used together to model the progress of a fire. Heavily instrumented buildings typically equipped with 10,000 sensors providing observations of smoke, carbon dioxide and temperature every 0.1 seconds feed into CFD fire models and FE structural models to simulate the fire and its damage to the building. Such models are enormously complex; to simulate a 15 minute fire for a small hotel room is estimated to take 6 hours on a single processor with 1GB of RAM (Han *et al.*, 2010). We can thus infer that a minimum of 24 nodes would be required to perform this computation in real-time. Parallel computing is clearly necessary to achieve results in a useful time period.

Cities or regions that are vulnerable to natural disasters or terrorist attacks are faced with the problem of emergency evacuation route planning. Attempts to solve this problem have traditionally used one of three possible approaches;

micro-simulators, meso-simulators and macro-simulators (Southworth, 1991). Micro-simulators attempt to take into account the movement and behavioural interactions of individual entities such as people and vehicles (Pidd *et al.*, 1996). This approach is often based on cellular automata modelling which generally requires considerable computational resource as the state of each entity must be individually modelled. The resulting evacuation plan is likely to be realistic however, as real-life factors can be accounted for easily such as traffic congestion and vehicle breakdowns. Meso-simulators take a similar approach, but consider groups of entities rather than individuals in order to reduce computational complexity (Barcelló and Grau, 1993); however advances in computing power have rendered this approach redundant for planning applications (Pidd *et al.*, 1996). In contrast, macro-simulators do not track the properties of single vehicles or people, but use equations originating from fluid flows in networks to estimate the state of congestion in the road network, thus they produce less realistic evacuation scenarios but require less computational resource (Pidd *et al.*, 1996).

Lammel *et al* (2010) designed a microscopic simulator for a scenario in which the Sihlsee Dam bursts and floods the city of Zurich in Switzerland. The system is based on CA simulation modelling where 100 iterations of the simulation are run in which each agent learns to optimize its route from experience gained in previous iterations. Road capacity is considered through the use of a queuing simulation, a time-constrained Dijkstra algorithm (Dijkstra, 1959) is used to plan every evacuee’s route to a single destination. For 165,000 agents in the model it takes 3 hours 24 minutes to run using a single Linux processor with 2GB RAM. The utility of such a system is clear for predictable situations such as a dam-burst for which the plan can be pre-computed. However, given a scenario such as a hurricane evacuation, in which the source location and spatial extents of the hazard are unknown until the period immediately preceding the event, the model will not run quickly enough to produce useful results and thus precomputing a number of likely scenarios may prove beneficial (Southworth, 1991). Kim *et al* (2008) argues that the macro-simulator approach is favourable because it scales well to large network sizes and that the significant runtime suffered by micro-simulators restricts their ability to compare alternative configurations in a timely fashion.

2.2.2 Real-Time Entity Monitoring

Our ability to monitor a range of phenomena in environments that were previously inaccessible is now possible due to advances in micro-electro-mechanical systems. Wireless sensor nodes are now small and inexpensive and so are relatively easy to install both densely and unobtrusively in remote places (Martinez *et al.*, 2004). Furthermore, the widespread prevalence of Global Positioning System (GPS) receivers on wireless sensors, in vehicles and in mobile phones enables us to monitor the location of moving entities such as people, vehicles and animals. The monitoring of animals is generally performed for the purpose of scientific research such as studying animal movement patterns (Moen *et al*, 1996) but the monitoring of people and vehicles enables us to improve our transportation infrastructure and surveillance systems.

Efforts to reduce congestion by influencing the route choices of drivers have so far focussed on the use of GPS equipped vehicles that are able to wirelessly share traffic flow information. This approach enables congestion to be reduced collaboratively and in an ad-hoc manner. For example, Dashitenezhad *et al.* (2004) designed a system in which traffic information is relayed between neighbouring vehicles as they pass each other on a road network. A unit is fitted to each participating vehicle which automatically joins an ad-hoc wireless network to broadcast and receive traffic information when they come in range of other similarly fitted vehicles. In this design, on-board routing systems use the additional traffic flow information to adjust their route, which is computed locally on-board each vehicle. A more centralised approach to data processing is used by the satellite navigation system manufacturer Tom-Tom in their system “Tom Tom One XL HD Traffic”. Location information provided by Vodafone UK is sourced from mobile phone owners and is aggregated at a central location and combined with information from in-situ road sensors to estimate traffic flow (Chu *et al.*, 2008). The traffic information is then published to subscribing in-vehicle Tom-Tom navigation systems. Google and Yahoo have also implemented similar systems. Aggregating traffic flow information at a central location ensures a large sample size and eliminates the need for on-board processing, although it does create a potential bottleneck if all users simultaneously request data from a provider's single endpoint. Nekovee (2005) suggests the data collected on-board vehicles could be fed into traffic forecasting models and traffic light control systems. However, it is noted that a computational grid would be required to aggregate, store and process the vast data volume generated by such a system.

Whereas traffic monitoring is concerned with monitoring the state of an entire road network or sub-network, vehicle tracking is simply concerned with tracking the state and location of a vehicle or a set of vehicles. The proposed ANGEL project provides an interesting vehicle guidance scenario in which the protection of a hazardous cargo is the primary concern. ANGEL forms a part of the Mitra Project (Planas *et al.*, 2008); its primary objective is the safe, secure, environmentally-friendly and cost-effective routing, navigation, tracking and tracing of vehicles. In this context, safe refers to journey planning that minimises the risk of road traffic accidents, and secure refers to the minimisation of vulnerability to terrorist hijacking. A multi faceted system is proposed in which the driver, the cargo and the environment are heavily instrumented and a number of risk factors are continually assessed. It is anticipated that the routing system will utilise vehicle mounted sensors to determine factors such as driver alertness, cargo condition and fuel range, in combination with external data sources such as live traffic information and real-time security alerts. This live data is to be combined with static base mapping data and fed into a continuous risk modelling process. The system would not only be able to determine efficient and safe routes but would also be capable of identifying safe places to stop. Such a system is envisioned to carry a heavy compute burden given the large number of risk factors to be considered. Furthermore, the size of this compute burden is liable to vary depending on the current size of the vehicle fleet that is being monitored.

Ghiani *et al* (2003) present a number of vehicle routing problems that can be considered variations of the classical travelling salesman problem. Each problem relates to a real-world routing application such as emergency services, taxi services, couriers and fleet management and is concerned with reducing cost and improving service level. Particular consideration is given to problems that are dynamic in nature; these are scenarios in which the input data such as travel times and demands depend explicitly on time and so prevent routes from being precomputed.

Monitoring people and their whereabouts is routinely carried out by government agencies for purposes of security. GPS tagging devices are commonly used in the criminal justice system to enforce bail terms such as curfews, and exclusion from particular areas (Black and Smith, 2003). Covert location monitoring can also be achieved by police forces using mobile phone pinging (Shields, 2006).

Furthermore, intelligent closed circuit television systems are also being prototyped that can identify persons of interest through face-recognition and searching a database of static images (Peacock *et al* 2004). It has been found that face recognition software that uses principal component analysis can outperform human face recognition (Burton *et al.*, 2001). Despite promising results in the literature there is a long way to go before this technology matures, as demonstrated by the fact that the most widely used benchmark database FERET (Phillips *et al.*, 1998) only contains 14,000 images.

2.2.3 Design Traits in Geospatial Monitoring & Prediction Systems

The host of real-time monitoring and prediction applications discussed thus far vary enormously in purpose and in terms of the ease with which they can be implemented. Furthermore, these applications differ in their suitability for a grid computing / sensor web approach to system design; important differences in system characteristics are discussed below.

Real-time systems can be divided into two major categories based on the importance of producing a result within a given time limit. Hard systems must meet a specific deadline to avert a catastrophe whereas information provided by soft systems is still useful after the deadline has passed (Kopetz, 1999). An example of a soft application is long term climate change studies as the results do not necessitate urgency. On the other hand, hard applications such as early warning systems, evacuation and post disaster management systems for geohazards including earthquakes (Kanamori *et al.*, 1997) landslides (Carrara *et al.*, 2000), wildfire (Goldammer, 2006) and floods (Hughes *et al.*, 2006) require results within a fixed time-frame. Hard systems are characterised by demanding response times and their ability to cope with peak-load conditions; short-term temporal accuracy of data takes precedence over long-term data integrity. In contrast, soft systems are generally designed to cope with average load conditions and are capable of extending their response time if they cannot cope with peak-load (Kopetz, 1999).

A trigger is defined as an event that causes some communication and processing action to begin (Tisato and de Paoli, 1995). A time-triggered event is caused simply by a change in time; for example in Japan’s REIS discussed by Nakamuru (2009) seismological observations are retrieved and processed every second for earthquake detection. In contrast, an event-triggered event is caused by the change in state of some property other than time; in Japan’s REIS the hypocentre location system is triggered in the event of earthquake detection. Typically, monitoring systems are time-triggered and prediction systems are event-triggered; such architectures are also referred to as pull and push-based systems respectively. Monitoring systems therefore present a steady but relentless data stream whereas prediction systems present an irregular data pattern that requires processing on demand.

The length of time a processing operation takes to complete is almost always related to the size of input data (Worboys and Duckham, 2004). This relationship between compute time and data input size is referred to as time complexity and is defined in terms of big-oh notation which gives an approximate indicator of how a given algorithm will perform. In this notation the processing time is defined in terms of data input size n. For example O(1) indicates that an operation will complete in constant time, i.e. is independent of data input size. Geospatial algorithms rarely execute in constant time, although some operations on geospatial data such as the insertion of records into a spatial database can be completed in constant time. O(n) indicates that there is a linear relationship between processing time and data input size. For example the calculation of a polygon's area has a linear time complexity as the processing time increases linearly with the number of vertices. $O(n^k)$ is referred to as polynomial time and indicates a polynomial relationship; for example Dijkstra's shortest path routing algorithm (Dijkstra, 1959) has a polynomial time complexity. Finally, $O(K^n)$ is referred to as exponential time and indicates the problem is intractable, i.e. no optimal solution exists. The travelling salesman problem (Schrijver, 2005) exemplifies an $O(K^n)$ geospatial problem with an exponential time complexity. Although the actual time taken by a given problem will depend on a variety of factors such as processing hardware and software, the time complexity gives a useful indication of how an algorithm is expected to perform in relation to its input data volume.

Resource scalability refers to the ability of a system to gain higher performance by increasing the size or number of processors (Hwang, 1996). In monitoring and prediction systems dynamic resource scalability may be required to cope with greater volumes of input data resulting from an increased spatial precision of analysis, number of sensors or size of study area. Increasing the size of a study area may bring an extra cost unrelated to the number of sensors as a larger volume of map data may need to be processed. Resource scalability may also be necessary if an increase in the accuracy or precision of output results is required. Tom Tom's XL One HD traffic monitoring system (Chu *et al.*, 2008) is an example of a system that may require the use of an increased number of processors to carry out observation aggregations as more sensors come online.

2.2.4 The Motivation for Integrating Grid Computing with Geospatial Monitoring and Prediction Systems

Successful grid implementations through projects such as EGEE (Gagliardi *et al.*, 2005), TeraGrid (Catlett, 2002) and CrossGrid (Marco and Marco, 2003) have shown that grid systems are particularly well suited to applications that involve significant computational modelling, the collaboration of multiple organisations or the integration of multiple data sources. In application areas such as fluvial flood monitoring and prediction, grid computing has already been incorporated into a number of systems including the GridStix project (Hughes *et al.*, 2006), the Data Fusion Grid Infrastructure (Kussul *et al.*, 2008) and the ANFAS/CrossGrid/K-WF/EGEE Flood Knowledge System (Hluchy *et al.*, 2005).

Besides performance improvements, grid computing also enables sensor based geoprocessing systems to increase or decrease their scale of analysis, either in terms of the number of sensor data streams being processed, the geographical extent of analysis or the precision of analysis. In this regard the computational grid has often been compared to the electrical power grid in its ability to make computational power available “on demand” (Foster and Kesselman, 1999). This property of elasticity is important for applications such as early warning systems as well as several traffic management and vehicle monitoring applications. However, the hard real-time requirement of such systems cannot currently be met by grid computing due to time lags in job scheduling systems (Padberg and Kiehle, 2009). Geoprocessing operations such as route-finding algorithms often resort to heuristic methods (Ghiani *et al.*, 2003) to solve computationally complex problems resulting from large spatial extents, fine scaled analysis or high multiplicity of observations. However, as the size of the analysis is increased, computational limits will eventually be reached for algorithms that have a time complexity exceeding linearity unless a scalable processing architecture is adopted (Openshaw, 2000). Grid computing offers a solution as it enables processing power to grow dynamically to meet an increased demand.

In the period immediately succeeding a natural disaster, both static and real-time geospatial data is in high demand from rescue organisations and from those responsible for repairing damaged infrastructure. Grid computing provides a common platform through which such organisations can collaborate and share resources (Follino *et al.*, 2010). In addition, the pool of services that results from the de-coupling of data resources, business logic and visualisation tools enables higher level geospatial applications to rapidly be created to suit changing circumstances (Kiehle, 2006).

Applications that have a large computational requirement such as microscopic evacuation planning may benefit from accessing high performance computing resources through the grid. Examples of parallel microscopic traffic simulators are provided by Cameron and Duncan (1996), Barcello *et al* (1998), and Nagel and Rickert (2001). Arguably it would be more convenient and efficient if the parallel computers used in these cases were located on the grid and accessed via a service interface. This would enable organisations wishing to run the traffic simulator to do so without hosting expensive high performance computing facilities. Furthermore this would enable better utilisation of computing power as it could be accessed on demand.

The key advantages of integrating grid computing with geospatial monitoring and prediction systems can be summarised as follows:

1. Access to computing on demand for applications exhibiting temporal variability in computational load.
2. Ability to scale-out analysis over a larger geographic area or at a finer spatial scale without hardware restrictions.
3. Ability to share data and compute resources across organisations.
4. Access to high performance computational capabilities.
5. Minimal initial hardware investment cost.

Consequently there is a considerable motivation to port the computational aspect of geospatial monitoring and prediction systems to the grid. From a technical perspective three major challenges are envisioned in integrating geoprocessing services and grid computing (Brauner *et al.*, 2009). The first of these is the architectural challenge of orchestrating services across the geospatial and the grid computing domains (Section 2.3). The second issue is the computational challenge of improving geoprocessing performance; geospatial datasets are characteristically large and geoprocessing operations are typically computationally intensive thus the use of parallel geoprocessing techniques is advocated (Section 2.4). The third research challenge is that of semantic descriptions for geospatial services to facilitate discovery and reconfigurable chaining; however this falls outside the scope of this thesis.

2.3 Distributed Computing Architectures and Standards for Real-Time Geospatial Applications

2.3.1 Web Service Styles and Standards

Web services, an implementation of SOA design principles have become the de-facto platform for distributed computing. SOA is described as "a paradigm for organising and utilising distributed capabilities that may be under the control of different ownership domains" (MacKenzie *et al.*, 2006). Parallels can be drawn between a SOA service and an object in object oriented programming in that the internal workings are hidden but a standard interface through which to interact with the object or service is made available (Worboys and Duckham, 2004). Because the service description is kept distinct from the implementation, SOA components using different technologies can interoperate through this common interface (Kaye, 2003). When a service is created its description is published to a searchable registry so clients can find their required service and bind to it directly (Figure 2.1). This enables clients, either end users or other applications, to interact with the service without any prior knowledge of it.

SOA has rapidly gained popularity as a software architecture and older distributed object systems have become virtually obsolete, largely as a result of their reliance on proprietary formats and their inherent communicational inefficiencies (Cook and Barfield, 2007). A critical reason for the success of the SOA is that it does not rely on sending entire objects around a network, instead only minimal requests and responses are communicated. Additionally, this architecture scales well and tolerates systems that cross ownership boundaries (MacKenzie *et al.*, 2006). Furthermore, the SOA enables existing services to be used as building blocks for new services that add some value or provide some original content (Foster, 2005).

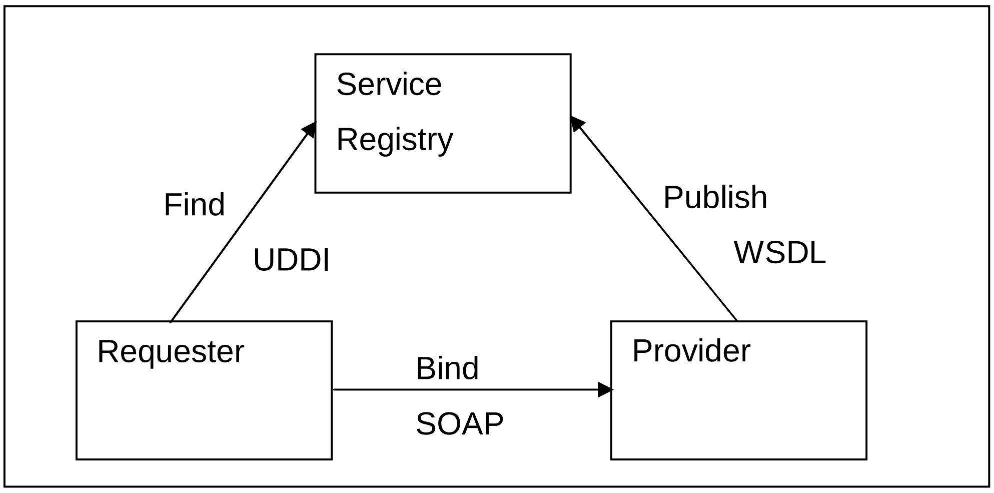


Figure 2.1: Service Oriented Architecture

Issues of interoperability between web services are the concern of Web Services Interoperability (WS-I) (http://www.ws-i.org); an open industry organisation chartered to establish best practices for web services interoperability for selected groups of web service standards across platforms, operating systems and programming languages. WS-I define profiles and implementation guidelines for web services standards. The WS-I's Basic Profile 1.1 (WS-I BP 1.1) sets out Web Service Description Language (WSDL) version 1.1, Universal Description Discovery and Integration (UDDI) version 2.0 and Simple Object Access Protocol (SOAP) version 1.1 as the core web services specifications which have been designed to standardise the processes of publishing, finding and binding to web services (Ballinger *et al.*, 2006).

In conformance with WS-I BP 1.1, service providers publish their services to a UDDI registry using WSDL. Clients are then able to locate these services by searching the UDDI registry, thus enabling the requester and provider to bind directly to each other using SOAP (Gottschalk *et al.*, 2002), a simple XML based protocol that lets applications exchange information over HTTP (W3C, 1999). This series of interactions is depicted in Figure 2.1. SOAP is a format for sending messages between applications via the internet and it is commonly used because it is text based so can easily pass through firewalls (Scribner and Stiver, 2000), because it is platform, language and vendor independent and because it is has been adopted as a W3C standard (Chatterjee and Webber, 2004).

In grid computing, web service technology is used to federate distributed resources using grid services which are defined as a web service that conforms to a particular set of conventions (Grimshaw, 2003). One of the major problems with using web services in grid computing has been that web services are stateless and many grid applications require the ability to store state, i.e. data values that persist across, and evolve as a result of web service interactions (Foster *et al.*, 2004). This difficulty has been overcome by the development of the Web Services Resource Framework (WSRF), a collection of web services specifications developed by the Organisation for the Advancement of Structured Information Standards (OASIS) that allow web services to store state. In the same way that SOAP based web services presume conformance with WS-I BP 1.1, grid services rely on the OGSA-WSRF Basic Profile v1.0 (OGSA-WSRF BP 1.0), a WS-I profile that extends the WS-I BP 1.1 whilst integrating WSRF specifications. WSRF web services are bundled with their associated resources; collectively this package is known as a WS-Resource which is addressed using an endpoint reference. Standard interfaces are defined to name and bind to resources, to create and destroy resources and to query resource properties. WSRF provides a means of describing and controlling the lifetime of a WS-Resource, of describing and handling faults systematically, of aggregating information about resources and services and of providing a notification mechanism to the change in state of resources (Czajkowski *et al.*, 2004).

Closely tied to the WSRF specifications is another family of OASIS standardised specifications, Web Services Notification (WSN). WSN incorporates three standards, WS-BaseNotification (Graham *et al.*, 2006), WS-BrokeredNotification (Chappell and Liu, 2006) and WS-Topics (Vambenepe *et al.*, 2006) which collectively define a framework through which web services can disseminate events (Niblett and Graham, 2005). Under the WSN architecture, notification producers publish their notifications to a topic which is a notional endpoint used to categorise notifications. Notification consumers can then receive notifications by subscribing to a topic. Through WSN specifications, web services can be invoked in reaction to events, thus extending the paradigm of a service-oriented architecture to that of an event driven architecture (Etzion, 2005). An example application of WSN is to notify a client each time a WSRF resource is modified.

WSRF specifications have largely become obsolete, due to a lack of uptake. A divide in the web services community became apparent when a competing set of specifications known as WS-Transfer (Alexander *et al.*, 2006) was introduced. WS-Transfer was championed by Microsoft and submitted to W3C for standardisation, thus causing uncertainty as to which specification set would prevail. Essentially, WS-Transfer provides the same functionality as WSRF though through a create, read, update, delete interface. Furthermore, WS-Eventing (Box *et al.*, 2006) provides a closely related notification framework to WS-Transfer, as WSN does to WSRF. A comparison of WSRF and WS-Transfer is conducted by Humphrey *et al.* (2005) who note only minor differences in the specifications but conclude that WS-Transfer is slightly easier to implement. The industry is now converging on a compromise between WSRF and WS-Transfer known as Web Services Resource Transfer (Davis *et al.*, 2009) that combines some features from each specification set. Due to the slow pace of progress in the standards community, the UK e-Science programme has ratified a core set of standards to enable current projects to move forward (Atkinson *et al.*, 2004). These are collectively termed WS-I+ and encompass WS-I, Business Process Execution Language (BPEL), WS-ReliableMessaging and WS-Addressing. BPEL is a web service orchestration language, WS-ReliableMessaging is a specification that is used to ensure the delivery of SOAP messages and WS-Addressing is a web service standard that incorporates message source and destination information into the SOAP envelope. Due to the present state of flux in notification and state representation standards, these standards were altogether omitted from WS-I+ although controversy does remain over the best way to model state in web and grid services (Foster *et al.*, 2009).

Foster *et al* (2009) argue that there are currently four different ways to model state in web and grid services. Explicit methods to model stateful resources as XML documents are provided by WSRF and WS-Transfer, each of which provides a number of common operations to access, update and delete such resources. Proponents of these explicit methods argue that it is logical to provide a standard interface for resource creation and management as it simplifies development, enables code reuse and encourages standard tooling to be developed. Another school of thought suggests that such conventions are overly complex and that state should be modelled implicitly through web service operations that are application specific; proponents of this approach value simplicity over structure. Finally, HTTP provides another method of coping with state based on principles that have become known as Representational State Transfer (REST). REST is an architectural style proposed by Fielding (2000) which describes a set of principles that outline how resources are defined and addressed, and provides an alternative to the WS-I based web service design. In REST terms, a resource is a communication endpoint that is addressed using a Universal Resource Identifier (URI) and manipulated through one of the four HTTP header operations: get, put, post and delete. Like the internet, REST web services support only these four methods but an infinite number of resources which are defined using a URI. In contrast, SOAP web services support a theoretically infinite number of methods, each of which corresponds to a port type. However, SOAP web services don’t provide direct access to resources; access is only provided through web service operations.

REST web services do not just provide a method of modelling state, they represent a fundamentally different style of web services which has lead to a long standing debate in the web services community over the relative merits and shortcomings of both RESTful and SOAP based web services. Those in favour of RESTful web services argue that the small number of methods ensure simplicity of design and ease of deployment; requests are self contained and do not require complicated sessions to be maintained with clients (Muehlen *et al.*, 2005). Furthermore, SOAP based services have traditionally been weak at addressing, meaning that it is not always apparent where a message is going, how to return a response or where to report an error. This weakness has been mitigated by the introduction of the WS-Addressing standard (Box *et al.*, 2004) which incorporates such details in the SOAP header, thus providing a standard way to route messages over multiple transports, or to direct a response to a third party. However, the heavy use of one URI as an endpoint through which a number of different services are accessed has been cited as a criticism of SOAP based web services (Muehlen *et al.*, 2005). Additionally, the heavy use of application specific methods that require encoding and decoding by higher level programming languages adds significantly to the communication overhead and overall complexity of SOAP based services. Conversely, proponents of the SOAP approach argue that it is preferable because it is not tied to the internet's HTTP transport protocol and it has better support for security features. It is also more suitable for publishing large complicated applications (Prescod, 2002).

The majority of OWS specifications predate SOAP and WSDL and a custom interface was therefore developed by the OGC based on the RESTful model. To improve interoperability with other web services the OGC is currently refactoring OWS to support SOAP and WSDL.

Listing 2.1 and Listing 2.2 give respective examples of SOAP and RESTful WPS *Execute* requests that specify the execution of a buffer operation; this is a simple geoprocessing operation that creates a new feature of a specified width around an existing feature. It can be seen that the SOAP wrapper details the method to execute "ExecuteProcess_GMLBuffer" and provides the input parameters, the URI of the polygon to buffer and a distance value representing the width of the buffer. The SOAP request assumes a connection to the WPS endpoint reference has already been established. If this endpoint reference offered any other methods, they too could be accessed through the same connection by specifying a different method name in the SOAP body. In contrast the RESTful request is made using a HTTP get request that specifies the address of the service, and the input parameters are provided as key-value pairs. The *DataInput* parameter is a URL reference to the feature to be buffered, and *BufferDistance* specifies the width of the buffer to be created.

Listing 2.1: SOAP WPS Execute Request

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<ExecuteProcess_GMLBuffer
xmlns="http://wpsint.tigris.org/soap/SpatialAnalysis">
<GmlUrlResource>http://onotta499/gml/polygon_gml.xml</GmlUrlResourc
e>
<Distance>10</Distance>
</ExecuteProcess_GMLBuffer>
</soap:Body>
```

Listing 2.2: RESTful WPS Execute Request

```
http://foo.bar/foo?request=Execute&service=WPS&version=1.0.0&langua
ge=enCA&Identifier=Buffer&DataInputs=Object=@xlink:href=http://foo.
bar/foo;BufferDistance=10& ResponseDocument=BufferedPolygon
```

2.3.2 The Open Grid Services Architecture (OGSA)

The OGSA framework specifies an extensible set of services that support the coordination and sharing of distributed computing resources. The core services specified by OGSA encompass execution management, data, resource management, security, self management and information. The role of OGSA is to facilitate interoperability within and between grid hardware, middleware and software services. OGSA is still a work in progress and many standards are not yet in place. Where possible existing web services standards are used or adapted which makes it easier for organisations that already support key web services standards to adopt OGSA. The OGSA framework is modular which enables basic capabilities to be mixed and matched to provide a higher level capability. This building block approach and the fine-grained nature of OGSA services ensures that only relevant parts of each specification need to be implemented.

Unlike the OWS and SWE frameworks a rigorous approach to security has been adopted in OGSA. A full description of the OGSA security model is provided by Nagaratnam *et al* (2002); in summary it addresses three major challenges; integration of disparate security systems, interoperability between distributed system components and the establishment of a trust relationship across organisational domains. Virtual Organisations (VO) address the trust relationship issue; VO members are granted access to their organisation's resources and each member is authenticated and authorised using their personal X.509 certificate (Cooper *et al.*, 2005) which is issued by a Certification Authority (CA). Typically each VO has its own CA and each CA is itself issued with certificates from a higher level CA. For example, a university department may have a CA which is issued with certificates by a university wide CA, which is in turn issued with certificates by a national CA. As every VO shares the same top level CA, the International Grid Trust Federation (www.gridpma.org), implicit trust can be assumed between each VO. Users that violate their trust agreement can have their certificate revoked and therefore lose access to the grid infrastructure. Security aspects of OGSA are outlined in the OGSA Basic Security Profile 2.0 (Snelling *et al.*, 2008) (OGSA-BSP 2.0) which is currently a recommended OGSA standard that extends the OGSA WSRF BP 1.0 and incorporates both the Secure Addressing Profile 1.0 (Merrill, 2008a) and the Secure Communication Profile 1.0 (Merrill, 2008b). Collectively these specifications set out a standardised means of overcoming the challenges specified above.

The OGSA data services architecture provides a means of moving data, running queries and updates, managing data replication and federating data resources in a grid environment (Foster *et al.*, 2005). The OGSA WS-DAI specification enables access to and integration, transformation and delivery of heterogeneous data resources through a web services interface. There are currently two realisations of this specification; OGSA-DAIX and OGSA-DAIR which allow access to and provide descriptions of XML and relational databases respectively (Antonioletti *et al.*, 2006). Using a WS-DAI service it is possible to perform data centric workflows, for example it is possible to access data from multiple sources such as relational and XML databases, transform the data and deliver it to another data repository. Furthermore, OGSA-DQP is an extension of the OGSA-DAI implementation that enables queries to be executed across resources federated by OGSA-DAI. Transporting data within a grid architecture is typically achieved using GridFTP (Allcock *et al.*, 2003) which can be considered an extension to the File Transfer Protocol (FTP) that has been designed specifically for grid data transport. GridFTP extends FTP with key features such as parallel and third party data transfers and critically it includes support for grid security measures. This protocol is best suited to transferring large files and is capable of scaling to network speeds, 27GB/s has been achieved on a 30GB/s bandwidth (Allcock *et al.*, 2005).

Basic Execution Service (OGSA-BES) (Foster *et al.*, 2008) is an OGSA standard for the submission of simple computational jobs; it specifies operations for the creation, monitoring and control of jobs or activities (Marzolla *et al.*, 2007). The individual activities or ‘jobs’ performed by OGSA-BES must be defined in a Job Submission Description Language (JSDL) document (Anjomshoaa *et al.*, 2005) which is an XML schema that has been adopted as an OGSA standard. JSDL is used to describe a job or activity in terms of a unique identifier, an application description, the resources it requires and the data files it needs.

Whereas the SWE and OWS specifications remain relatively stable, the OGSA specifications remain in a state of flux. Several of the specifications outlined in The Open Grid Services Architecture: Version 1.5 (Foster *et al.*, 2006) have been abandoned by the OGF and new ones have been proposed. It therefore seems likely that it will be several years until there is a complete set of OGSA implementation specifications. However various implementations of OGSA standards exist in the form of grid middleware such as the Globus Toolkit, and the standards that have been developed thus far are being adopted by the grid community.

2.3.3 OpenGIS Web Services (OWS)

OpenGIS Web Services (OWS) are a family of web service specifications defined by the OGC that enable maps, geospatial data and geoprocessing functionality to be discovered, accessed and visualised through the internet. As OWS provides a vendor neutral communication format it has rapidly been adopted by software and data providers. OWS is comprised of a set of independent specifications, each of which provides a particular function, although they each share a common design pattern including data structures, requests and responses. This common base is outlined in the OGC Web Services Common specification (Whiteside and Greenwood, 2010), currently in version 2.0.

OWS includes specifications such as the Web Mapping Service (WMS) that delivers visual map images in response to simple HTTP GET or POST requests. The Web Feature Service (WFS) (Panagiotis and Vretanos, 2010)delivers vector data in GML format and the Web Coverage Service (WCS) (Baumann, 2010) delivers raster data in a variety of common formats such as GeoTIFF. OWS also includes a catalogue service the Catalogue Service for Web (CSW) that enables geospatial data and services to be discovered. Furthermore, the Web Processing Service (WPS) (Schut, 2007)provides an interface through which geoprocessing functions can be carried out remotely.

All OWS publish a *getCapabilities* operation that returns a *Capabilities* document, an XML encoded description of what the service does and how to interact with it. The capabilties document is comparable to WSDL, and in the case of OWS with SOAP bindings, the *Capabilities* document is encoded as WSDL.

2.3.4 OpenGIS Web Services (OWS) Test bed Activity

Since 1999 the OGC have been running interoperability programs to design and develop geospatial web services. In the first initiative known as the Web Mapping Test bed some of the core OWS standards were developed including the WMS, WFS, GML, Filter Encoding Specification and the Styled Layer Descriptor. The next initiative, OWS-2 began in 2004 and introduced WSDL, SOAP and UDDI in the context of OWS and explored interoperability with these common web service specifications. Subsequent test bed activity have further developed and refined OWS. Notable activity of relevance to this thesis includes the introduction, definition and refinement of SWE, the development of an approach to manage OWS workflow chaining using BPEL, and integration of sensor device standards such as IEEE 1451 with higher level data services. In OWS-6 an event architecture for OWS and SWE was explored and grid processing in the context of WPS profiles was identified as a task area requiring work.

The current test bed OWS-7 is divided into 3 threads, sensor fusion enablement, feature and decision fusion, and aviation. Sensor fusion enablement follows on from the geoprocessing workflow and sensor web enablement threads of OWS-6 and is researching change detection from motion video using WPS, dynamic sensor tracking and notification, and best practice for integrating the Common Chemical, Biological, Radiological and Nuclear Sensor Interface with SWE.

2.3.5 Review of OpenGIS Web Services (OWS) / Grid Integration

Combining OWS that are geared towards the unique nature of geospatial data with core grid services that are capable of dealing with common distributed computing challenges such as security, information management and discovery is expected to provide a number of benefits to the geospatial community. These benefits include the enhancement of geospatial web services with security measures (Higgins, 2008) and a reduction in initial hardware acquisition investment due to increased sharing of computational and data resources (Padberg and Kiehle, 2009). Furthermore it is expected that integration of grid services with OWS will facilitate the chaining of geospatial workflows (Fairbairn *et al.*, 2008) and expedite the execution of large geo-processing operations by harnessing available processing capability from the grid (Koutroumpas and Higgins, 2008). Accordingly, there has been significant research activity in this field.

Particular attention has been drawn to the commonalities between geoprocessing services and grid computing and to the apparent benefits and challenges of integration. The principal similarity between OWS and OGSA in this regard is their common endeavour to define interface standards through which to enable open distributed processing. In OWS this is realised through the WPS and in OGSA through OGSA-BES and related specifications. Of significance to both frameworks is the ISO Reference Model for Open Distributed Processing (RM-ODP) that collectively defines a coordinating framework and methodology for distributed systems (Vallecillo, 2001). The RM-ODP specifications are subdivided into five viewpoints that each represents a different abstraction of distributed processing systems; enterprise, information, technology, computation and engineering. A key benefit of the viewpoint approach is to address a separation of concerns in the design of distributed systems. The OGC has adopted the RM-ODP viewpoints in its own OGC Reference Model (Percivall *et al.*, 2008) although the main focus of their service interface specifications are on the technology, computational and engineering viewpoints (Whiteside, 2005).

Fundamental differences in approach between grid services and the WPS in service description, service interface, security and statefulness present a considerable challenge in integration (Padberg and Kiehle, 2009, Padberg and Greve, 2009). Currently OWS suffer from a lack of security measures, as noted by Woolf and Shaon (2009a). This has prompted service providers to implement their own security controls at the client level resulting in interoperability problems in complex service chains. Conversely, grid services employ a comprehensive security framework based on public key cryptography. Although the OGC has proposed a security framework for OWS known as GeoDRM (Vowles, 2007) it has not yet reached maturity; as such there is no standard method of securing OWS at present. Another deficiency of OWS is that they are typically stateless and thus have limited ability to perform asynchronous workflows whereas grid services are capable of maintaining resource state through frameworks such as WSRF (Section 2.3.1). Various methods have been suggested to overcome the difficulties described here in order to integrate OWS and grid services.

According to Krüger and Kolbe (2008) OWS can be adapted to fit the grid environment, a process termed ‘gridification’ by either high-level or low-level means. Low-level gridification can be achieved by configuring a typical OWS to use the grid as a backend processing or data storage resource whilst maintaining its service description and interface. Conversely high-level gridification involves converting the OWS to a stateful grid service so that it can fully interact with other grid services; however an OWS proxy is required to ensure the service remains OWS compliant. Baranski (2008) demonstrates the low-level approach in the development of a grid-enabled WPS. The service extends the 52-North WPS and enables *embarrassingly parallel* tasks to be subdivided and processed in parallel on a back-end Unicore infrastructure after which the processed features are reassembled before the results are returned.

Further research into low-level gridification has been conducted by Woolf and Shaon (2009a) who highlight the overlap between the OGC WPS specification and the OGSA JSDL specification. Both specifications enable computational jobs and their process inputs and outputs to be described, however WPS lacks the ability to specify the computational resource requirements whereas JSDL lacks a web service interface. The specifications do overlap in some areas such as process description and process input and output. Woolf and Shaon (2009a) suggest embedding JSDL resource description parameters inside WPS *Execute* requests to specify computational resource requirements; three syntactical options are presented. The first option is to use a JSDL snippet containing the relevant computational resource requirements, the second option is to use a URL reference to a full JSDL document and the third option is to use key value pairs in an XPath style syntax referred to as micro-format style. It is suggested that interoperability could be improved through the definition of a WPS-grid profile containing for example the core subset of JSDL job description and resource description elements that form part of the HPC Basic Profile (Dillaway *et al.*, 2007), which has already been successful in facilitating interoperability.

Hobona *et al* (2007) provides an example of high-level gridification in their work on incorporating OWS into grid based geo-processing workflows. To solve the compatibility issues between OGSA and OWS an intermediary proxy web service was used to serialise and de-serialise SOAP messages to and from XML to allow OGC services to read them; essentially providing a SOAP interface to the OGC services. This approach proved to be problematic in that encoding the vector and raster payloads in SOAP messages placed a heavy demand on the Globus container. However, the problem was successfully addressed by storing the map features in a web server and simply returning the URL reference to the features inside the SOAP message rather than the features themselves. Lanig and Zipf (2009b) also describe a high-level approach to gridification for 3D terrain generalization of LIDAR data using Globus WSRF services although no results are detailed.

Friis-Christensen *et al* (2007) recognised inefficiencies in low-level gridification geoprocessing chains that involve the repeated sending of input data to a service instance to perform several related operations. The problem is that for each geoprocess the data is transferred from OWS to the grid for processing and the results returned to the client, and the process repeated for the next geoprocess in the chain, causing a great deal of data transfer. This style of chain is referred to as transparent and it occurs in processing chains based on the current WPS specification. As an alternative they suggest adapting the WPS specification to allow an ordered sequence of processes to be performed in which the output of one process can be used as an input to the next, a style they term translucent processing.

Krüger and Kolbe (2008) extend the concept of translucent processing to grid architectures; in addition to high and low level gridification they introduce a third style which they term transcendent gridification. This style is designed to reduce the overheads in complex geoprocessing chains resulting from unnecessary split and merge operations. Firstly a dataset is partitioned and each partition is allocated to a different processor. Secondly the first operation in the geoprocessing chain is performed. However, instead of reassembling the results after the first geoprocessing operation has finished, each set of processed features are passed directly to the next operation in the workflow. Finally the results are reassembled once the workflow has been completed. The advantages of this approach are not only that unnecessary split and merge operations are avoided, but also that information transfer can take place in multiple smaller and concurrent streams between services for the entire workflow. This style is thus suitable for multi-stage geoprocessing workflows on large datasets.

The ability to control processes asynchronously has been identified as an important requirement for efficient geoprocessing. Asynchronous services must equip each service call with a unique identifier so that results can be retrieved at a later time (Friis-Christensen *et al.*, 2007). Furthermore, asynchronous geoprocessing services need to provide the capability to check on the status of pending processes, and to pause or cancel processing jobs at any stage in their execution. Currently the WPS specification contains some basic functionality to store process results through the use of a unique job identifier. As yet the ability to control asynchronous processes is unsupported although pending change requests (Woolf and Shaon, 2009b, Woolf and Shaon, 2009c) make the case for additional enhancements to the WPS interface including a mechanism to check on the status of a pending process and the ability to pause or cancel processing jobs at any stage in their execution.

Besides these investigations into grid enabling geoprocessing services, other grid OWS integration work has considered OWS data and discovery services. Early work in this area was conducted by Di *et al* (2003) who attempted to broaden access to NASA's Earth Observing System (EOS) data repositories by integrating a pre web-services version of the Globus Toolkit with OWS data and discovery services. In their system OWS compliant WFS, WMS and WCS were exposed in addition to a catalogue service; no OWS discovery standard was available at this time. Requests to the catalogue prompted the search of a Globus information service that returned a physical file name which could then be used as a parameter in subsequent data retrieval requests from the OWS data services. More recently a grid-enabled CSW has been developed in addition to WCS and WMS portals that expose the typical OGC service interface at the front end whilst requests are forwarded to a mediator service at the backend that retrieves the required data from grid storage using a set of Globus based data services (Di *et al.*, 2008). The work proved to be successful; large volumes of NASA EOS data were made available to an extensive user community. However, it was found that the security and reliability afforded by the grid services suffered a performance penalty in terms of response time when compared to their traditional web service based counterparts.

Rather than exposing OWS to applications at the system's front-end, Shu *et al* (2006) take a different approach whereby grid-enabled OWS at the backend are federated using OGSA-DAI (Section 2.3.2) in conjunction with a mediator service to expose the data to applications. Another service OGSA-DQP schedules, manages and executes distributed queries on the OGSA-DAI resources, enabling the application layer to easily access all of the underlying data sources using a standard query interface. This architecture has been implemented for a case study based on a wildlife sighting database in Australia called WildNet.

In 2008 the G-OWS working group formed with the aim of integrating the gLite grid middleware, developed as part of EGEE (http://www.eu-egee.org), with OWS. Funded through the European projects GENESI-DR, CYCLOPS and DORII the group has so far developed a gLite enabled WCS and WPS, and implemented a shibboleth authentication method for gLite OWS. The group also plans to develop a gLite API for interfacing with OWS and a reference model containing procedures and guidelines for grid enabling OWS (Mazzetti, 2010).

Ghimire *et al* (2005) highlight a key problem facing distributed service architectures, the transfer of large datasets over limited network bandwidth. In addition to the integration of OWS and grid services, it is suggested that mobile grid services be developed which they describe as 'intelligent code wandering between grid nodes to accomplish certain tasks'. The idea of mobile grid services is to move the computation to the data rather than vice versa, as this is envisioned to reduce bandwidth use and thus improve performance. In the context of the OGC architecture, Friis-Christensen *et al* (2007) therefore suggest that data reducing processing operations be performed at the data source, a style they refer to as tightly-coupled geoprocessing. It is noted that the WFS specification already provides some basic processing capabilities such as coordinate transformation and the ability to clip features by specifying bounding box extents. Following this approach an extension to the WFS is suggested, the WFS-P that enables other data reducing operations to be carried out at source; examples include feature generalization and the calculation of summary statistics.

Müller *et al* (2010) extend the concept of processing data at source using the technique they refer to as 'moving code'. In contrast to data driven workflows whereby data is requested from an OWS data service and sent to a processing service, the moving code approach sends an algorithm to the data source to perform the processing. The main idea of the moving code concept is that the transference of large datasets can be avoided.

Four different moving code scenarios are presented:

1. The sent algorithm is tightly coupled to the data; data is shipped to the processing service with the code or is already known to the service.
2. The sent algorithm is loosely coupled to the data; data is retrieved through standard service interfaces at runtime.
3. The algorithm is deployed on the service prior to execution; data is shipped to the processing service at runtime.
4. The algorithm is deployed on the service prior to execution; the algorithm is repeatedly executable through a service interface that enables input data to be passed at runtime.

In the first case data is either already known to the service or is shipped with the code to the service at runtime, and the code is executed instantaneously but does not persist after execution. Sending a SQL query to a spatial database is an example of this scenario. Case 2 is exemplified by the prototypical transactional WPS that enables algorithms to be dynamically deployed and un-deployed via a service interface (Schaeffer, 2008). In case 2 data is passed to the service directly. Similarly case 3 describes a transactional WPS in which data is retrieved from OWS data services. In the fourth case, sent algorithm is deployed on the service prior to execution and can then be repeatedly executed. The WFS-P described by Friis-Christensen (2007) provides an example of this style.

The existing OGC architecture is focussed on the data driven approach rather than the moving code approach discussed here. However a prototypical implementation of the loosely coupled / permanent deployment scenario was created as part of the SoKNOS project (soKNOS, 2010) in which two decision support tools were developed as deployable algorithms; an assessment tool that determines the effect of an escaped gas on population centres, and a delineation algorithm that determines inundated areas from a flooding simulation.

The conclusions drawn from this work were that the moving code scenario is ideal for frequently changing algorithms or in situations where the same algorithm has to be deployed to several service instances simultaneously. Furthermore, moving code rather than data offers a significant performance improvement as it reduces the amount of data that must be shipped across the network. Conversely, the data driven approach is suitable for the one off execution of workflows and when the required simple operators are available at the data service level.

2.3.6 Review of Sensor Web Enablement (SWE) / Grid Integration

Despite its relative infancy there has already been considerable interest in integrating SWE with the grid infrastructure. This is unsurprising as sensor web and grid are both concerned with resource sharing across organisational boundaries, albeit from different perspectives.

SWE is comprised of a set of encoding languages and web service interface specifications that collectively define a framework for managing geospatial sensor data. The principal encoding languages in SWE are Sensor Model Language (SensorML), TransducerML (TML) and Observations & Measurements (O&M). As detailed by Botts *et al* (2007) SensorML is an XML language to encode sensor metadata that is capable of describing any sensor system and any data processing steps associated with the system. As it is an open and vendor neutral language SensorML eliminates the need for sensor systems to support multiple proprietary sensor description formats and facilitates the rapid integration of new sensors. TML (Na, 2007) provides an alternative sensor description language to SensorML. However, its primary concern is describing and transporting data close to the source, whereas SensorML addresses the higher level data processes including how to represent and display data. Both languages are self contained so that implementation of either one or the other is possible independent of the wider SWE framework.

Observations & Measurements (O&M) is an XML based encoding language for observations from sensor systems (Cox *et al.*, 2006). In contrast to SensorML that simply describes the sensor system, O&M provides a description of the actual sensor observations which includes the time and place of observation, the sensor system used to make the observation and the process chain used to derive the resulting measurement. It also describes the feature and the phenomena that is being observed (Cox *et al.*, 2006). Bermudez *et al* (2009) note that the Sensor Observation Service (SOS) provides a more robust interface than the WFS for providing time series data as a result of the O&M observation model that permits queries by observation, procedure and observed property as well as temporal and spatial queries. The basic O&M observation model is depicted as a Unified Modelling Language (UML) object diagram in Figure 2.2. It can be seen that each observation forms a part of a result, and includes a single procedure, observed property and feature of interest. The procedure refers to a description of the process used to generate a result; this is usually a sensor. The observed property describes the phenomenon being sensed and the feature of interest is the real world object that is representative of the objects target (Cox, 2007).

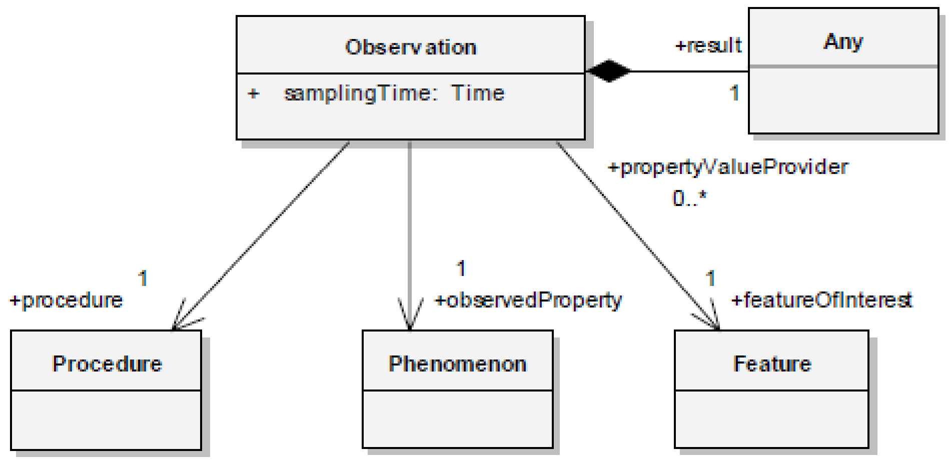


Figure 2.2: The O&M Observation Model (Stasch *et al.*, 2008)]

The key SWE services are the Sensor Observation Service (SOS) and the Sensor Planning Service (SPS). SOS is a service by which a client can obtain observations from one or more sensors/platforms. It essentially provides an Application Programming Interface (API) for managing deployed sensors and for retrieving their observations and aims to provide a standard means of access to all types of sensors and sensor systems, including remote, in-situ, fixed and mobile sensors.

SPS provides a service to manage sensors and sensor platforms. Given an instrument platform such as an orbiting satellite, many different user groups are likely to want to task it towards different areas of interest, and to configure it in different ways depending on the information they are trying to extract. The SPS enables the planning, scheduling, tasking, collection, archiving and distribution of data from sensor systems (Simonis *et al.*, 2007).

In addition to the SOS and the SPS a further SWE service was proposed, the Sensor Alert Service (SAS). This is an event notification system that is capable of notifying clients of sensed phenomena according to a specific set of conditions (Simonis, 2006). However, the SAS has not been formally approved as a SWE standard and it now seems likely that another candidate specification, the Sensor Event Service (SES) will supersede it (Everding and Echterhoff, 2009). The SES (Echterhoff and Everding, 2008) essentially acts as a notification broker to which sensors can publish their observations and from which clients can subscribe to receive observations, thus enabling push-based access to sensor data. Registered sensors push all of their observations to the SES, which then filters them according to client subscriptions. This ability to detect and react to events is considered crucial to the SWE architecture as it enables processing chains to be automatically invoked.

One of the most complete implementations of SWE is the Open Sensor Web Architecture (OSWA) which is under development by Melbourne University. It aims to integrate sensor networks and distributed computing to provide the ability to push heavy processing of sensor data to computational grids and to dynamically compose higher level services that incorporate real-time sensor data (Chu and Buyya, 2007). The proposed OSWA is composed of four layers; the sensor fabric layer which consists of the actual sensors, the sensor service layer consisting of services such as those detailed in SWE, a development layer that provides APIs to facilitate the creation of sensor based applications, and the application layer which consists of end-user sensor based applications.

Kobialka *et al* (2007) suggest the use of stateful web services as an improvement to OSWA in an attempt to enable multiple users to query the SOS and schedule SPS requests concurrently. Using this approach a new instance of each service can be created by the web service container for every request. Stateful web services have been introduced in the latest implementation of OSWA using Java WS-Core, the Globus implementation of WSRF. OSWA implements SOS and SPS but also extends SWE to include other services such as a sensor directory service which acts as a sensor registry. SOAP/WSDL bindings for each SWE service are provided to enable integration with other grid and web services. Additionally, other grid services are defined; a sensor data grid service which maintains replicas of sensor data, and a sensor grid processing service which collects and processes sensor data using the grid infrastructure. However, the grid services have yet to be implemented despite the grid enabled architecture design which includes the adoption of WSRF and SOAP/WSDL bindings to SWE services.

Since 2004 NASA has operated an Earth observing sensor web known as EO-1 that is capable of operating without human intervention. The EO1 project has focussed on monitoring volcanoes, floods, cryosphere, forest fires and clouds. Chien et al (2007) describe a scenario in which the the sensor web is used as an automated event detection tool. Low resolution satellite sensors (MODIS Terra / Aqua) imagery is continuously downloaded and analyzed via OGC web service interfaces and compared with previously captured time-series data of the same locations. If a significant change event is detected then higher resolution satellites are tasked via a SPS request to acquire further data in the given area. Example use cases that have been trialled for this system include the monitoring of sea-ice concentrations, the Mt. St. Helens volcano and the Mt. Erebus volcano.

In the case of the Mt. Erebus volcano automated analysis is also built into the sensor web routine. When the volcano erupts it is captured by in-situ seismic sensors. The sensors trigger a request to re-task a MODIS sensor to gain a better understanding of the eruption. All the data from the eruption is automatically downloaded at the NASA Jet Propulsion Laboratory where it is fed into a lava flow model. If the model finds anything unusual in the results then it requests further imagery of higher resolution from satellite based sensors to confirm the findings. This project demonstrates the power of sensor web for large scale environmental monitoring applications in which events that could otherwise go un-detected are being properly investigated using automated techniques.

GridCC (McGough and Colling, 2006) is an EGEE project which attempts to enable sensors as grid resources using the gLite middleware. In addition to existing computational and storage elements of gLite, GridCC introduces the instrument element (Frizziero *et al.*, 2006) which consists of a set of services to configure and control sensors remotely. The project deals with issues of information and monitoring, security, execution and planning of workflows related to sensors. GridCC has been designed with scientific instrumentation in mind and is concerned with providing collaborative access and control of such instruments to a virtual organisation. For example, it is to be used to control instrumentation in the CERN's Large Hadron Collider project. However, there is no attempt in GridCC to use OWS or SWE and although the architecture provides a secure environment for the sharing of sensor resources it doesn't consider the domain specific needs of geospatial users.

In comparison, Aloisio *et al* (2006) consider the information management aspect of sensor web and grid integration; they present an information service to monitor and discover sensor resources in a grid environment which uses an information model abstracted from SensorML. The iGrid monitoring and discovery service (Aloisio *et al.*, 2005) is loosely based on the Globus Monitoring and Discovery Service but has adopted a relational data model and this provides benefits such as the ability to query resources using SQL. Like the instrument element in the GridCC project, this service plays a pivotal role in integrating sensors into the grid environment as it enables sensor resources to be managed and discovered in the same way as other grid resources.

A gridification of the SOS using the Globus Toolkit is carried out by Kussel *et al* (2009) using a low level approach. Gridification of the SOS in this manner is expected to facilitate sensor discovery in a grid environment through the use of the Globus index service. Additionally the reliable transfer of large datasets can be achieved using Globus RFT, and security policy implementation can be more flexibly defined using the Grid Security Infrastructure. However, the authors were not able to create WSRF Resource Properties from SOS and O&M schema due to WSDL incompatibilities. Instead, service capabilities and sensor descriptions were stored as DOM elements which were serialized using custom bindings, thus enabling the service to return XML documents on request. However, XML elements could not be properly accessed in an object oriented manner.

The SensorGrid project (Tham and Buyya, 2005) addresses the specific shortcomings of coupling live geospatial data resources with high performance computing resources. Bottlenecks were discovered in the messaging mechanisms between sensors and applications. Firstly, generating and parsing XML was found to have a large time overhead. Secondly, HTTP was found to be too inefficient for systems requiring high performance and fast responses due to the request / response overhead and network constraints. As an alternative to sending XML based SOAP messages over HTTP, Narada Brokering is suggested; a content distribution infrastructure for voluminous data streams (Pallickara and Fox, 2003). Narada Brokering presents a novel messaging solution based on a peer to peer architecture. It enables scalable, efficient, secure and reliable messaging that is capable of passing through proxies and firewalls and that supports multiple transport protocols. Compression and decompression is provided for messages with large payloads and it is also possible to fragment very large files and re-merge them at the client side. In the SensorGrid project Narada Brokering is used in conjunction with a SOS to enable high performance data transfer between sensors and client applications. Better performance is achieved by eliminating the single direct connection between the sensor and the client which is a common bottleneck when dealing with voluminous messages. Instead, Narada Brokering routes messages via a network of message brokers and is thus capable of delivering messages at a greater rate.

As part of the NASA AIST ServoGrid project, Fox *et al* (2008) reached similar conclusions on the suitability of HTTP and XML as a basis for transportation and message encoding. The ServoGrid project attempted to use grid computing to integrate complex scientific applications with large data sets through a number of systems designed for earthquake simulation and prediction. Two of these systems, GeoFEST and Virtual California could be considered traditional parallel computing applications with an external but static data source, whereas other systems such as Pattern Informatics (Tiampo *et al.*, 2002) relied on a regularly updated data catalogue. The function of the Pattern Informatics system is to calculate probable regions of future seismic activity based on past and current seismic data; it uses a regularly updated WFS as a data source. Again, HTTP and XML were found to be too inefficient for non-trivial data transport. In addition to Narada Brokering, another potential solution is cited that could improve data streaming speed albeit at the expense of streaming initiation time. The proposed technique suggests two new web service specifications; WS-StreamNegotiation and WS-FlexibleRepresentation.

It is suggested that on initiation of a data stream, WS-StreamNegotiation messages are passed between the data source and sink to agree on the most efficient encoding and transport protocol that each actor can tolerate. Once established, the streaming is commenced on a different port using the fastest available protocols. Neither of these proposed standards have been developed but the same concept is used in Hand-held Flexible Representation (Oh and Fox, 2005), a software architecture for mobile devices that enables the source and sink to negotiate their preferred data representation.

Andrews (2007) observes that XML is generally unsuitable for encoding data; this is particularly the case for live data streams such as those produced by sensors. The main concern is that if every observation is wrapped in a set of tags the data rapidly becomes very voluminous. An additional problem for live data streams is that XML documents must be closed before they can be parsed or transported and this cannot occur until the data stream has ended. To some extent these issues are mitigated in SWE as it is possible to encode a block of observations in an O&M XML documents as a single element. Additionally, instead of providing observations inside an XML document it is possible to provide a reference to a data stream; these are referred to as ‘out of band’ observations (Cox *et al.*, 2006). However, the web services community acknowledge that the transmission of XML over the wire suffers performance overheads resulting from a large data volume, as well as data conversion and parsing and this is proving problematic for mobile applications and high performance parallel computing (Oh *et al.*, 2005).

The efficient encoding of XML data is an active research topic that has been discussed in detail by Chiu *et al* (2002) and van Engelen (2003) and is also the topic of a W3C working group (Goldman and Lenkov, 2005). Binary XML encodings are considered to be both faster to serialize and parse than their text-based counterpart and also less voluminous so they can be transported more efficiently. Binary SOAP attachments are usually encoded using MTOM (Gudgin *et al.*, 2005a) and XOP (Gudgin *et al.*, 2005b) although newer and more efficient formats are emerging such as Fast Infoset (Sandoz *et al.*, 2003) which is currently undergoing ISO standardisation. Further protocol specifications for binary data exchange are under discussion by the OGF.

The problems caused by heavy XML payloads are exacerbated by high traffic volumes at the data service which are likely to delay data delivery further. Havlick *et al* (2009) present caching and replication of SOS as a solution. It is argued that environmental monitoring data is ideally suited to this approach because archived environmental data doesn't change with time and because propagation is always from the data source to the replica, not vice versa. The proposed SOS-X service (Havlik *et al.*, 2008) automatically aggregates observations from one or more SOS thus increasing data availability. Furthermore, using this approach enables data providers to publish a controlled subset of data without having to implement complex security restrictions. To facilitate data replication Havlick *et al* (2009) recommend the following changes to the SOS and O&M specifications. Firstly, each observation should have an explicit and unique identifier so it is possible to differentiate between new and altered observations. Secondly, observations should contain a timestamp of the last data change and thirdly a mechanism should be provided to request deleted data. Finally, the SOS should provide a mechanism to publish the time-span for which each observation will be available, so the SOS-X can prioritise its data replication strategy.

As an alternative to channelling all of the data from sensor networks to the grid to be remotely processed, in-network processing is advocated by Gaynor *et al* (2004) using their Hourglass system. Hourglass aggregates data across geographically diverse sensors in order to obtain a global picture of the network's state. This approach is similar to that used by Cornell University in their COUGAR project (Bonnet and Seshadri, 2000) and is based upon the idea that in wireless sensor networks communication of data is several times more costly in energy consumption than computation. Distributed querying enables computation to be performed at the sensor nodes to return only an aggregation of observations. Although this approach overcomes the difficulties of high payload messages it does degrade the temporal resolution of observations through aggregation.

The potential of grid / SWE integration for purposes of disaster monitoring has attracted attention from a number of research projects. Fang *et al* (2009) propose a disaster relief system that facilitates a fair distribution of stockpiled resources amongst affected regions by channelling them to the worst affected areas. The perceived benefit of using grid computing for this purpose is that of collaboration between regional authorities as well as the sharing of data and computational resources. Chen *et al* (2010) present a wildfire detection system in which SWE data sources are chained to WPS processing services, using BPEL workflow orchestration language to detect hot pixels in EO-1 remotely sensed images. Interoperability, flexibility and re-usability are cited as the key motivations for using an open distributed architecture.

2.3.7 Cloud Computing

The academic community is beginning to show an interest in cloud computing as a means of reducing fixed costs for storage and data processing (Dikaiakos *et al.*, 2009). However, standardisation initiatives for cloud computing are only just beginning. In an attempt to reach early consensus on best practice the Open Cloud Consortium (www.opencloudconsortium.org) is championing an academic cloud test bed, the Open Science Data Cloud (Grossman *et al.*, 2010) while the OGF has begun work on developing the Open Cloud Computing Interface (OCCI), a standard API for cloud development.

Cloud service providers have presented several models of utility computing that each offer different levels of abstraction and resource virtualization (Armbrust *et al.*, 2009). The models can be broadly categorised into three major groups; Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). SaaS simply enables users to run software from their computer, on demand, without having to install it locally (Schwiegelshohn *et al.*, 2010); the SaaS model has been popularised by the widespread adoption of web services and SOAs. PaaS enables users to deploy their own applications onto a remote platform comprised of hardware, software and data access (Wang *et al.*, 2008a). IaaS provides a lower level of virtualization that enables users to deploy virtual machine instances over which they have almost complete control at the operating system level. Google App Engine (http://code.google.com/appengine/) exemplifies PaaS; it offers a limited software development kit in which to develop web applications that are hosted and managed by Google, who also handle automatic scaling of service provision as user demand levels change. The Amazon EC2 service (http://aws.amazon.com/ec2/) provides an example of IaaS in which users can deploy virtual machines instances into the cloud. Although IaaS offers a greater level of flexibility and control than PaaS it cannot offer indefinite scalability by continually porting the instance to a more powerful machine, or invoking more instances unless the application has been well designed for this purpose (Armbrust *et al.*, 2009). Conversely, PaaS such as Google App Engine provide a restricted API that forces developers to code in a shared-nothing style which facilitates elastic scaling (Abadi, 2009).

As noted by Baranski *et al* (2009) the cloud concept shares many features with the grid but there are some important distinctions between them. Both terms refer to a distributed computing system that provides data storage and computational power in a scalable fashion. However, the main target user group of grid computing is the scientific community with the purpose of running large scale simulations, whereas the cloud is targeted at small to medium sized businesses to enable scalability in web applications. A further distinction is that the grid infrastructure tends to be owned and funded by governments or research communities whereas cloud infrastructure is owned and operated by mainstream IT players such as Google, Amazon and Microsoft. Essential to grid computing is the concept of sharing computational resources amongst disparate organisations (Foster *et al.*, 2001). However, cloud systems are based on a model of utility computing in which service providers make a seemingly infinite pool of resources available on-demand and charge their users according to the quantity they consume.

Standards for IaaS and PaaS have only recently begun to emerge. A draft of the OCCI specification was released by the OGF OCCI-WG in December 2010 and is comprised of a RESTful API for managing the lifecycle of virtual machine instances. Future work is planned to create aggregators to enable existing infrastructure providers to adopt the interface. Concurrent work is being carried out by the Distributed Management Task Force (http://dmtf.org) to define an Open Virtual Machine Format, an open file format that will enable virtual machine images to be ported between cloud infrastructure providers.

2.3.8 Summary of Key Issues for OGF and OGC Standards Alignment

Grid computing, sensor web and distributed GIS technologies have reached a certain level of maturity. Version 2.0 of SWE and OWS standards has recently been realised and over a decade of development based on these standards has taken place. A memorandum of understanding between the OGF and the OGC was signed in 2007 which has resulted in significant collaboration on issues of interoperability between the grid computing and geospatial communities (Higgins *et al.*, 2008, Lee and Percivall, 2008). Despite significant headway in this regard there are several outstanding issues which remain to be addressed. These are summarised as follows:

1. The architectural challenge of integrating SOAP based and RESTful web services.

OWS and SWE versions 2.0 have incorporated SOAP/WSDL interfaces and this is set to facilitate interoperability of geospatial web services with grid computing services. The use of SOAP/WSDL bindings to OGC services is exemplified in the work of Hobana (2007), Shu *et al* (2006), Lanig and Zipf (2009a) and Kurzbach et al (2009). However, due to the complexity of the OGC schema, it has been found that the majority of standard web service toolkits have difficulty in parsing OGC WSDL documents in order to generate client web service stubs (Sonnet and Savage, 2003). Thus, integrating grid and geospatial web services into a unified workflow still presents significant difficulties.

2. The inherent lack of security in OWS and SWE

Security in the grid computing domain has traditionally been much stronger than in the geospatial domain. To this end the OGC GeoDRM working group has been developing a standard for digital rights management of geospatial data

(Vowles, 2007). Early work in this area took place under the SEE-GEO project (Higgins, 2008) that was specifically concerned with providing secure access to geospatial data on the grid. SEE-GEO addressed the role of Shibboleth, WS-Security and the Globus Grid Security Infrastructure in enabling secure access to OWS in a grid environment (Farnhill and McAllister, 2006). SEE-GEO activity was the foundation for the currently accepted practice of securing geospatial web services using Shibboleth, an open source software package that facilitates secure access to web content using a single sign-on (http://shibboleth.internet2.edu/). Shibboleth operates by redirecting a client to a shibboleth sign on page when they attempt to access a secured resource. If successful the client is then redirected back to the requested resource after logging on.

Matheus and Higgins (2009) have worked through some of the challenges in securing geospatial web services using Shibboleth such as the initiation of a login sequence by a web service client, and the establishment of a secure session without modifying OWS interfaces. This improvement in security is expected to significantly increase the availability of geospatial web services, as organisations will be able to publish data while controlling access to it. Although there are few implementations of secure geospatial web services to date, it seems likely that their prevalence will grow in the future.

3. Difficulties in complex service orchestration in SWE

Complex orchestration of OWS in a grid environment has been achieved in the SAW-GEO project (Fairbairn *et al.*, 2008) using Globus, Geoserver and the ActiveBPEL workflow engine. However, orchestration of SWE services in a grid environment is likely to encounter additional challenges. For example, issues such as how to incorporate relentless streams of data into a workflow must be addressed. This needs further research.

4. Messaging inefficiencies using SOAP and XML to encode and transport geospatial data.

The messaging inefficiencies of using XML and SOAP encodings (Tham and Buyya, 2005) are a significant obstacle to the coupling of live geospatial data to higher level applications. In terms of real-time systems this poses a big challenge, particularly as communication inefficiencies could negate the benefits of using a high performance grid resource. In-network aggregation may be a solution for some applications but it does result in the loss of much of the collected data. Messaging middleware such as Narada Brokering shows promise for reliable high performance message delivery in large distributed systems. This approach does however add an extra layer of complexity to distributed systems and is unlikely to be worthwhile for smaller implementations. Furthermore it is not yet clear whether this approach will be universally adopted. Other solutions such as tightly-coupled geoprocessing (Friis-Christensen *et al.*, 2007) and mobile grid services (Ghimire *et al.*, 2005) present alternatives that avoids the transfer of large datasets across the network, but for real-time sensor sources some data transfer is unavoidable.

5. Monitoring, discovery and general management of sensors in a grid environment.

A question is raised about how sensors should be managed in a grid environment. The approach taken by the GridCC project (McGough and Colling, 2006) is to simply consider sensors as grid resources and develop a set of services to manage them along the lines of other grid resources. Using this approach, sensors would be managed through standard grid services such as Globus Monitoring and Discovery Service. An alternative would be to grid enable the CSW; this approach would be favoured by the geospatial community as it provides an accepted interface and enables discovery of resources through SWE encodings and through spatial queries.

6. Immaturity of OGSA standards

Integration efforts are stalled by the relative immaturity of OGSA standards. The standards are rapidly emerging but many have not been approved yet; as a result there are few implementations to date.

2.4 Parallel Geoprocessing

2.4.1 Why Process in Parallel?

Geoprocessing refers simply to the processing of spatial data (Kiehle *et al.*, 2006) and is an integral part of most spatial information workflows which typically follow a three stage pattern of data acquisition, geoprocessing, and results dissemination (Burrough and McDonnell, 1998). Every operation that involves the manipulation of spatial data can be considered geoprocessing; it therefore encompasses a diverse collection of operations that include tasks from the fields of network analysis, spatial and geostatistics, image-processing, spatial analysis and generalization in addition to more commonplace tasks such as spatial and attribute queries and data or format conversions.

Processing spatial data is notoriously time-consuming and it is not uncommon for geoprocessing to present a bottleneck in spatial information workflows (Shi *et al.*, 2002). For example, Hawick *et al* (2003) found that interpolating only 100 points onto a 500x500 unit grid using kriging took over 10 minutes on a single processor. Commonly such processing delays can be attributed to either the size of the dataset or the complexity of the processing; an increase in either of these phenomena causes the processor to execute an increased number of instructions. Spatial data is often voluminous and there is a trend towards bigger, higher resolution datasets as data measurement, storage and processing tools continue to improve (Zhu *et al.*, 2009) and as our thirst for detailed spatial information continues to grow. Furthermore, the demand for data analysis capability is rising faster than the volume of data itself because algorithms are becoming more sophisticated and often carries time complexity above linearity (Gray *et al.*, 2005). As a consequence many geoprocessing operations are suitable candidates for parallel processing.

Parallel processing is a form of computation in which many calculations are carried out simultaneously with the goal of reducing the overall execution time (Almasi and Gottlieb, 1990). The amount of time saved by processing in parallel is quantified by Amdahl’s law (1967) which defines the metric ‘speed up’ (Equation 1) in terms of the number of processors and the proportion of the task that can be executed in parallel. In Equation 1, S_N is the speed-up achieved with N processors, T_s represents the fraction of sequential operations and T_p the fraction of parallel operations.

$$S_N = \frac{T_S + T_P}{T_S + \frac{T_P}{N}} \qquad \textbf{Equation 1}$$

Unfortunately parallel processing is difficult to implement for a number of reasons. Firstly, it requires the use of a parallel computer, defined as a set of processors able to work cooperatively to solve a computational problem (Foster, 1995). Parallel computers take many forms (Section 2.4.2) and usually require specifically tailored software. This brings us to the second difficulty; parallel software development is disproportionately labour intensive in comparison to its serial counterpart (Danelutto *et al.*, 1992). As a result the effort of implementing parallel code can only be justified under certain circumstances. Healey *et al* (1998) have identified three scenarios that justify the use of parallel code; compute intensive operations (Gittings *et al.*, 1994), operations that have data volumes so high they cannot be executed in serial (Lehning *et al.*, 2009) and finally operations that require a real-time response that cannot be met by a serial system (Xiong and Marble, 1996). Although many geoprocessing operations can be said to match at least one of these scenarios the ultimate decision as to whether to invest in parallel code must be made on a case by case basis. As the remainder of this Section will demonstrate there are a variety of tools and techniques available to exploit parallelism; different processing tasks can require dissimilar approaches to achieve a speed up.

2.4.2 Parallel Processing and Data Architectures

Flynn (1966) proposed a four group taxonomy of processing architectures; Single Instruction Single Data (SISD), Multiple Instruction Multiple Data (MIMD), Single Instruction Multiple Data (SIMD) and Multiple Instruction Single Data (MISD). The MIMD element of Flynn's taxonomy can be subdivided into shared memory architectures; Single Node Multiple Processors (SNMP), Multiple Node Multiple Processors (MNMP), and distributed memory architecture referred to as Multiple Node Single Processor (MNSP). MNSP can be further subdivided into high-speed Massive Parallel Processing (MPP) clusters and lower-speed Network of Workstations (NOW). The dominant architectures to emerge have been the SISD architecture which is typified by a standalone PC with one processor, and for distributed applications MIMD which refers to an ability to asynchronously perform multiple sets of instructions on different sets of data (Abbas, 2004). Figure 2.3 provides a graphical depiction of how Flynn's classification and the common MIMD architectures; MNSP, MNMP and SNMP are related.

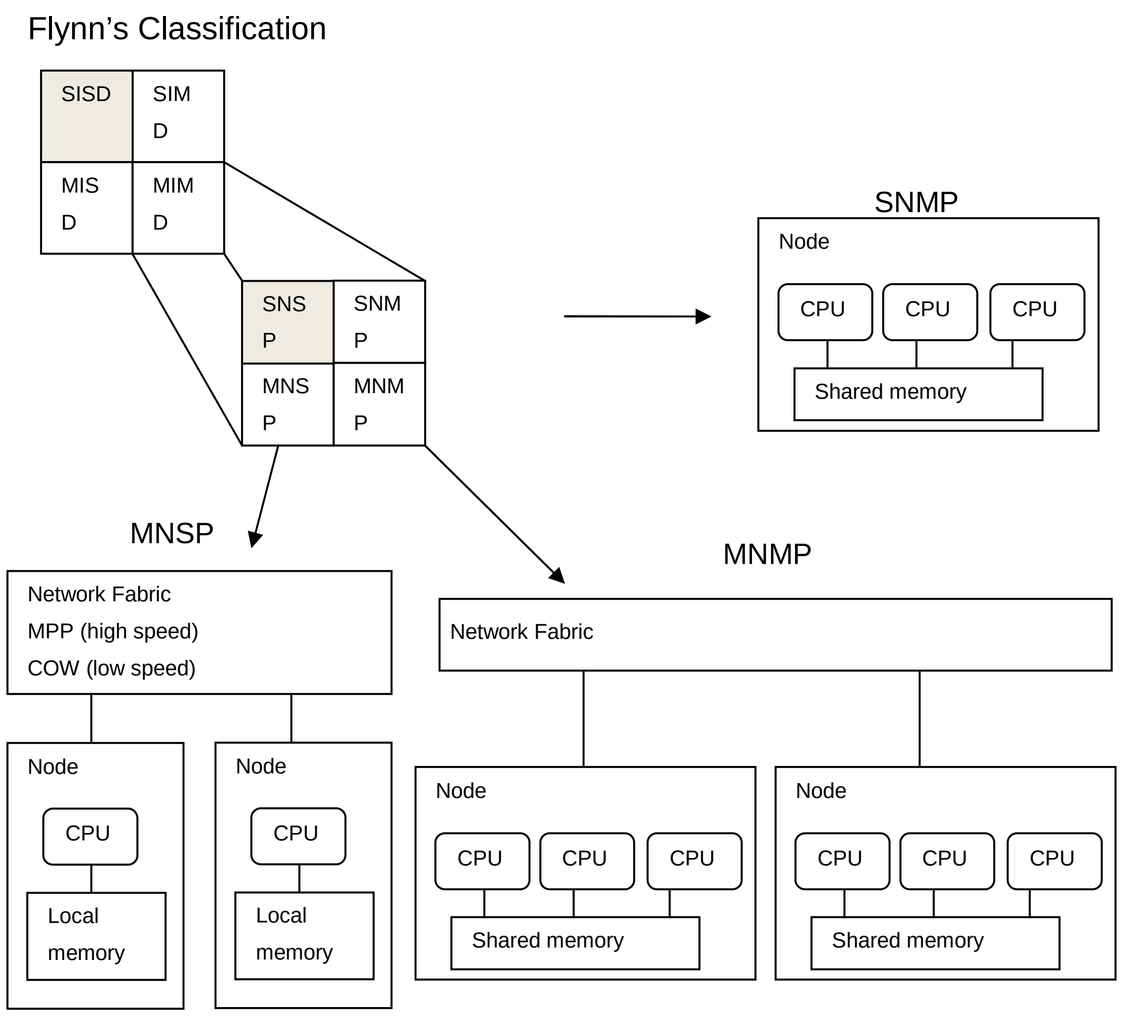


Figure 2.3: Flynn's Taxonomy and MNSP, MNMP, SNMP Architectures

Typical computational tasks are designed to run in serial on SISD architectures, i.e. they have a single thread of execution that is run on a single processor. SISD architectures are constrained in terms of scalability by the capacity of their processor and memory; as a system grows in size the cost of running a centralised architecture will eventually outstrip the cost of a distributed architecture. SISD architectures are therefore not suitable for very large or scalable processing operations.

In SNMP systems many processors share the same memory; this design was popular in early supercomputers but the shared memory aspect limits the ability of such systems to scale (Hwang and Xu, 1996). Scalability can be achieved in these systems by networking several machines together to form a MNMP system but the expense, lack of uptake and the need for different programming constructs has resulted in the majority of supercomputers using shared memory architectures to be phased out; MNSP architectures now dominate the tables of the world’s top performing 500 supercomputers, which shared memory systems haven’t entered since 2002 (http://www.top500.org).

By far the most common MIMD architecture is MNSP; which can be further subdivided into MPP and NOW. MPP and NOW are similar in that they are both affordable as they use commodity microprocessors and they both have distributed memory and can thus scale to hundreds or thousands of nodes (Abbas, 2004). The difference lies in that MPP clusters, often referred to as HPC clusters, are interconnected by high bandwidth low-latency connections. In addition their network interface connects directly to the memory bus rather than an input / output bus thus reducing latency further (Hwang and Xu, 1996). Therefore MPP can quickly exchange messages and thus run parallel programs whereas NOW is a lower cost alternative that uses standard commodity connections such as Ethernet (Hwang and Xu, 1996). The scalability, performance, and cost performance of MNSP have lead to their current monopolisation of the computing market for both the typical enterprise and academic market and for the specialised supercomputing market.

Fox *et al* (2008) make a further distinction between NOW that have high performance but non-optimised communication networks and distributed or grid systems. This distinction becomes pertinent when we consider distributing tasks over a loosely-coupled cluster. For example, in grid systems a cluster may consist of nodes that are distributed either geographically or amongst a number of organisations. Communication between these nodes is likely to be inhibited by network bandwidth and by message envelope overheads. The ‘rule of the millisecond’ (Fox, 2004) distinguishes these distributed systems from the aforementioned NOW and MPP systems. Many parallel programming constructs cannot tolerate latencies of more than a millisecond; typical messaging latencies in MPP are 20 microseconds, 100-1000 microseconds in NOW and 100 milliseconds in grid systems (Fox *et al.*, 2008). As a result, new programming constructs have been developed that enable parallel tasks to be run on distributed and grid systems; these are reviewed in Section 2.4.4 with reference to geoprocessing applications.

The major design goals of parallel processing architectures are also shared by parallel database systems; these are speed-up and scale-up, i.e. the ability to carry out processing operations faster and on larger amounts of data than their serial counterparts.

A simple taxonomy of parallel database architectures has been developed by Stonebraker (1986) that is comprised of three categories; shared memory, shared disk and shared nothing (Figure 2.4). Shared memory refers to a database architecture in which all processors share a common global memory in addition to their own private memory cache and have access to all the storage disks. Processors in the shared disk architecture also have access to all the storage disks but each has their own private memory, whereas processors in the shared nothing architecture each have their own memory and their own storage disk which only they can access.

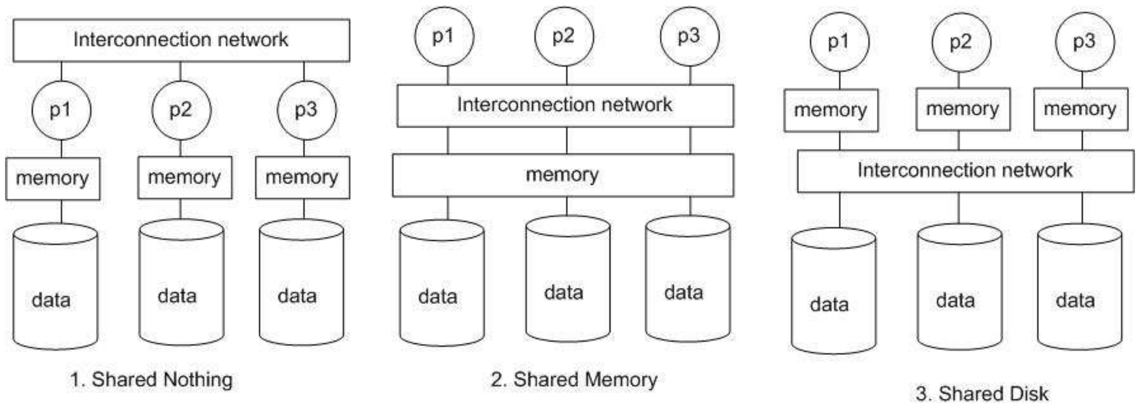


Figure 2.4: Parallel Database Architectures [adapted from Dewitt and Gray (1992)]

Relational DBMS have been widely used for storing and manipulating relationally structured data since their conception by Codd (1970) although it was not until the early 1990s that RDBMS were used to store spatial data (Adler, 2001). In recent years the massive data volume generated by large internet companies has prompted them to move away from storing data in relational DBMS. Google, Amazon and Facebook have each developed their own non-relational databases to store and analyse their vast quantities of data. Such databases typically conform to the shared-nothing architecture and because many of them do not use Structured Query Language (SQL) they have collectively become known as Not Only SQL (NOSQL) databases. The key advantages of such systems over their traditional relational counterparts are that they can handle non-structured data very efficiently, can easily be scaled horizontally and can scale write transactions more effectively (Leavitt, 2010).

These gains in scalability are largely achieved by sacrificing consistency. Gray (1981) set out the rules governing transaction processing which are conformed to by all major relational DBMS vendors. These rules are referred to by the acronym ACID which stands for Atomicity, Consistency, Isolation and Durability (Reuter and Haerder, 1983). Atomicity ensures that all operations in a transaction will complete, or the entire transaction will be rolled back. Consistency ensures that the database will be in a consistent state when the transaction starts and ends by enforcing integrity constraints. Isolation ensures that each transaction occurs individually without interference from other transactions, this is typically achieved by locking records for editing. Durability ensures that once committed a transaction will not be reversed.

Brewer’s CAP theorem (Lynch and Gilbert, 2002) states that it is not possible for a system to simultaneously provide consistency, availability and partition tolerance; at most two of these properties can be achieved. Availability refers to the number of simultaneous users that can access the system, whereas partition tolerance refers to the ability to split the system amongst multiple nodes. Single-site databases opt for consistency and availability, whereas distributed databases opt for consistency and tolerance to network partitions. Brewer suggests the BASE model as an alternative to ACID; it stands for Basically Available, Soft state, Eventual Consistency. Using the BASE model an optimistic approach to consistency is taken whereby partial transaction failures are supported (Pritchett, 2008). Thus if a transaction fails on one partition of a database, then it is still committed to the other partitions so maintaining some degree of availability. The transaction is eventually committed to the failed partition when it becomes available.

For the majority of spatial applications the commercially available parallel relational DBMS offers sufficient scalability in terms of data volume and number of concurrent users (Zhao *et al.*, 2005). However, NOSQL databases may prove useful for storing and querying massive volumes of spatial data. Existing implementations include key-value, document-based, column-oriented and graph databases, some of which include explicit support for spatial data types although with limited functionality. Currently CouchDB (http://couchdb.apache.org), MongoDB (http://mongodb.org) and Neo4j (http://neo4j.org) have facilities to create spatial indexes and to perform bounding box queries. However, even relatively simple spatial functionality such as the ability to compare geometry identities and perform intersection, distance-to and nearest neighbour have not yet been realised.

In NOSQL databases analytical processing other than simple queries is typically achieved using MapReduce, Dryad or a similar shared-nothing processing framework. Thus it seems likely that in the future NOSQL databases will include basic spatial tools written as MapReduce processes to support spatial data management.

2.4.3 Parallel Geoprocessing Strategies

Various attempts have been made to classify parallel programming paradigms in the literature and while a number of common themes are in evidence there does not appear to be a generally accepted classification. Fox (1989) presents four classes based on the temporal communication structure of the parallel program; synchronous, loosely synchronous, asynchronous and embarrassingly parallel. Synchronous refers a style of computation for which the same algorithm is run in parallel on a number of machines that share information at regular time steps. Conversely, asynchronous refers to a style for which different algorithms are executed on different machines and communication patterns between these machines is irregular and varies through time. Loosely synchronous is an intermediate between the two styles, for which machines synchronise with each other sporadically (Fox *et al.*, 1994) and embarrassingly parallel refers to an execution style for which no communication is required between nodes.

Healey and Desa (1990) present a different classification comprised of three classes; geometric parallelism, algorithmic parallelism and event parallelism. Geometric parallelism refers to the decomposition of the spatial domain, whereas algorithmic parallelism refers to a functional decomposition. Event parallelism is the simplest class presented here in which a master processor distributes tasks to a set of slave processors. Wilson (1995) suggests another classification which is based on decomposition technique, these are geometric, recursive, iterative or pipeline, functional and speculative.

Wagner and Scott (1995) identify three different decomposition strategies; control, domain and hybrid. Control decomposition involves decomposing the processing task into a number of constituent parts, each of which is assigned to a different processor. Conversely, domain decomposition is achieved by splitting the dataset up into a number of parts and assigning each part to a different processor. Hybrid decomposition uses both of these techniques and is particularly useful when a very fine-grained decomposition is required (Foster, 1995).

A further four classes are presented by Hansen (1993); pipelining, divide and conquer, master / slave (task farm), and cellular automata. Burkhart *et al* (1993) presents a more complex classification that encapsulates the approaches described thus far; it is based on the properties of the algorithm, its data and the inter-node communication patterns. Silva and Buyya (1999) identify the most popular paradigms as task farming, geometric decomposition, pipelining, divide and conquer and speculative parallelism. In the following text a review of the most relevant approaches to geoprocessing are presented.

Healey and Desa (1990) present event parallelism as the simplest of parallel strategies. Adopting this strategy, a master processor distributes tasks to a number of slave processors and reassembles the results when all the tasks have been completed. This approach of splitting, executing in parallel and then merging is often referred to as a task-farm (Bowler *et al.*, 1987) and the type of problem to which it is suited is referred to as embarrassingly parallel (Foster, 1995). The task farm application graph topology is outlined in Figure 2.5. As can be seen in the examples that follow, event parallelism is typically used to perform the same task on different data and thus its use can be considered a simple form of domain decomposition. The essential problem characteristic that permits this style of execution is a simple application graph topology in which each sub process can execute independently of the other processes.

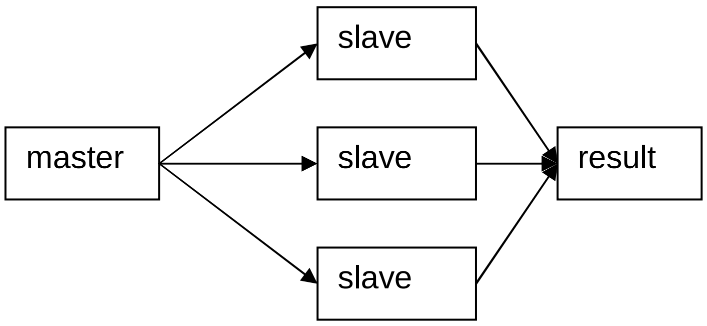


Figure 2.5: The Task Farm Application Graph Topology

There are a number of examples in the literature of using a task-farm approach for geoprocessing. Mineter and Dowers (1999) exemplify the use of a task-farm to process an atmospheric transport model, the Hull Acid Rain Model (Metcalfe *et al.*, 1995). Gong and Xie (2009) use a task-farm approach to extract drainage networks from large Digital Elevation Models (DEM), in this case the DEM is decomposed by watershed. Tehranian *et al* (2006) present a more complex system that combines event and pipeline parallelism for processing data from the Image Fourier Transform Spectrometer, a hyperspectral instrument that is scheduled for deployment on the GOES-R satellite and is expected to produce data at a rate of 130Mbps. The proposed processing system uses a task-farm approach to allocate incoming data to worker pipelines, which are comprised of a series of five processors each of which performs a stage in the processing of the raw inteferogram. Preliminary results show an almost linear speed-up. In each of these cases, the same process has been applied to different datasets in parallel to solve a data intensive problem.

A similar approach known as the Monte Carlo method (Metropolis and Ulam, 1949) is commonly used in simulation to translate uncertainty in a model's inputs, to uncertainty in its output. This is achieved by running the model several times with different input parameters and thus it fits with the task-farm paradigm. The Monte Carlo method is commonly applied to geoprocessing simulations such as estimating the probability of slope failure (Zhou *et al.*, 2003), estimating error propagation in seismic activity (Emmi and Horton, 1995) and carrying out flood simulations (Muzik and Chang, 1993).

Task farms can also be used at a finer granularity when dealing with very large datasets or computationally intensive problems by dividing a data aggregate into a number of constituent elements. For example, Baranski (2008) uses a task-farm approach to perform spatial intersection as a demonstration scenario for his grid enabled WPS in which different elements of the data aggregate are assigned to different processors. Hong-Chun *et al* (2009) apply the task-farm approach to the spatial filtering of a remotely sensed image and Xue *et al* (2005) use a task farm to calculate the Normalised Difference Vegetation Index from a MODIS satellite image. Hawick *et al* (2003) provide an example of classifying remotely sensed images in parallel using a task-farm hierarchy that enables a coarse-grained classification to be performed by assigning different images from a time-series to different processors, or a finer grained decomposition can be achieved by geometrically partitioning each image. A similarly fine-grained partition for image-processing is achieved by Nicolescu and Jonker (2002) using a task-farm style data decomposition in conjunction with a functional decomposition.

Equation 2 shows the total processing time of an operation using the task-farm approach to divide a dataset into as many segments as there are available processors (Hong-chun *et al.*, 2009). Total processing time is given by T_{all}, T_{cut} is the time taken to divide the dataset into as many segments as there are available processors, max(T_{proc}) is the maximum time taken to perform processing on the data segments and T_{merge} is the time taken to reassemble the data aggregate.

$$T_{all} = T_{cut} + \max(T_{proc}) + T_{merge}$$ **Equation 2**

From Equation 2 it can be observed that a significant speed up is possible using the task-farm approach, with a proviso that the time taken to split and merge the dataset is not considerable. However, parallel execution using the task-farm approach is not always viable; Trewin (1998) notes the following limitations of task farms. Firstly, inefficiencies can result from applications in which the time taken to compute a subtask can vary and is not known before execution. For example, if one sub-task takes substantially longer to complete than others, then several processors will remain idle whilst waiting for the final subtask to complete; in parallel database terminology this effect is referred to as skew. Secondly, the initial and final processes of splitting and merging can themselves present bottlenecks in execution and this prevents task-farms from scaling indefinitely to larger numbers of processes although in some cases this effect can be mitigated by using a slightly more complex application graph topology such as multi-source or multi-sink task-farms.

The divide and conquer strategy (Quinn, 1994) is an alternative approach to the task-farm that alleviates the load balancing problem described above; the procedure is outlined in Listing 2.3 and described as follows. The master processor divides the task into two subtasks which are each assigned to worker processors. If a subtask is found to be small enough it is solved directly, this is termed the *base case* in Listing 2.3. Otherwise it is divided into two and allocated to two more processors; this process continues recursively, forming a tree shaped graph topology. When the problem has been solved the results are passed back up the branch. The application graph topology of the divide and conquer approach is shown in Figure 2.6.

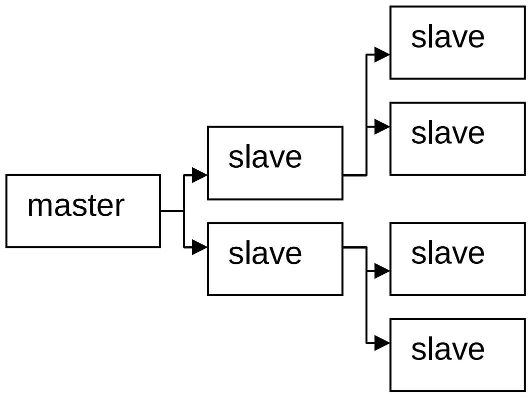


Figure 2.6: The Divide and Conquer Application Graph Topology

Listing 2.3: The Divide and Conquer Strategy (Foster, 1995)

```
procedure divide_and_conquer
begin
      if base case then
            solve problem
      else
            partition problem into subproblems L and R
            solve problem L using divide_and_conquer
            solve problem R using divide_and_conquer
            combine solutions to problems L and R
      end if
end
```

The divide and conquer strategy has been used to parallelise a 3D viewshed analysis (Katz *et al.*, 1991) and is a recognised technique for generating voronoi diagrams and performing delaunay triangulation; there are a number of examples in the literature (Aggarwal *et al.*, 1988, Davy and Dew, 1989, Cole *et al.*, 1990, Clematis and Puppo, 1993, Cignoni *et al.*, 1993, Ding and Densham, 1994, Wang and Tsin, 1987). To achieve a spatial interpolation on an irregularly spaced set of points, Wang and Armstrong (2003) exemplify a quadtree domain decomposition in which the spatial domain is recursively partitioned into four quadrants until a constant amount of information is held in each partition. A quadtree is essentially a divide and conquer approach in which a region is recursively subdivided into four equal sized blocks until each block is of the desired data volume (Samet, 1984). This approach was found to be a successful method of achieving an approximately equal load on each processor. It is noted by Magillo and Puppo (1998) that the divide and conquer approach is particularly well suited to coarse-grained MIMD architectures.

The MapReduce programming model (Dean and Ghemawat, 2008) can be considered a special form of event parallelism; in essence it is composed of two functions, *Map* and *Reduce*. The *Map* function takes a set of key-value pairs as input and produces a different set of key-value pairs as output. As such the *Map* function can be considered conceptually similar to task-farm data decomposition because in both cases a master processor subdivides a dataset amongst a set of slave processors. The key distinction is that the *Map* function operates strictly on key-value pairs whereas there is no such restriction for task-farm decomposition. Once the *Map* process is complete the resulting key-value pairs are allocated to *Reduce* processes which groups values with a common key, and outputs a list of values. These functions are formally expressed in Listing 2.4 and a diagram of the process is depicted in Figure 2.7.

Listing 2.4: The Map and Reduce Functions

```
map(key1,value1) → list(key2,value2)
reduce(key2,list(value2)) → list(value2)
```

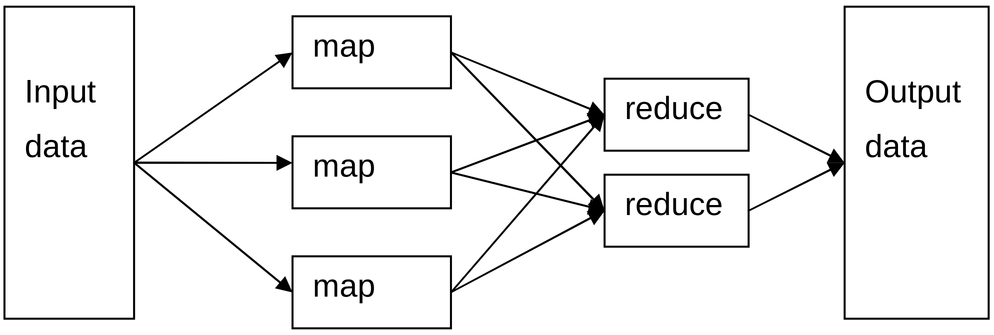


Figure 2.7: The Map Reduce Programming Model

MapReduce is a relatively recent phenomena; it was devised by Google to simplify the process of parallelising large data processing tasks such as the indexing of web pages. However, it has since proved popular for a number of applications including spatial data processing. Cary *et al* (2009) demonstrate the use of MapReduce in two spatial scenarios; firstly to bulk construct a set of R-tree spatial indexes and secondly to calculate quality metrics for aerial imagery. Chen *et al* (2008) developed a MapReduce based GIS workflow system, MrGIS, that is capable of performing raster algebra operations in parallel. MrGIS is based on the GRASS (http://www.osgeo.org/grass) open-source GIS software and operates by wrapping tools from the GRASS raster algebra toolkit as MapReduce jobs. Wu *et al* (2007) use MapReduce to determine road network alignment by combining satellite imagery and vector data.

Image processing algorithms often exhibit data independence and are therefore particularly well suited to the MapReduce approach. For example, Lv *et al.*

(2010) present a parallel implementation of the iterative unsupervised K-Means classification algorithm. For the K-means algorithm it is assumed that the number of land cover classes is known in advance but the spectral centre of these classes in *n*-dimensional feature space is unknown, where *n* is the number of spectral bands in the image. Initially an arbitrary spectral centre is selected and each pixel is assigned to its nearest centre. Subsequently the spectral centre of each class is changed to the mean location of all the pixels assigned to that class. The new spectral centre is used as the starting point for the next iteration. In the methodology adopted by Lv *et al.* (2010) a new MapReduce process is instantiated for each iteration. As the assignment of each pixel to a spectral centre can be carried out independently of the other pixels, this process is carried out inside the *Map* function. In terms of key value pairs the pixel's identifier (key) and digital number (value) are mapped to a pixel (key) and spectral centre (value). Pixels assigned to the same spectral centre are all sent to the same *Reduce* process which can then calculate the new spectral centre by averaging the position of all pixels, for each dimension.

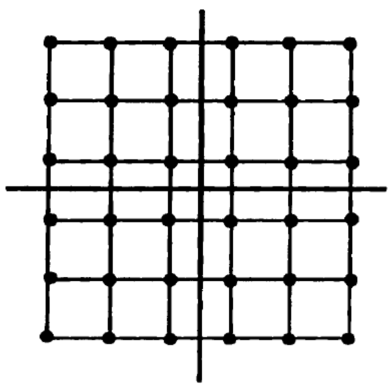
A similar implementation is presented by Li *et al.* (2010) for the ISODATA unsupervised classification algorithm. ISODATA does not require the exact number of land cover classes to be known in advance and extends K-means by taking into account the compactness of clusters which is measured using their standard deviation. This enables clusters with a standard deviation above a certain threshold to be split, and overlapping clusters to be merged. Li *et al.* (2010) use almost identical *Map* and *Reduce* functions to Lv *et al.* (2010), but extend these with another serially implemented *Refine* function that performs cluster splitting and merging.

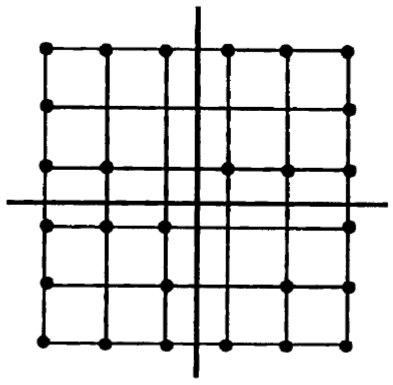
Another image processing example is described by Chapman *et al.* (2010) who use MapReduce to perform geo-correction. The cited example determines the ground location of each pixel sensed by a thermal infra-red satellite by implementing a geo-correction function (*Map*) and then averages the resulting temperature for each of the measured ground regions (*Reduce*). Despite the success of these implementations not all geoprocessing tasks can be easily transformed into the MapReduce paradigm, notably operations that involve relational joins or multi-stage processes (Cary *et al.*, 2009).

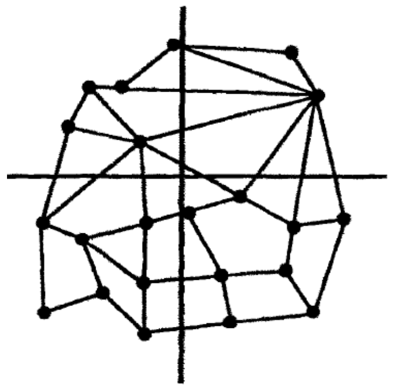
Geometric parallelism is another parallel strategy that relies on a specific form of domain decomposition in which the geographical data space is partitioned into sub regions, each of which are executed on different processors (Healey and Desa, 1990). In reference to geometric parallelism, Armstrong and Densham (1992) suggest that two characteristics of spatial domains, regularity and homogeneity are important in determining decomposition strategy. Furthermore, they suggest that domain decompositions fall into one of four classes, regular and homogeneous, irregular and homogeneous, regular and inhomogeneous and irregular and inhomogeneous. Regular refers to the spatial arrangement of data elements; i.e. a geometric partition would result in an equal number of data elements in each segment. Homogeneous refers to the data elements being of the same type, and implies that each data segment will require a similar amount of processing effort. Each of these categories are depicted in Figure 2.8 where it can be seen that although A and B both exhibit a regular grid, B contains an inhomogeneous arrangement of nodes.

Irregular domains are characterised by irregular mesh data structures such as Triangulated Irregular Networks (TIN), vector point, line and polygon data and vector network data. Using geometric partitioning it is difficult to achieve load balancing for irregular domains. However, through the use of quadtree partitioning and the divide and conquer strategy this difficulty can be overcome.

Regular domains are characterised by gridded data structures such as regular gridded DEMs and raster images. Regular and homogeneous domain decompositions are preferred because they are easily accomplished (Armstrong and Densham, 1992). Many geoprocessing tasks make use of such decompositions, particularly in the field of image processing (Hawick *et al.*, 2003, Plaza *et al.*, 2009), raster GIS (Wagner and Scott, 1995) and even tsunami wave modelling (Glimsdal *et al.*, 2004). Regular domain decomposition is most easily achieved by dividing a data aggregate into a number of segments comprised of contiguous data elements such as rows, columns, blocks or columns. However this is not always the case; Kidner *et al* (1997) use an equiangular data decomposition for a 360° line of s ight analysis in which 360/n degrees of data are assigned to each processor, where n is the number of available processors. Similarly, it is sometimes advantageous to perform scattered domain decomposition, particularly for scenarios such as ice sheet modelling (Mineter and Dowers, 1999) in which the majority of computation lies in certain spatial regions. Scattered decomposition involves decomposing the data into many more segments than there are processors and assigning each processor a number of segments from scattered spatial regions of the data aggregate (Trewin, 1998).


A. Regular and homogenous


B. Regular and inhomogeneous


C. Irregular and homogeneous

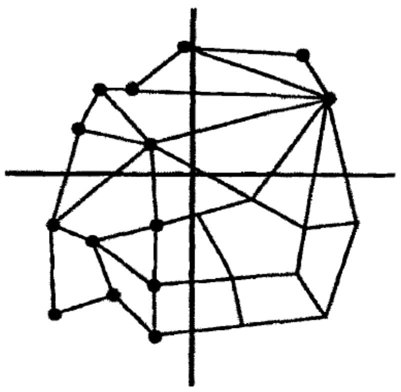

D. Irregular and inhomogeneous

Figure 2.8: Classification of Spatial Domains [(Armstrong and Densham, 1992)]

Using a geometric parallel strategy in a distributed memory environment necessitates the exchange between processors of data elements that lie on the boundary between neighbouring regions; this procedure is known as halo or boundary exchange. Lee and Hamdi (1995) present the parallel application of a convolution filter over an image that has undergone a regular domain decomposition. To successfully apply the filter to the entire image, pixels on the division boundary must be exchanged between processes. This is depicted in Figure 2.9; the image has been divided into nine segments each forming a separate process. In the figure every process exchanges each cell that forms a boundary with their neighbour, enough exchange to apply a 2x2 convolution filter. To apply a 3x3 filter, two rows of data would have to be exchanged. Other examples of boundary exchange in domain decompositions include the work of Lanthier *et al* (2003) on implementing a parallel version of the shortest path algorithm. The alternative is to use a shared memory architecture in which each processor already has access to the entire dataset. For example Hickman *et al* (1995) achieved an almost linear speed-up of texture based feature extraction from a remotely sensed image using a regular domain decomposition on a shared memory architecture.

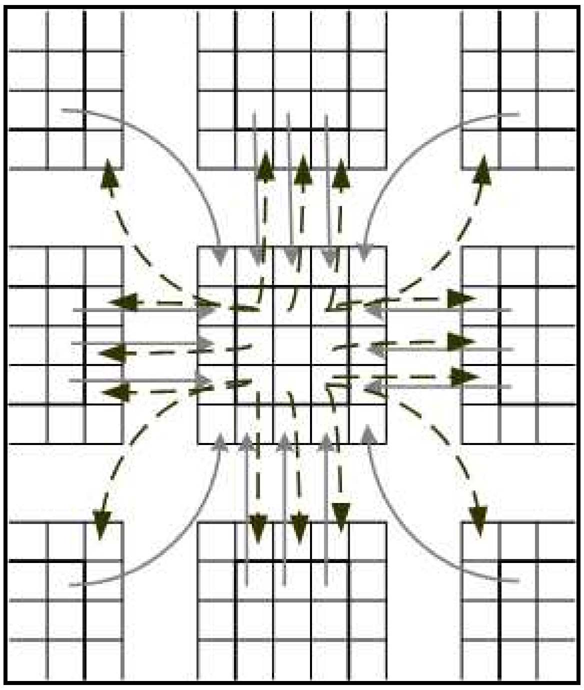


Figure 2.9: Boundary Exchange for a Convolution Filter

Because of the relative expense of communication in comparison to computation (Fox *et al.*, 1994), the exchange of data between processes is a common bottleneck in parallel programs. As such this style of parallel programming is best suited to MPP architectures in which dedicated high speed connections exist between processors, or shared memory in which all processors can access all the data. Performance can be increased for operations requiring boundary exchange by positioning neighbouring regions on adjacent processors (Bowler *et al.*, 1987).

Pipeline parallelism or ‘pipelining’ involves splitting an operation up into constituent stages, each of which is assigned to a different processor. Once the first processor has finished processing the first data item, it relinquishes control of this item to the second processor and begins processing the next data item, thus increasing overall throughput (Trewin, 1998). This style of parallelism is depicted in Figure 2.10; data is fed from left to right and an additional processing step is performed at each stage. Healey and Desa (1990) referred to this style of processing as ‘algorithmic parallelism’ and noted that whilst the concept is attractive, it is difficult to implement as each processor requires a different set of instructions. Furthermore, it is noted that fast interconnects must exist between machines in the pipeline, and to perform complicated workflows the dynamic reconfiguration of machine interconnects is a desirable feature. Another caveat of this approach is that the time taken to process each stage in the pipeline must be roughly comparable to maximise efficiency and avoid either idle processing time or the development of a processing backlog (Trewin, 1998). Finally, there is a limit to the scalability of this approach in that the maximum number of processors that can be employed is limited to the number of stages in the workflow.

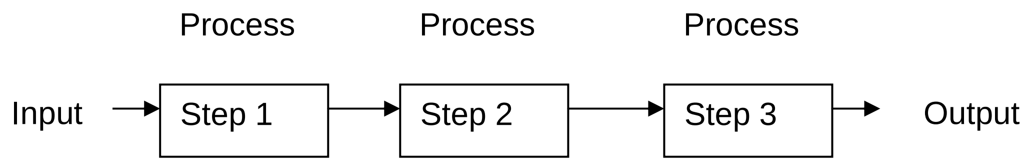


Figure 2.10: Data Pipelining Structure

Despite the preference to data decompositions in both the GIS domain (Mineter and Dowers, 1999) and more generally (Foster, 1995) for the reasons stated above, there are a number of scenarios for which pipeline parallelism is suitable. For example, the use of pipeline parallelism in conjunction with event parallelism for hyperspectral image processing has already been described above. Additionally, pipeline processing is commonly used to render large high resolution images (Bethel *et al.*, 2003) although it is noted by Sorokine *et al* (2005) that contemporary parallel rendering systems are not yet advanced enough for the GIS domain. Cited deficiencies include a lack of support for common geospatial data formats and an inability to render either cartographic symbology or more than one layer at a time. Kidner *et al* (1997) successfully used a 20 processor pipeline to obtain visibility indices from a DEM, achieving a speed-up of 12 (Equation 1). Koutroumpas and Higgins (2008) make the point that pipelining is the only valid functional decomposition technique to parallelise geospatial problems that exhibit flow dependence or anti-dependence. Given two tasks that are performed in a directed sequence, such as Step 1 and Step 2 in Figure 2.10, flow dependence exists if Step 1 modifies a data item that Step 2 reads. Conversely, anti-dependence exists if Step 2 modifies a data item that Step 1 reads. Thus in both of these scenarios it is not possible to execute Step 1 and Step 2 for the same data item at the same time, although pipelining can be used. Koutroumpas and Higgins (2008) describe a pipeline parallel geo-linking service that streams geographical features from a WFS and attribute data from a geo-linked data access service using an OGSA-DAI workflow. Data from each of these sources is combined and the geographic features are converted to a raster format and delivered to a GridFTP endpoint. Due to a lack of balance between the processing stages, the processing improvement was only three times better than a serial execution for this five stage process. However, this provides a good example of how pipelining can be used to increase the speed of geoprocessing workflow chains.

Glatard *et al* (2006) suggests another form of parallelism known as service level parallelism that achieves a speed-up in a cluster or grid computing environment. The actual process of submitting a job in a grid environment has a significant time overhead attached comprising of job submission, scheduling, queuing and data transfer. Service parallelism enables two or more sequential workflow tasks to be combined and submitted to the grid as a single task. However, service parallelism may negate any time savings if it limits any other form of parallelism (Glatard, 2008).

2.4.4 Parallel Programming Constructs

According to Foster (1995), there are three major parallel programming models; message passing, data parallel and shared memory. Associated with each of these models are a plethora of programming languages, compilers and standards; this Section provides a brief overview of these constructs.

Parallel programming languages are designed to simplify the process of developing parallel applications; to date there are several in existence that are capable of exploiting the parallelisation strategies outlined in Section 2.4.3. Parallelisation can either be achieved implicitly, using an auto-parallelising compiler, or explicitly using a parallel programming language (Hwu *et al.*, 2007).

Auto-parallelising compilers have been developed to exploit parallelisms inherent in sequential programs through automatic restructuring of the code. Typically this is achieved by searching for loops in the code in which there are no cross iteration dependencies and dividing them amongst available processors (Gupta *et al.*, 1999). Power Fortran Accelerator (Hogue and Graves, 1994) provides an example of a parallelising compiler that enables Fortran 77 code to run in parallel. Similarly the Sieve C++ compiler (Richards, 2006) enables C++ code to be run in parallel, although this can be considered semi-explicit as it requires code annotations which point to sections of code to be parallelised. The major advantages of implicit parallelism are that it enables legacy code to be implemented in parallel, and that it requires very little additional development effort. However, it is not as efficient as explicitly defined code; back in 1996 the NAS benchmark, a set of programs designed by NASA to evaluate parallel program performance (Bailey *et al.*, 1994) was found to run two to forty times faster using an explicit approach (Hwang and Xu, 1996). Recent improvements in parallelising compilers along with higher capacity hardware that is capable of checking equivalence to serial code has resulted in a renewed interest in this approach as a viable alternative to explicit methods (Hwu *et al.*, 2007).

Data parallel refers to a programming paradigm in which the same operation is performed on all elements of a data aggregate (Graham *et al*, 2005); as such it is suitable for speeding up the processing of large data volumes. Data parallel languages present an explicit method of developing parallel code for data aggregates, in which the developer is responsible for specifying the domain decomposition so the compiler can partition the computation accordingly (Foster, 1995). Fortran90 and High Performance Fortran (HPF) are both considered to be data parallel languages; the former is an official International Standards Organisation (ISO) standard whilst the latter, although more feature rich and widely supported, has no official status (Healey *et al.*, 1998). Notable features of HPF include the ability to specify abstract arrays of processors and the mapping of data array elements to these processors. Using the *ALIGN* directive it is possible to allocate specific data array elements to the same processor, thus if there is much interaction between these elements, inter-processor communication cost can be minimised. The *DISTRIBUTE* directive, enables a data array to be allocated as a *BLOCK*, i.e. to a single processor, or in a *CYCLIC* manner, i.e. consecutive elements in the array are to be mapped to different processors, thus exemplifying a scattered decomposition (Section 2.4.3). MapReduce can be considered an explicit data parallel approach as the same operation is applied to each element in a data aggregate at the Map phase. However, it also incorporates an element of functional decomposition as a task is split into two consecutive stages, Map and Reduce.

Whilst data parallel languages are useful in many circumstances, they are only suitable for reasonably simple tasks due to their single thread of control (Sawyer, 1998). Li *et al* (2008) exemplify the use of Fortran90 to perform the point in polygon operation and Douglas-Peucker line simplification. In addition they present a method of constructing a connectivity matrix between cartographic objects, as is required for a number of spatial analysis operations. Mower (1996) compares data parallel and message-passing techniques for performing line-simplification and concludes that data parallel is generally quicker than message passing if all processors are kept active, particularly as synchronous communication can adversely affect the performance of the message passing approach. Data parallel constructs have also been used to perform spatial interpolation on various architectures; Kriging on NOW using HPF (Hawick *et al.*, 2003, Kerry and Hawick, 1998), Kriging on the CM5 machine which is a MNSP supercomputer using CMFortran, a precursor to HPF (Hawick *et al.*, 2003, Kerry and Hawick, 1998) and MacDougall's (1984) interpolation algorithm on the Encore Multimax, an SNMP supercomputer using Encore Power Fortran (Armstrong and Marciano, 1993). We can conclude that data parallel constructs are useful for relatively simple forms of parallel processing where little inter-processor communication is required. In contrast, the message passing approach presents a more complex solution but one that enables more difficult application graph topologies to be executed.

Message Passing refers to a programming model in which processing operations are divided into a series of tasks that interact with each other by sending messages (Gropp *et al.*, 1999); the concept originated from the work of Hoare (1978). According to Sawyer (1998) a message passing system must provide the programmer with four types of operation; point to point communications, collective communications, process management and synchronisation primitives. Point to point refers to one processor sending a message to another, whereas collective refers to communications between the entire collection of processors such as broadcast operations in which one processor sends a message to all other processors, or reduction operations in which each processor contributes a value to an aggregate operation. Process management is used to commence and terminate processes, and synchronisation primitives are markers that one process sends to another to let it know that a certain point in the program has been reached.

The message passing model has become extremely popular as it enables processors in distributed memory architectures to synchronise with each other and to directly read and write to each other's memory (McBryan, 1994). Like the data parallel paradigm, processing operations using the message passing approach can be split up into tasks, each with their associated portion of data.

However, this approach differs from data parallel in that each task can request and receive pieces of data from each other at any stage in their execution, thus enabling significantly more complicated process topologies to be performed than is possible in the data parallel model, whilst remaining suitable for execution in a distributed memory environment. This ability of processes to communicate with each other during execution is known as inter-processor communication. Parallel tasks are often described in terms of their granularity, a ratio describing the size of computation that can be performed by a processing node between communication or synchronisation with other processing nodes (Wilkinson and Allen, 1999). Coarse-grained tasks can perform a lot of computation before communication with other nodes is required; this is generally desirable due to the relative expense of communication in comparison to computation (Fox *et al.*, 1994).

A popular single standard has emerged for message passing programs (Booth *et al.*, 2003) which is known as the Message Passing Interface (MPI). Unlike the other languages described here, in message passing applications the communication between processes must be explicitly coded. MPI offers both a standard communication interface, and an API that enables parallel message passing programs to be implemented in C, C++ and Fortran with some degree of platform portability. Messages can be sent both point to point, using the *MPI_SEND* command, or collectively using the *MPI_BROADCAST* command. However, despite its flexibility, developing message passing programs is difficult. Firstly, when using blocking communication deadlock is a common problem; blocking refers to a style of communication in which process B waits for a message from process A before continuing execution. Thus dead-lock occurs when processes A and B are both waiting for messages from each other and neither can progress. Secondly, there is no easy way to debug an application that is running on several processors at once (Samofalov and Konovalov, 1996).

Fox's 'rule of the millisecond' (Section 2.4.2) deems MPI applications suitable only for MPP architectures. However, a number of alternative MPI implementations have appeared that tolerate greater communication latencies.

For example, MPICH-G2 (Karonis *et al.*, 2003) and PACX-MPI (Keller *et al.*, 2003) make it possible to run parallel jobs on distributed grid resources as though they were a tightly coupled cluster. PACX-MPI has adopted a two-tier programming model, one for internal intra-cluster communication and one for inter-cluster communication. MPICH-G2 however appears to the programmer as a standard MPICH implementation of MPI but it has been developed specifically for the Globus middleware and can be used in conjunction with Globus' security, resource management and job submission services.

Dattilo & Spezzano (2003) provide an example of a message passing processing operation that demonstrates the increased level of complexity that the paradigm can withstand. They describe how a problem solving environment called CAMELOT is used to run a Cellular Automata model that simulates the debris and mud flow of a landslide. The Cellular Automata approach is useful to model flow as it captures the collective effect of several locally interacting components. A Cellular Automata model is comprised of a grid of cells, each with a state and a set of transition rules that define how the state will change, based on previous states or the state of neighbouring cells. In this case several properties of the debris are considered; altitude, thickness, run-up, depth, mobilisation, outflow direction and water content. The simulation is implemented in parallel using a high-level language CARPET, which uses an underlying message passing approach in which every cell is represented as a process. At each time interval the interactions between cells are managed by message passing between processes.

Other geoprocessing applications that utilise MPI include parallel image rendering systems (Sano *et al.*, 2004, Sorokine *et al.*, 2005) and hyperspectral image processing. Plaza *et al* (2009) in their work on clustering, classification and spectral mixture analysis of remotely sensed hyperspectral data use MPI C extensions to implement their algorithms on both heterogeneous and homogeneous NOWs. Conversely Tehranian *et al* (2006) deemed a generic MPI implementation unsuitable for the constraints of a real-time system in terms of reliability and availability and thus opted to use a non-standard software framework to implement their hyperspectral processing system.

The Shared Memory model uses a different approach to parallelisation; it relies on a processing architecture in which several processors have access to the entire dataset (Foster, 1995). Parallelisation is achieved by multiple processors simultaneously processing different parts of the same dataset and locking mechanisms are used to ensure conflicting read and write operations are not imposed on data elements.

Shared Memory programming is commonly realised using OpenMP (http://www.openmp.org), an open standard for shared memory parallel programming. OpenMP provides an API that enables developers to easily write code for multiprocessor shared memory architectures (SNMP and MNMP) in the Fortran, C and C++ languages. Implementing shared memory code is easily achieved due to a global address space from which each process can access all the data. As such sequential code can be parallelised for shared memory execution with relative ease (Sawyer, 1998). An alternative approach to shared memory programming is to use a message passing library (Sawyer, 1998) which is also capable of running on a shared memory architecture.

There have been a number of efforts to simplify the use of parallel programming constructs through high level abstractions. The MPI standard itself contains some abstraction such as collective communication functions like *BROADCAST;* before MPI, CHIMP (Bruce *et al.*, 1993) provided some of this functionality. In addition there have been domain specific efforts at producing parallel libraries such as the image processing library described by Seinstra *et al* (2002) which attempts to mask parallelism from the user.

As described by Trewin (1998) the Parallel Utilities Library is the most comprehensive of these libraries; built on the MPI standard it consists of a number of C and Fortran 77 modules that can be harnessed to perform basic parallel operations. The modules are categorised into paradigm specific, domain specific and non-specific. Whilst non-specific modules include generic tools to perform tasks such as parallel IO, the paradigm specific modules include tools specific to a parallel strategy such as task-farming (PUL-TF) or regular domain decomposition (PUL-RD). Finally, domain specific modules are targeted at specific applications, for example PUL-SM provides a basis for decomposing irregular mesh data structures. Mineter and Dowers (1999) provide an example of using PUL-RD to decompose an image to perform a moving window noise reduction filter. PUL-RD handles the splitting of data amongst processors, halo exchange and reassembling of the results.

2.5 Summary

The goals and structure of this literature review were set out in Section 2.1. In Section 2.2 the key characteristics of Earth systems monitoring and prediction applications were reviewed in order to establish the computational and data requirements of such systems and to evaluate the suitability of grid computing as a processing resource. Five key motivations for integrating grid computing were identified; the ability to access computing on demand, the ability to scale-out analysis without hardware restrictions, the ability to share resources across organisations, the ability to access high performance computing resources and a minimisation of initial hardware investment costs.

Section 2.3 describes the current status of generic web service standards, and reviews existing work on integrating web services across the grid computing and geospatial domains. Middleware standards in OWS, SWE and OGSA are discussed and incompatibilities between these specifications are considered. Furthermore, strategies to align these specifications are reviewed and a list of key issues in the alignment of standards is presented.

In Section 2.4 parallel processing architectures are described and existing work on parallel geoprocessing is presented. Furthermore, tools and compilers for parallel processing are also briefly discussed. The key outcome of this Section is a presentation of geoprocessing strategies.

It can be concluded that grid based geoprocessing is a vast and rapidly expanding field in which there has been a lot of recent work. Considerable progress has been made in identifying and overcoming the architectural challenges of integrating grid and geospatial web services such as OWS and

SWE although there appears to be a general consensus that more geospatial grid implementations are required to fully understand the remaining challenges.

A similar research effort has gone into parallel geoprocessing. The majority of this work was carried out in the 1990s since which time the interest in parallel geoprocessing, and parallel computing in general has declined due to advancements in processing hardware and an overall trend away from high performance computing ‘scale up’ approaches towards a high throughput ‘scale out’ approach as realised in grid and cloud based systems. However, the processing strategies remain highly relevant and renewed interest in parallel geoprocessing from the perspective of real-time processing remains pertinent.

A number of individual real-time geoprocessing systems are described in the literature that have been developed for specific applications but it appears that to date there has been a failure to consider real-time geoprocessing from a more generic perspective. Thus it is the aim of this thesis to fill this research gap by attempting to identify the broad categories of real-time geoprocessing operations and determine the relevant strategies for their implementation. The aims, objectives and major research questions of this thesis are therefore set out in the following Section.

2.6 Aim, Objectives and Research Questions

Aim:
To develop an appropriate conceptual and implemented framework in which open standards in grid computing, sensor web and geospatial web services can be combined as a technological basis for monitoring and prediction of geospatial phenomena in the Earth systems domain, to facilitate real-time decision support.

Objectives:

1. Describe the current and emerging standards in sensor web, grid computing and geospatial web services that are relevant to the integration of large scale geospatial processing operations.

2. Investigate the difference in approach to standards development across the geospatial and distributed computing domains and the impact these differences have on geospatial workflows. Suggest areas where such workflows can be improved.

3. Assess the design of existing monitoring and prediction systems reliant on computationally intensive processing of geospatial data in the Earth systems domain.

4. Develop an initial prototypical categorisation of geospatial processing algorithms for both static and real-time data.

5. Design standards-based middleware to seamlessly incorporate real time sensor data into distributed geospatial processing operations within a web services environment.

6. Design and develop use cases for a real-time distributed geoprocessing framework that are exemplar of each algorithm category specified in objective 4. Test and evaluate each system.

7. Propose frameworks and areas for future research and development and suggest areas where existing standards need to be augmented or improved.

By fulfilling the aim of this thesis and addressing the objectives it will be possible to evaluate several core research questions in relation to this work. These research questions are listed as follows:

To what extent can standards in geospatial web services, sensor web and distributed computing be integrated within a geoprocessing context?

This is a natural starting point for this project. Further work in this project is dependent on the extent to which these technologies can be aligned using existing and emerging open standards.

What are the potential bottlenecks in a distributed real-time monitoring and prediction system in relation to distributed geoprocessing?

Bottlenecks in data transfer and processing are inherent in real-time monitoring and prediction systems. It is important to identify the stages in the workflow that are constraining each system in order to streamline each system and make it fit for purpose.

Are there any generic methods of distributing real-time geoprocessing operations?

One of the most interesting outcomes of this research project will be whether a family of methods can be developed to distribute static or real-time geoprocessing operations amongst several processors in a grid or cloud architecture.

Chapter 3 Categorisation of Real-Time Distributed Geoprocessing Paradigms

3.1 Introduction

A major goal of this thesis is to explore how grid computing can be used in conjunction with sensor web to assist in the monitoring of spatially complex systems, processes and activities. It is expected that the primary function of grid computing in this regard is the provision of a pool of computational resource that enables geospatial data to be processed in a timely fashion. However, given the diversity that exists amongst geoprocessing scenarios and algorithms it is difficult to ascertain the most effective method of integrating grid computing into geospatial processing workflows.

In the previous Chapter the motivation for integrating grid computing with geospatial monitoring and prediction systems was asserted and the technicalities of integrating web service technologies from different domains were explored. Furthermore the array of tools and techniques for performing geoprocessing operations in parallel was also evaluated. However, much of what can be concluded from the technical discussion in Chapter 2 is that different monitoring and prediction scenarios require very different geoprocessing techniques and architectural approaches. There is no single best way to integrate these technologies; it is a complex choice that is dependant on the nuances of the dataset, the execution environment, the geoprocessing operation, the network and the encapsulating interfaces (Padberg and Kiehle, 2009). For example, the processing of a remotely sensed satellite image for a disaster monitoring scenario may gain a significant improvement in performance if it is executed in parallel on a grid, but similarly this performance gain could also be offset by the time spent transferring the image on to the grid. As such, an important step in the advancement of this field is to determine which combinations of data type, execution environment and monitoring / prediction scenario characteristics work well together.

In this Chapter an attempt is made to develop a typology of geoprocessing operations with respect to distributed processing architectures with a particular focus on real-time operations. The typology defines geoprocessing operations in terms of those characteristics that influence the design choices made by system architects in the development of distributed geospatial processing systems. This delineation of commonly occurring paradigms in geoprocessing is intended to facilitate the future development of application specific tools, frameworks and software development kits for real-time distributed geoprocessing. Furthermore it is anticipated that the typology will enable generic methodologies and integration profiles to be developed that suit the majority of cases and provide a template for geoprocessing application and system design.

The typology developed in this Chapter provides a framework for the practical work of this thesis. An exemplar operation from each geoprocessing category is implemented in Chapters 4-6. This serves to provide a firm foundation for discussion and a basis for answering the research questions set out in Section 2.6. Within this Chapter Section 3.2 reviews existing efforts at classifying geoprocessing operations and Section 3.3 provides a thorough examination of the differences between static and real-time geoprocessing. The main focus of this Chapter, the typology of geoprocessing operations is detailed in Section 3.4. In Section 3.5 an attempt is made to classify common geoprocessing operations in the context of this typology and a critique of the typology is conducted in Section 3.6; the main conclusions of this Chapter are summarised in Section 3.7.

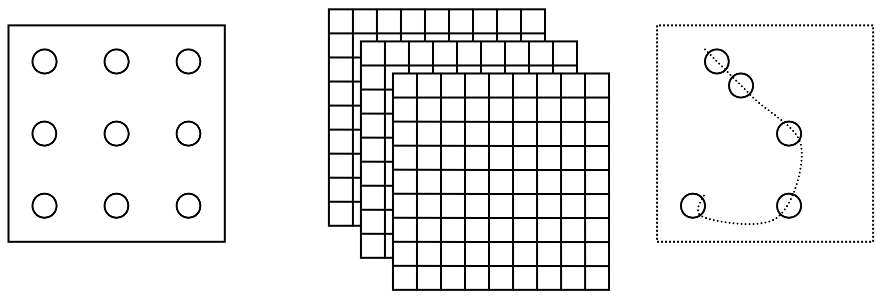
3.2 Review of Existing Geoprocessing Classifications

The ISO has defined a classification of geographic processing services, ISO 19119 (Percivall, 2002) that essentially groups processing services by the function that they perform. Four major classes are defined; spatial, thematic, temporal and metadata. Example operations for each of these classes respectively, are spatial coordinate conversion, thematic feature generalisation, temporal sampling and aggregate statistical operations. This functional classification is useful for defining broad classes of geoprocessing operations although in terms of RM-ODP (Section 2.3.5) it is biased towards the informational viewpoint. According to Faroqui *et al* (1995) it is the computational and engineering viewpoints that are the most important factors in determining the design and implementation of distributed systems as these viewpoints consider issues such as problem partitioning and the matching of applications to platforms. As such this functional classification is not particularly helpful in mapping geoprocessing operations to processing methodologies and architectures.

Di *et al* (2008) suggest a crude classification of geoprocessing based on the stage of the geoprocessing operation in the workflow. It is suggested that there are three stages in the process of converting geospatial data to information. The first stage, *geoquery* is the discovery and acquisition of data, the second stage *pre-processing* involves assembling the data and converting it to the required format and the final stage *geocomputation* is concerned with conducting analysis and simulations with the data. This classification is more pertinent to distributed computing architectures as it considers geoprocessing operations in relation to the workflow. For example, data reducing *geoquery* operations and some *pre-processing* or transformation operations should be performed close to the data. This is noted by Friis-Christensen (2007) who suggests that the OGC WFS be augmented with data reducing operations such as clipping, generalisation and coordinate conversion. While this classification does not specifically consider processing architectures it introduces the workflow as an important geoprocessing characteristic.

Wang and Armstrong (2009) explore the decoupling of parallel geoprocessing solutions from specific high-performance computing architectures through the use of computational transformations which characterise the computational intensity of geographical analysis. Four major types of geoprocessing operations are considered; operation-centric, data-centric, operation and data-centric, neither operation nor data centric. Data-centric transformations are considered to be functions that have a high memory or I/O requirement such as large spatial database transactions, whereas operation-centric transformations have a high computing time requirement. This classification helps to determine the processing methodology that the operation is most suited to. For example, operation-centric tasks may seek to exploit a message passing processing methodology (Section 2.4.4), whereas an operation and data-centric task may be more suited to a data parallel processing methodology (Section 2.4.4). A similar typology is defined by Shi *et al* (2002) who suggest three types of geoprocessing algorithms that are suitable for parallelisation; algorithms for which loop constructs in the code can be exploited by explicit or implicit parallelism (operation-centric), algorithms that can be sub-divisible into smaller geographical areas (data and operation centric), and algorithms with a large data volume but modest compute requirement (data-centric). Shi *et al* (2002) consider it unnecessary to parallelise operations that are neither data nor compute intensive.

In the context of real-time systems, geoprocessing operations may also be classified by the type of sensing system used to collect the data. Beard (2007) proposes three types of sensing system. The first type is termed a Spatial Field of Time Series (SFTS) and refers to multiple fixed spatial locations where one or more attributes are measured at regular intervals resulting in one or more time series. Therefore, for multiple locations a spatial field of time series is created. For example, a fixed set of weather observing stations over a given area would collectively comprise an SFTS. The second type in the classification is a Time Field of Spatial Series (TFSS) and this refers to a time ordered set of spatial fields. For example, a set of images collected by an orbiting satellite over a month would represent a TFSS. The key difference is that SFTS represents a temporally continuous but spatially fixed data series whereas the TFSS represents a data series that is spatially continuous. The final type is a Moving Point Time Series (MPTS) and refers to a sensor that moves and measures its location at regular intervals. The attribute being measured is typically the label for the unit carrying the sensor, i.e. a person, an animal or a vehicle. The MPTS outputs a set of observed positions for the moving object. The three sensing systems described here are depicted in Figure 3.1.



1. Spatial Field of Time Series

2. Time Field of Spatial Series

3. Moving Point Time Series

Figure 3.1: Types of Sensor System [*Langran et al, 1992]*

The development of Map Algebra (Tomlin, 1991) resulted in the delineation of four major classes of geoprocessing operation; local, zonal, focal and incremental. Functions in the local category operate on each individual location, such as a pixel in a raster image, or a point or feature in a vector dataset, and produces for each location a new value that is some function of one or more existing values from that location. Common local operators include math functions such as maximum, minimum, difference, product, square root, sin, cosine and tan.

Global functions compute a new value for each location that are a function of existing values associated with the entire layer. For example, a global maximum function would set the value of each location to the maximum value found in the entire layer, whereas a local maximum function would set the value of each location to the maximum value found at the corresponding location in each of the specified layers.

Zonal functions compute a new value for each location that are a function of existing values within the same region in another layer. For example, the ZonalProduct function multiplies each of the values in one layer by the value of each zone in another layer.

Focal functions calculate a new value for each location as a function of the values taken from surrounding although not necessarily adjacent locations.

Common focal functions include high-pass and low-pass frequency filters and the focal mean that calculates the area weighted average of all values within a neighbourhood.

In Tomlin's (1991) classification the incremental class contains functions that are biased towards hydrological modelling such as aspect, drainage, volume and linkage. However, a subsequent revision of this classification has amalgamated the incremental class into the zonal class due to its algorithmic similarity (DeMers, 2002).

Mennis *et al* (2005) extend the notion of map algebra from two dimensional Cartesian data-space to include time as a third dimension. It was found that this approach enables map-algebra functions to successfully be performed on time-series data thus enabling phenomena to be modelled in both time and space.

Nicolescu and Jonker (2002) conform to the notion of global, zonal and focal operators in their classification of image processing functions. Here they are referred to as point, neighbourhood and global. Point operators are defined as those for which each output pixel is dependant only on the corresponding input pixel such as arithmetic and logical operators on two corresponding images. Neighbourhood operators however create an output pixel that is dependant on the value of several input pixels from the surrounding region. Examples include moving kernel functions such as low/high pass filters and edge detection algorithms. Global operators are dependant on the entire image; examples include average, maximum and minimum functions. The dependence of each operation on other elements in the data aggregate has wide reaching consequences for data decomposition strategy and architectural considerations. For example, point operators can be easily parallelised using a task-farm style decomposition but neighbourhood and global operators are more difficult to parallelise (Braunl *et al.*, 2001). Similarly, geoprocessing operations involving global operators are better suited to processing at source to avoid the transfer of unwieldy datasets across a network.

3.3 Geoprocessing and Time

Introducing real-time data into distributed geoprocessing workflows has a substantive effect on the relative suitability of different processing architectures and methodologies. Primarily this effect is due to differences in the volume and nature of the data and the way in which it is delivered. However, the effect can also be attributed to the overall context in which geoprocessing is taking place. Issues such as how the workflow is invoked, for what purpose and with what degree of immediacy, are crucial to the selection of suitable tools and techniques.

3.3.1 Snap vs Span

GIS have traditionally taken a simplistic view of the world in which all phenomena are represented in a static manner (Langran, 1992). Various attempts have been made to represent the dynamic nature of real world phenomena within GIS; Worboys and Duckham (2004) outline the following four stages in this progression:

Static:	A single static view of the world.
Snapshot:	A view in which dynamic phenomena are represented as a collection of time-stamped states.
Object Lifeline:	A view in which the lifecycle of objects including creation and destruction are recognised.
Events, actions & processes:	A view in which continuous and instantaneous phenomena can be modelled.

Towards the events, actions and processes end of this spectrum the complexities inherent in modelling the real-world in time and space become apparent. Mourelatos (1978) attempted to rationalise the representation of reality by using a taxonomy in which every situation is comprised of both a state and an occurrence, and the occurrence could be represented by either an event or a process. Whereas events occur at a fixed instant in time, processes occur over a time interval; this disparity between instantaneous and interval representations of spatial phenomena is formalised by Grenon and Smith (2004) with their SNAP and SPAN ontology.

In terms of geoprocessing systems, 'real-time' implies we are dealing with temporal representations at the snapshot level or higher in Worboys and Duckham's (Worboys and Duckham, 2004) progression. As such, real-time geoprocessing covers a range of temporal scenarios. At the simplest level an operation may involve the processing of a fixed snapshot of recently collected spatial data. An example of this form of snapshot processing is given by the interpolation of a sea surface temperature map from a series of weather buoys for a given time instant. Only minor differences exist between this form of snapshot geoprocessing and classical static geoprocessing. The actual computation involved is the same but the data may be corrupt or missing due to less reliable sensor data sources. Furthermore, snapshot geoprocessing on live sensor data is likely to take place in an environment in which the results are required immediately.

Conversely, real-time geoprocessing may involve the processing of a series of observations representing a time interval. Extracting information from an observation sequence requires a radically different approach to static geoprocessing and draws on techniques from the field of Data Stream Processing (DSP). In DSP terms a data stream is a potentially unbounded sequence of tuple, timestamp pairs; DSP can be considered an alternative to database technology for coping with streams of data as opposed to persistent datasets (Babu and Widom, 2001). In contrast to traditional database management systems, DSP is concerned with performing static queries on transient data rather than vice versa. Data Stream Management Systems (DSMS) have emerged as a means of managing data streams, both as extensions to existing DBMS (Krishnamurthy *et al.*, 2003) and as systems in their own right (http://esper.codehaus.org). Furthermore, a query language, Continuous Query Language (Arasu *et al.*, 2006) has emerged as a standard means of performing queries over data streams. Notable geoprocessing work in this field includes the GeoStreams project (Hart and Gertz, 2005) on processing streams of remotely sensed image data and the doctoral thesis of Rueda-Velasquez (2007) that presents a framework for stream based change detection in remotely sensed images.

3.3.2 Real-time Data Sources

Madden (2002) in his work on query processing of remote sensors noted that a major difference between sensor data sources and traditional databases is that real-time data is generally delivered in streams without being specifically requested. Sensor network architectures conform to either the warehousing approach of extracting data from devices in a predefined manner and depositing it in a database, or the distributed approach in which only specifically requested data is retrieved directly from the sensors. Clearly the warehousing approach is similar to a static data source; subsequent processing operations can simply adopt a polling mechanism to retrieve data from a repository. In the distributed approach sensor devices form part of the database and processing operations can request streams of data directly from the devices. As noted by Bonnet *et al* (2000) the preferred architecture is dependant on the prevailing type of query. Historical queries that aggregate data over a long time period are better suited to the warehousing approach. However, snapshot queries where data for a given epoch is retrieved on request, and long-running queries that retrieve data over a given time period are better suited to the distributed approach as it avoids the unnecessary collection, transfer and storage of large data volumes. Real-time applications often depend on long running or snapshot queries and the ability of real-time workflows to handle streams is therefore desirable.

Delivery of data in streams does have certain advantages for real-time processing. Transferring large static data sets across networks presents a bottleneck in distributed architectures whereas transferring observations as they are collected enables pipeline parallelism to be exploited (Section 2.4.3); this allows actors on the same branch in the workflow to work at the same time on different parts of the same stream (Rueda *et al.*, 2006).

In the OGC SWE architecture, sensor data is obtained through an SOS or SAS/SES interface which represents pull and push based access to observations respectively. Whether these services adopt a distributed or warehousing approach is unrelated to the service interface, this decision is left to the service implementer. However, the SOS does support the querying of historical data and most implementations to date (52North, NASA, Northropp Grumman) have chosen the warehousing approach. Consequently, in a distributed architecture, the nature of data delivery is on the whole irrelevant, unless access to historical observations are required, in which case a warehousing approach must be used.

3.3.3 Invocation of Real-time Geoprocessing

Coping with continuous data streams introduces a new set of challenges to web service based GIS workflows. In a number of real-time scenarios new data is constantly being produced which must be processed. For example, monitoring applications typically produce observations which must be pre-processed. This presents a design choice for processing services; either the service can be invoked every time a new piece of data arrives, or a long-running process can be established that listens for new observations, and processes each piece of data as it arrives.

Due to the request-response pattern of web services the former approach is the simplest; each item of input data can be passed to the processing service as a parameter in the form of a request, the geoprocessing operation will have a finite runtime and will return a result on completion. However this approach is inefficient and particularly so in a grid environment. Firstly, numerous requests must be formulated by wrapping each data item in a messaging envelope to send to the processing service. Secondly, each request must be sent over HTTP thus consuming network bandwidth and suffering from latency. Thirdly each request must be de-serialized by the processing service and finally, in the case of a grid based processing service the job must be scheduled and queued before it is executed. This is likely to result in significant delays between the data arriving at the source, and the processing results being delivered at the destination.

The alternative is to invoke a single processing task, and pass it a reference to the data source as a parameter. The processing service can then poll the data-source directly and deliver results to the destination as they are processed. The advantages of this approach are that the client need only make a single request to start the processing service and that scheduling and queuing delays are only incurred once when the processing is initiated. Therefore, provided that the frequency of data arrival doesn't exceed the time taken to process the data, the time between the arrival of the data at the source and delivery at the sink is minimised. There are however some disadvantages to this approach. Firstly the ability of OWS to maintain state is poor, thus there is no inbuilt mechanism to provide lifetime management of ongoing processes. As a result, ongoing processes that are started, using a WPS *Execute* request, cannot be stopped. Secondly, each ongoing process is assigned to only a single processor; thus the processor must be able to keep pace with the incoming data. If this is not possible then a backlog will occur, causing the time between arrival of data at source and delivery at destination to steadily increase. Thirdly, this approach doesn't represent an efficient use of grid resources. Once an ongoing process is allocated to a processor, the processor is entirely unavailable to other users, for the duration of the ongoing process. As already discussed, the process must be able to keep pace with the incoming data, thus it will spend a proportion of its time waiting for new data to arrive. It could be argued that these wasted processor cycles could be better utilised by others, and in a utility grid scenario in which compute processing time is charged per hour this may prove to be expensive.

3.3.4 Reliability and Variability of Real-time Data

A major requirement of many real-time geoprocessing systems is full automation; geoprocessing must be able to take place without any manual intervention in the workflow. However, such applications are often safety critical and it is therefore vital that such systems can be relied on (Zerger and

Smith, 2003). Furthermore, it is often the case in hazard monitoring applications that sensors detect no change for the majority of the time and therefore require very little processing capacity, but when an event does occur the need for processing power suddenly increases to cope with the influx of data (Hingne *et al.*, 2003). Real-time systems must therefore be capable of dynamically scaling up and down to cope with the processing burden whilst minimising the usage of processing resources.

3.4 A Real-time Geoprocessing Typology

Following an extensive review above of existing geoprocessing classifications a new geoprocessing typology is presented in this Section. This new typology takes into account the data, compute and usage characteristics of geoprocessing operations as well as considering the real-time scenarios in which they are employed. The purpose of this typology is to relate the physical data and compute aspects of geoprocessing to specific styles of monitoring and prediction problem. It is anticipated that the formalisation of this relationship will be of benefit to the future development of large scale distributed monitoring and prediction systems.

Let us consider three temporal representations of geospatial data; static, snapshot and stream-based. Static data represents a single unchanging view of reality, snapshot data represents a fixed view of reality at a number of discrete moments in time and stream-based data represents a dynamic view of reality over a continuous time interval. These three representations are depicted in Figure 3.2 in which a two-dimensional data space is extruded through time in accordance with its temporal representation.

In reality data streams are always comprised of a series of discrete observations. Therefore snapshot and stream based data representations could be considered one and the same. However, in terms of data processing the key difference between stream and snapshot representations is that processing operations on data streams are invoked regularly and frequently, i.e. they are time triggered rather than event triggered. Typically data stream processing involves basic pre-processing or change detection monitoring whereas snapshot processing typically involves large one-off modelling or simulation tasks.

A further difference between these two paradigms is that stream processing is typically confined to processing a single stream of data and this usually implies a single sensor data source. Conversely snapshot processing may incorporate data from multiple sources for a given time instant. As a consequence, more complex operations involving multiple sensors such as simulations and predictions usually fall into the snapshot processing category.

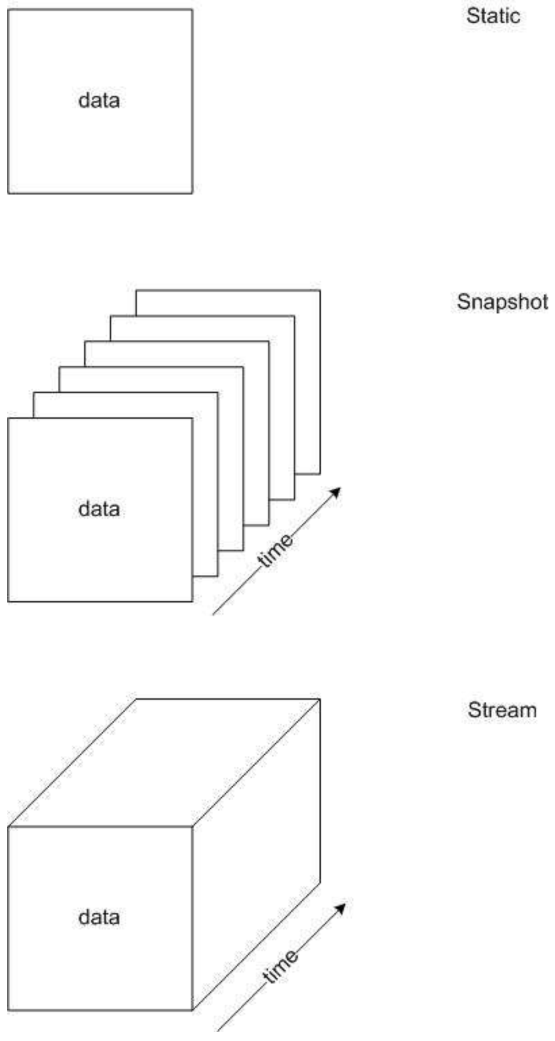


Figure 3.2: Static, Snapshot and Stream Data Representations

Two distinct geoprocessing categories have been delineated, stream geoprocessing and snapshot geoprocessing. The characteristics of each of these categories are outlined in Table 3.1

Table 3.1: Characteristics of Geoprocessing Paradigms

Characteristic	Stream	Snapshot
Regularity of invocation	Regular	Regular or irregular
Trigger	Time	Event
Number of sensors	1	>=1
Temporal representation	Interval	Instant

With regards to a distributed processing architecture the above categorisation facilitates the choice of design. The processing of numerous observation streams can easily be parallelised by assigning one stream to each processor, or by using pipeline parallelism as data items are already divided into an ordered sequence (Section 2.4.3). Alternatively, for stream based processing operations that carry a high time complexity, data stream partitioning can be used to divide the workload amongst several processors. Furthermore, the small but relentless torrent of data associated with the stream paradigm is easily managed in a distributed network environment whereas larger data files are more cumbersome to work with as they require longer transfer times and can often not be read until the transfer is complete.

In processing terms the snapshot paradigm can be considered very similar to static processing; input and output data are discrete and the operation has a finite lifetime. The main differences are that snapshot processing is triggered by an event and the temporally discrete input data is obtained from sensor data sources. Furthermore the results are likely to be required within a certain time frame. The appropriate parallelisation technique for processing snapshot data is dependant on the granularity of the geoprocessing operation, thus requiring snapshot processing to be further disaggregated. Granularity can be considered a spectrum (

Figure 3.3) with high data volume and low computational intensity operations such as spatial database transactions at one extreme and low data volume, high computational intensity operations at the other. The former are termed 'fine-grained' operations because the dominant resource constraints are memory and I/O which result from excessive communication. Conversely, coarse-grained operations utilise virtually no communication but a large amount of processor cycles, serial tasks fall into this category.

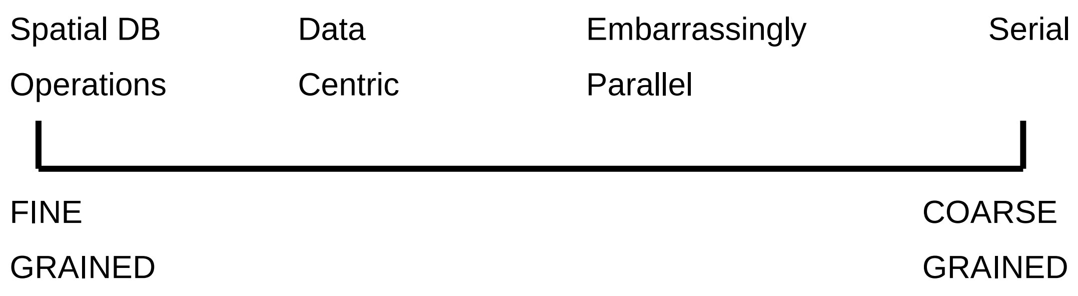


Figure 3.3: The Granularity Spectrum

Considering these categories it is proposed that the snapshot geoprocessing category can be further subdivided into fine-grained and coarse-grained geoprocessing operations. Fine-grained geoprocessing operations are typically global, i.e. they require the entire data aggregate to compute a result. This includes simple spatial database operations such as unary and binary selection as well as complex simulations and predictions that involve machine learning.

Partitioning fine-grained operations is difficult and often unnecessary as the primary resource constraint is that of memory and I/O. Where the partitioning of computation is unnecessary, fine-grained geoprocessing operations can often be performed within a spatial database using either standard database queries (SELECT,JOIN,INSERT,UPDATE) or tightly integrated spatial functions (area, boundary, buffer, distance_to), i.e. SQL-MM. However, if the partitioning of computation is necessary then two approaches are possible. Firstly, a parallel relational DBMS could be used as these are capable of automating the process of parallelisation using table partitioning (DeWitt and Gray, 1992) and parallel spatial joins (Zhou *et al.*, 1998). Processing at the database is preferred because fine-grained operations are best performed in a tightly coupled manner, i.e. close to the data, as this minimises costly data transfers.

However, in some cases it may be necessary to partition fine-grained operations that are unsuitable for database processing. Typically complex models and simulations will fall into this category that involves significant interaction and manipulation of data which cannot be expressed in SQL, or that have a very high time complexity. In these situations an MPP architecture (Figure 2.3) and a message passing programming paradigm (Section 2.4.4) is likely to be the most suitable combination. Fine-grained operations have a high degree of data dependence so the MPP / message-passing approach allow complex interactions between sub-processes to be rapidly exchanged. These two styles of fine-grained geoprocessing are depicted in Figure 3.4.

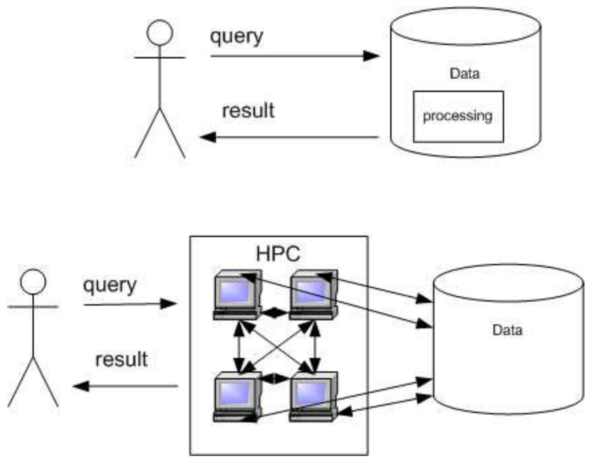


Figure 3.4: Database and MPI / database styles of fine-grained geoprocessing

Coarse-grained operations are easier to partition than their fine-grained counterparts; less interdependency between sub processes ensures that less inter-processor communication is required. Consequently coarse-grained operations are easy to partition using an event parallelism or geometric approach to decomposition. Coarse-grained geoprocessing operations are typically point or neighbourhood operations that can be processed as a series of independent sub-processes.

Thus, three distinct geoprocessing paradigms have been identified, data stream geoprocessing, coarse-grained snapshot geoprocessing and fine-grained snapshot geoprocessing. This simple taxonomy of geoprocessing paradigms is depicted in Figure 3.5 and formally described below.

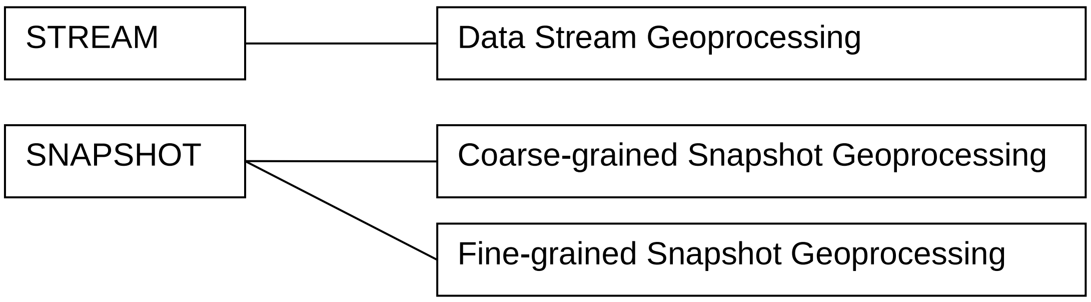

STREAM
Data Stream Geoprocessing
SNAPSHOT
Coarse-grained Snapshot Geoprocessing
Fine-grained Snapshot Geoprocessing

Figure 3.5: Geoprocessing Paradigms

3.4.1 Data Stream Geoprocessing (DSG)

Data Stream Geoprocessing (DSG) is carried out in monitoring scenarios for which a steady stream of incoming geospatial observations must be processed. DSG data consists of an observation stream that is unbounded in time; observations are frequently and regularly occurring but typically small in volume. The corresponding processing operation is thus perpetual. A UML2 sequence diagram (Figure 3.6) outlines the relationship between sensor, observation and geoprocessor in a DSG environment.

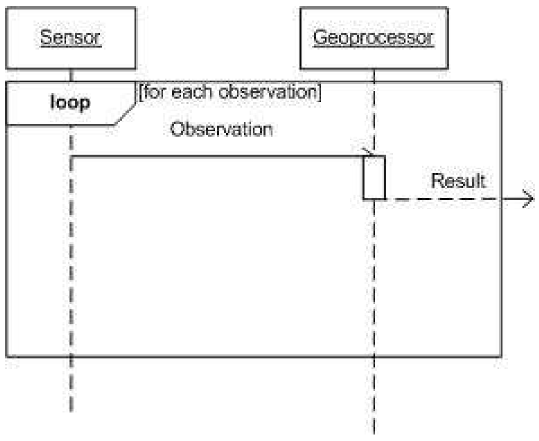


Figure 3.6: A UML2 Sequence Diagram of Data Stream Geoprocessing

3.4.2 Fine-grained Snapshot Geoprocessing (FGSG)

Fine-grained Snapshot Geoprocessing (FGSG) occurs in prediction systems and simulations. FGSG involves the one-off execution of a geoprocessing operation on a regularly updated data aggregate. Data dependence is high in FGSG and operators are typically global. Whereas DSG operates on a stream of observations over an unbounded time period, FGSG operates on an observation set at a fixed snapshot in time. FGSG are likely to be triggered by an alert caused by the change in a real-world condition.

3.4.3 Coarse-grained Snapshot Geoprocessing (CGSG)

Coarse-grained Snapshot Geoprocessing (CGSG), like FGSG involves the one-off execution of a geoprocessing operation on a fixed snapshot of a regularly updated data aggregate. However, CGSG operations have lower data dependence than FGSG. As such they are more likely to involve local or focal (point or neighbourhood) operators rather than global operators so the processing operation can be naturally subdivided for parallel processing through data decomposition. CGSG form the majority of parallel geoprocessing examples in the literature, encompassing both event parallel and geometric parallel approaches (Section 2.4.3).

3.5 Categorisation of Common Geoprocessing Operations

In this Section the typology outlined above is considered with reference to a number of geoprocessing operations that are commonly found in standard GIS software packages as well as those operations that are commonly used in monitoring and prediction scenarios. In a number of cases the examples are taken from studies cited in the previous Chapter (Chapter 2).

Using this typology it is not possible to categorise the geoprocessing operations commonly found in GIS toolboxes outside of the context in which they are used. For example, let us consider a simple boolean intersection operation on two real-time data layers; boolean intersection is a geoprocessing operation in which two layers of map data are overlaid and only the features that intersect are retained. Supposing we have a field of 100 temperature sensors and 100 rainfall sensors, each sensing 1m^2 of a 10m^2 grid and we are interested in identifying areas that have had no rainfall in the past hour and that has a temperature over 15℃. The first stage is to conve rt each dataset into binary, so for the rainfall datast the grid cells that have had no rainfall are assigned a value of 1, and all other cells are assigned a value of 0. Similarly for the temperature dataset, cells with a value of 15℃ or less are assigned a value of 0 and cells with a value greater than 15℃ are assign ed a value of 1. The actual intersection operation on these two layers simply involves multiplying the value of corresponding cells in each layer. The resulting layer will show only the dry cells for which the temperature is greater than 15°C as having a value of 1. This style of operation is easily sub divisible as each multiplication operation could be performed as a separate process in which case the operation would be classed as CGSG.

We could however consider boolean intersection in a different context. Let us consider a similar scenario in which the rainfall dataset is a static dataset comprised of yearly average values rather than regularly updated values from live sensors, but our temperature data is still sourced from a sensor network. In this case we are interested in identifying cells in which the temperature is greater than 15℃ and the yearly rainfall is less t han 800cm. Given that our temperature sensors collect readings every 1 minute we want to update a map with the areas that meet our criteria, every minute. In this case the operation is invoked continuously and regularly and has only one sensor input, thus the same boolean intersection operation would be classified as DSG in this instance.

There are numerous instances in which some overlap occurs between DSG and CGSG operations as well as DSG and FGSG operations. However, FGSG and CGSG operations never overlap. This relationship is depicted in Figure 3.7.

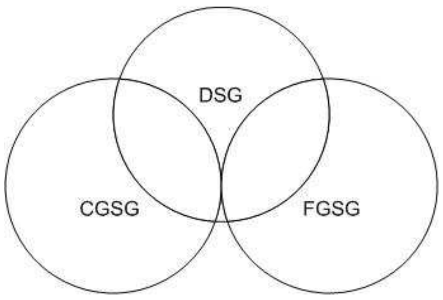


Figure 3.7: Venn Diagram showing the relationship between classes in the geoprocessing typology

In Table 3.2 the granularity of common geoprocessing operations are displayed. Coarse-grained operations may belong to either the DSG or the CGSG category depending on context. Similarly fine-grained operations may belong to either the FGSG or the DSG category.

Table 3.2: Common Geoprocessing Operations

Operation	Description	Granularity	Explanation
Subset: Select / Clip	Subsetting a dataset either by clipping it to the extents of a bounding box (raster data) or through a select query (vector data)	Fine	The entire dataset must be scanned through to select features of interest, i.e. this is a global operation. Although extract operations can be performed in parallel using one of several techniques, the primary constraint is that of data volume rather than computational power.
Overlay: Intersect / Union	The process of taking two layers of map data and overlaying them to form a new layer. The Intersect operator (Boolean 'and') retains only features that exist in both layers, the Union operator retains all features (Boolean 'or')	Coarse	Overlay can be achieved on a feature by feature basis with no knowledge of other features. Therefore this operation can be naturally subdivided using domain decomposition.
Buffer	The process of finding the region within a certain distance of a feature or featureset	Coarse	Features can be buffered on a feature by feature basis
Line Simplification	The process of simplifying features to condense a dataset.	Coarse	Features can be generalised on a feature by feature basis.

Create theissen polygons	The process of creating a polygon dataset from a point dataset where each polygon contains only one point and all locations within the polygon are closer to its point than its neighbours	Fine	The polygon surrounding each point cannot be generated without knowledge of neighbouring points in the dataset
Line of Sight Analysis	The process of analysing a digital elevation model to determine where features are visible from	Fine	Determining where a point is visible from is a global operator as it requires analysis of the entire dataset.
Network Analysis	The process of analysing a spatial network. Common functions include: Calculate network proximity Assign point to nearest point on a network Calculate the shortest path between two points	Fine	The entire network must be considered for every network computation.
Geostatistical Kriging	The process of interpolating the value of a variable at unsampled locations weighted using spatial dependence	Fine	Requires knowledge of all other values in the dataset.
Inverse Distance Weighted Interpolation	The process of interpolating the value of a variable at unsampled locations weighted by inverse distance	Coarse	Interpolation at each location requires knowledge of neighbouring values but not the whole dataset.
High-pass filter	The process of applying a moving window kernel to an image that increases the constrast and thus emphasizes edges and detail	Coarse	Requires knowledge only of neighbouring values
Low-pass filter	The process of applying a moving window kernel to an image that reduces the contrast, i.e. has a smoothing effect	Coarse	Requires knowledge only of neighbouring values
Geometric image correction	The process of resampling a remotely sensed image from image coordinates to ground coordinates, either using a mathematical model or ground control points	Coarse	Depending on the approach taken this operation requires knowledge of the transformation parameters and in some cases the values of surrounding

			pixels.

3.5.1 Data Stream Geoprocessing (DSG) Operations

Typical examples of DSG operations include basic transformations such as unit conversion, data format conversion and coordinate system transformations on observations from sensor data sources. DSG can also include sub-setting and filtering operations such as unary and binary selection and this category therefore fits a variety of real-time change detection applications. Included in this category are geoprocessing operations that search for specific patterns in an observation stream; this is termed Event Stream Processing (ESP) (Luckham and Schulte 2008). ESP is a subtype of Complex Event Processing (CEP) (Luckham, 2006) that enables higher level information to be extracted from a stream of observations. Recent interest in ESP has prompted the OGC to produce a discussion paper on a language for specifying event patterns, Event Pattern Markup Language (EML) (Everding and Echterhoff, 2008) which is currently used to specify level 3 filters for the proposed SES specification. Finally DSG can also include operations that augment observations with information from other static datasets. For example, the map-matching operation (Section 4.3.4) relates position observations taken by a moving entity to a road network dataset.

The face recognition CCTV system described by Peacock *et al* (2004) (Section 2.2.2) fits the DSG criteria. A stream of CCTV frames are analysed and compared to a static database of facial images in an attempt to identify persons of interest. Observations occur frequently, regularly and perpetually and the only data dependency is on an external static database of facial images.

The ANGEL vehicle monitoring project (Planas *et al.*, 2008) described in Section 2.2.2 provides another DSG example. Vehicle position data is continually processed by a risk monitoring system that analyses the vehicle’s position and identifies safe stopping places.

3.5.2 Fine-grained Snapshot Geoprocessing (FGSG) Operations

FGSG encompasses a wide variety of geoprocessing operations such as spatial interpolation, geostatistical and spatial statistical operations on a field of sensors. Additionally network analysis route finding operations such as Dijkstra's shortest path (Dijkstra, 1959) and the A* algorithm (Hart *et al.*, 1972) can be considered to fall into this category if the network cost is updated in real-time. Furthermore, most geo-hazard prediction and simulation algorithms also lie in this category as they are global operators based on regularly updated real-time information. FGSG operations are well suited to a tightly coupled processing style as they typically require access to a large data aggregate. Due to the high data dependence inherent in FGSG operations they are not easy to parallelise. However, parallelisation is possible either using a parallel database or a message passing approach.

The REIS described by Nakamuru *et al* (2009) (Section 2.2.1) in which observations from 800 seismometers are stored in shared memory and updated every second is an example of a FGSG system. The system calculates 1 and 30 second averages of ground acceleration and maximum amplitude. Although this operation is carried out regularly and frequently it can be considered snapshot processing as it combines observations from an array of sensors for a given snapshot in time. As the entire data aggregate is required to calculate average values this operation falls into the fine-grained category.

Evacuation planning simulators such as those described in Section 2.2.1 also fall into the FGSG category; a snapshot of the current situation is used to plan for several scenarios in which the entire data aggregate is required for computation.

3.5.3 Coarse-grained Snapshot Geoprocessing (CGSG) Operations

Coarse-grained Snapshot Geoprocessing (CGSG) is another geoprocessing paradigm that commonly forms part of geoprocessing workflows. Like FGSG, CGSG involves the one-off execution of a geoprocessing operation on a fixed snapshot of a regularly updated data aggregate. However, CGSG operations have lower data dependence and typically involve point or neighbourhood operators so the geoprocessing operation can be naturally subdivided for parallel processing through data decomposition. Common examples of CGSG operations include raster intersection, raster overlay, buffering, generalisation, frequency filters and geometric image correction.

Examples of CGSG in the literature are chiefly comprised of static geoprocessing operations although there are some cases in which such operations form part of a real-time workflow. Hawick *et al* (2003) describes the classification of remotely sensed imagery using a CGSG approach, Lee and Hamdi (1995) describe a CGSG convolution filter over a remotely sensed image and Wagner and Scott (1995) describe a number of parallel raster cost volume operations that fits the CGSG category (Section 2.4.3).

3.6 Typology Evaluation & Critique

The typology described in Section 3.4 provides a basis on which to relate geoprocessing operations to distributed computing architectures. An attempt has been made in Section 3.5 to consider geoprocessing operations in the monitoring and prediction domain in the context of this typology which is based on two broad principles. Firstly, operations that are invoked frequently and regularly should be treated differently to operations that are invoked on an occasional basis in a grid computing architecture to avoid cumulative job submission overheads. Secondly, coarse-grained and fine-grained operations should be treated differently because coarse-grained operations can easily be parallelised in a grid computing architecture using domain decomposition or through an event-parallel approach, whereas fine-grained operations are more difficult to parallelise because they require a larger degree of inter-processor communication. Furthermore it is unnecessary in many cases to parallelise such operations as the computational load is not always constrictive.

Although this typology appears to fit the majority of cases there are a number of geoprocessing operations that can be considered exceptions, and are consequently difficult to categorise. Firstly are operations that seemingly fit the

DSG category but that process observations from more than one sensor. One example of this is the Firegrid project described by Han *et al* (2010); observations must be processed continually and regularly but data is arriving from several sensors. In this case data from a large sensor array is fed into a model that calculates fire parameters and likely structural building damage in real-time; the data is fed in at 0.1 second intervals. As numerous sensors are involved this does not fall into the DSG category. However, nor does it fall directly into the snapshot category as data from a time-interval is processed. It could be argued that in these cases the processing operations described are of a higher taxonomical level because the processing resulting from each time step forms an FGSG operation in itself, thus the overall process can be considered an iterative sequence of FGSG operations.

Another situation that could be regarded as an anomaly in the context of this classification occurs when a geoprocessing workflow is comprised of several operations, each of which falls into different categories. A common scenario may involve a DSG operation that performs some pre-processing on a raw data stream which is subsequently stored as a data aggregate and subject to further CGSG or FGSG operations. For example, a set of GPS observations from a moving vehicle may undergo a coordinate conversion as a DSG operation before being stored in a database. Occasional analysis operations on this stored data may follow, such as the calculation of the vehicle's mean position which would constitute an FGSG operation. Multi-type workflows such as this are common, particularly in monitoring and prediction systems. As such it is important to recognise that this typology can only be used to categorise contextualised geoprocessing operations, not entire monitoring and prediction systems. This means that a certain operation may fall into one category in one system and another in a different system, depending on its frequency and regularity of invocation.

A final noteworthy point is that there appears to be a substantive difference between fine-grained data intensive operations such as spatial database transactions and fine-grained compute intensive transactions such as network analysis yet they appear in the same category in this typology. The key difference between these two types of operation is that in the former case there is rarely a need to parallelise the processing operation due to a low time complexity whereas in the latter case there is often a need to parallelise. Unless a spatial database can be utilised the parallelisation of either type of FGSG operation is non-trivial and is likely to require a HPC architecture and a message passing programming model. Although the parallelisation of complex FGSG operations is an interesting topic it has already been the subject of considerable research in the parallel processing and high performance computing domains and falls outside of the scope of this thesis. As such, the further sub-division of FGSG operations may be a topic that is worthy of future research but at this stage all FGSG operations are considered as one broad category.

3.7 Conclusion

A prototypical typology of geoprocessing operations has been developed in this Chapter that has attempted to relate geoprocessing to grid computing architectures and geoprocessing methodologies. The aim of this thesis as set out in Section 2.6 is to develop an appropriate conceptual and practical framework in which open standards in grid computing, sensor web and geospatial web services can be combined. In this regard the typology set out in this Chapter provides a skeleton for the conceptual framework. An attempt will be made in subsequent Chapters to build up the practical framework by providing an implementation of each geoprocessing category described in this Chapter.

Three classes of geoprocessing operation have been suggested and specific examples of each operation type have been given. The CGSG and DSG operations are suited to run on grid type architectures; i.e. geographically disparate processors that are connected using standard internet connections. However, FGSG are better suited to execution close to the data source, either in a spatial database or on tightly coupled HPC clusters.

The snapshot geoprocessing operations, CGSG and FGSG lie within the remit of the OGC WPS interface. However, this interface may need to be extended or modified to cope with stream-based geoprocessing operations because in its current state it does not support the execution of on-going processing operations such as DSG. Geoprocessing operations are currently invoked via the WPS interface using the *Execute* operation but there is no mechanism to start or stop an ongoing set of operations. Conversely, the OGC SAS and SES interfaces provide a mechanism to filter streams of geospatial observations, and in the case of the SES, complex ESP filters can be performed. However, it is not possible to transform observations through this interface.

CGSG operations appear to be well suited to basic low-level atomic geoprocessing operations such as might be found in a standard geoprocessing toolbox. Similarly, a number of FGSG operations such as network analysis and spatial joins can be considered generic, in that they are widely found in geoprocessing toolboxes. However the majority of spatial models and simulations also fall into the FGSG category and these tend to be complex, high-level operations that do not form a part of standard toolboxes.

Stream-based geoprocessing operations are fundamentally different, they input and output streams of data rather than discrete data elements or aggregates. This raises the question as to whether a new geoprocessing toolbox is required for real-time GIS that provides stream-based processing on geospatial data.

In order to further develop the classification described here and to help answer some of the remaining research questions in this field, the implementation of an operation from each category is described in subsequent Chapters. The systems presented in the following Chapters serve to address specific questions about the suitability of the proposed design strategies, tools, and techniques for implementing scalable and interoperable geospatial monitoring and prediction systems.

Chapter 4 Data Stream Geoprocessing

4.1 Introduction

In this Chapter a practical example of a scalable real-time geoprocessing system that conforms to current relevant standards in geospatial web services and grid computing is presented. Specifically, the geoprocessing system is to belong to the DSG category outlined in Chapter 3 and is to demonstrate how real-time data from a collection of independent sensors can be processed in near real-time by running several concurrent processes on the grid.

Two distinguishing features of this system set it aside from previous work in this field; first is the idea of using grid computing to run continuous open-ended jobs to process streams of sensor data in near real-time, as opposed to invoking finite compute jobs to process a portion of a sensor stream (Chen *et al.*, 2010, Williams *et al.*, 2009). Secondly, is the concept of pre-processing sensor data as soon as it is collected, and publishing the added value data alongside the raw data, thus enabling users or higher-level applications to usefully consume the pre-processed data in a timely fashion.

4.2 System Design

4.2.1 User Scenario

The geoprocessing system in question is designed to facilitate a scenario in which a fleet of vehicles equipped with onboard GPS receivers require the road they are positioned on to be identified. This geoprocessing operation is commonly known as map-matching and is a necessary pre-cursor for vehicle tracking and vehicle routing systems (Ochieng *et al.*, 2004). From a fleet management perspective vehicle tracking systems enable better vehicle utilisation through analysis of trends in historical data (Couillard, 1993). Furthermore, they are capable of improving response times for emergency jobs (Ghiani *et al.*, 2003) and for providing a means of accountability to service recipients by proving that a vehicle was at a particular place at a specific time (Crainic *et al.*, 2009). In addition to their application as a fleet management tool, the Floating Car Data (FCD) provided by vehicle tracking systems can also be usefully applied to traffic monitoring scenarios (Akinci *et al.*, 2003). For example Torp and Lahrmann (2005) developed a system that utilises FCD for traffic queue detection that was found to be substantially cheaper than using in-situ road sensors. Similarly, Wang *et al* (2008b) developed a navigation system that used both historic and real-time FCD data to predict road travel speed.

Local government agencies in the UK have been quick to adopt vehicle tracking systems; this is perhaps unsurprising given their combined interest in both traffic management and in the operation of a large fleet of maintenance vehicles. The system described in this Chapter is designed for use by Newcastle City Council (NCC) which currently operates a fleet of 890 vehicles (Anderson *et al.*, 2008). This system is not an entire vehicle monitoring, tracking and navigation solution as offered by a number of commercial companies; rather it provides only the map-matching aspect which could be further augmented with more complicated routing and fleet management functions as required.

A key functional requirement of this system is to correctly identify the road that each GPS measurement corresponds to, in near real-time. Given that the map-matched data is to be used for a wide range of fleet management and traffic monitoring tasks it is important that an acceptable level of map-matching accuracy is maintained. A trade-off exists between absolute map-matching accuracy and speed of computation and although a number of very accurate map-matching algorithms have been developed, the best of these are not capable of working in real-time (Marchel *et al.*, 2005). This system must be able to provide matched positions within an acceptable time period so its use in near real-time navigation systems is not precluded. Consequently, design goals of this system include the maximisation of map-matching accuracy and the minimisation of latency. The system must also be capable of scaling to the size of the entire fleet of NCC, and to provide map-matching for the entire Tyne and Wear output area.

4.2.2 Design Considerations

The proposed system requires the tracked vehicles to be fitted with GPS receivers that are capable of wirelessly streaming their observations back to a web server via a communication protocol such as GSM. To reduce costs a sensor emulator that publishes historical GPS measurements at regular intervals is used, mimicking the function of a real on-board GPS receiver.

Grid computing elements and associated services are accessed through the UK National Grid Service (NGS) (http://www.ngs.ac.uk). The NGS is a computational and data grid comprised of a number of computing clusters located at academic institutions throughout the country. The primary goal of the NGS is to federate access to computational and data resources at four core sites and a number of other affiliate and partner sites throughout the UK. It is a service run for researchers that aims to support a national grid infrastructure.

A flexible agile approach to software development (http://www.agilemaninfesto.org) is to be taken as this enables design changes to be made as new ideas come to light. The system as a whole is to adopt a SOA and will therefore not be tied to a specific operating system, although specific components may be subject to platform constraints.

4.2.3 Software & Tool Selection

Where possible, the components used in this system will use or extend existing open source tools and software. This approach minimises costs, avoids duplication of effort and provides scope to make a useful contribution to the open source community.

Spatial databases have become the preferred method of storing spatial data for a number of applications, largely due to their use of indexes to efficiently retrieve both spatial and attribute data (Worboys and Duckham, 2004). In this system, spatial databases are used to store data behind servers, both for the WFS and SOS. For this purpose it has been opted to use PostGIS (version 1.5) (http://postgis.refractions.net), a spatial extension to PostgreSQL (version 8.3) (http://www.postgresql.org) which is a free and open source object-relational database management system. PostGIS complies with the OGC's Simple Features Specification for SQL (Herring, 2006) and most of the Multimedia SQL standard (Stolze, 2003). Furthermore it is supported by a wide variety of open source software tools and applications. For example 52 North use PostGIS databases in their web applications and PostGIS can be configured as a data source in Geoserver.

52 North (http://www.52north.org) is an international research and development company that develop and promote open source geospatial software (Kraak *et al.*, 2005). Notably 52 North have been tracking the OGC standards process and have developed early implementations of young and proposed specifications. Currently the initiative is focusing its efforts on geoprocessing, sensor web and security. The system outlined in this Chapter relies on a 52 North SOS (version 3.0.1) and extends a 52 North WPS (version 2.0). Apache Tomcat (http://tomcat.apache.org) version 6.0 is used as an application container for the WPS and the SOS. Tomcat is a reliable servlet container that is widely used for deploying server-side Java applications.

Geoserver (http://www.geoserver.org) is the official reference implementation of OGC WFS, WMS and WCS. It is written in Java and is capable of serving spatial data from a variety of sources, including PostGIS databases. Geoserver (version 2.0.2) is to be used in the proposed system to serve road network data through a WFS interface.

There are a number of open source Java libraries available for storing and manipulating spatial data that are designed to facilitate the development of geospatial software, many of which form a part of the Open Source Geospatial Foundation (http://www.osgeo.org). Geotools (http://www.geotools.org) is one of the most comprehensive libraries and it is arranged in modules of functionality. A core part of Geotools is the Java Topology Suite (version 1.7) (http://www.vividsolutions.com/jts/jtshome.htm) that provides a useful library for representing two dimensional geometries in Java. GDAL/OGR (http://www.gdal.org) provides a comprehensive library for reading and writing both raster and vector data formats, and also for providing conversions between formats. PROJ4 (http://trac.osgeo.org/proj/) (version 1.6.2) provides a similar function, but for re-projecting and transforming data between coordinate systems. Whilst PROJ4 is built into PostGIS, OGR is packaged inside a software kit called FWTools (version 2.4.7) (http://fwtools.maptools.org) that enables data to be loaded into PostGIS from a variety of formats.

The OMII Campus Grid Toolkit version 2.2 (http://www.omii.ac.uk/wiki/CGT) forms a part of the OMII-UK software stack and provides a unified interface from which to access computational and data resources on the grid. In this work the Campus Grid Toolkit is used as a GridSAM client through which computational jobs can be submitted to the NGS through one of their published GridSAM endpoints. GridSAM is a JSDL and OGSA-BES compliant job submission interface that enables jobs to be submitted to a variety of back end distributed resource managers such as Globus, Condor and PBS.

Other tools used in the development of the system described in this Chapter include Apache Maven (http://maven.apache.org) (version 3.0.3) for building projects and managing dependencies, InterpOSe (http://www.dottedeyes.com/spatial_data_loading/interpose/) for transforming Ordnance Survey data and the Eclipse (http://www.eclipse.org) (version Ganymede) and Netbeans (http://www.netbeans.org) (versions 6.1 - 6.9) software development environments. Apache JMeter (http://jakarta.apache.org/jmeter/) (version 2.4) was used in the testing phase to carry out load and performance tests.

4.2.4 Architectural Overview

The proposed system is to be comprised of the components outlined in Figure 4.1; the request / response pattern which describes how the system operates is described here and outlined as a UML sequence diagram in Figure 4.2. Example requests and responses are detailed in Appendix D.

Initially, each instrumented vehicle must register with a SOS instance using a *RegisterSensor* request (Label 1 in Figure 4.1). This creates a unique identifier for each vehicle and this identifier is appended as a new procedure element in the SOS database. Once registered to the SOS each vehicle begins streaming their position measurements to the SOS using an *UpdateSensor* request (Label 2 in Figure 4.1). This request updates the SensorML document for each sensor with a new position and time stamp.

At this stage each vehicle is streaming their position to the SOS; at any point a client is now able to begin the process of map-matching these positions. Map-matching is invoked on a per vehicle basis by a client via a WPS *Execute* request (Label 3 in Figure 4.1). The *Execute* request must contain five parameters, firstly the unique identifier of the vehicle on which to commence map-matching, secondly the endpoint address of the SOS instance, thirdly the endpoint address of a WFS that contains road network data and finally the WFS namespace and typename parameters. These final parameters are used to identify the road network dataset from the collection of datasets hosted at the WFS address.

The WPS translates the *Execute* request into JSDL and submits the map-matching task as a new Grid processing job through a GridSAM client (Label 4 in Figure 4.1). Upon submission, GridSAM creates a unique identifier for the job which is returned by the WPS in the *Execute* response. The GridSAM service parses the JSDL document, authenticates the request, retrieves proxy credentials (Label 5 in Figure 4.1), translates the request into an infrastructure specific job submission language and submits the job (Label 6 in Figure 4.1). Authentication and credential retrieval is carried out by contacting the MyProxy service using MyProxy parameters embedded in the JSDL request. The NGS production grid to which the GridSAM instance submits jobs is built on the Globus Toolkit version 2, so in this case the request is translated into Globus Resource Specification Language which is the native job submission language of the Globus toolkit. Native Globus services subsequently handle all aspects of execution management, including job scheduling and submission.

Once the scheduled job reaches the front of the queue it is assigned to a suitable processor and the executables are staged onto this machine from the user's home directory on the GridSAM head node. The job is a Java program that performs the map matching operation and it inherits the arguments specified in the JSDL document, which in turn were inherited from the WPS *Execute* request (Label 7 in Figure 4.1). Initially the program polls the SOS to obtain positions sampled by the vehicle in the previous 60 seconds. This is done by making a *DescribeSensor* time period request that retrieves a SensorML document detailing the vehicle's position history.

For each sampled position in the SensorML document the bearing between observations is derived and a WFS bounding box request geographically centred on the observation is constructed and submitted to the WFS. Matching is performed by comparing each sampled position to the feature collection returned by the WFS to deduce the most probable road to which the observation belongs. The map matching algorithm is explained in more detail in Section 4.3.5. Finally, the unique identifier of the matched road is inserted into the SOS as a new observation, as is the bearing of the vehicle at each sampled position.

The whole process is repeated every 60 seconds and continues to run until the client passes a *StopExecuting* request to the WPS-proxy containing the unique identifier of the job they wish to stop. The WPS-proxy forwards this request to GridSAM and cancels the job.

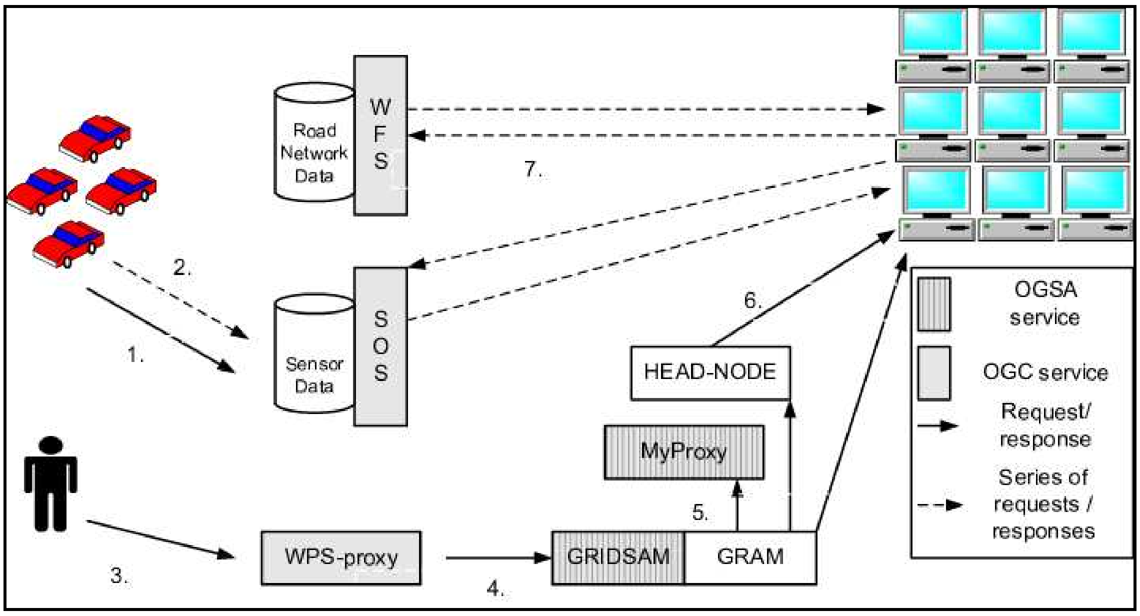


Figure 4.1: Interaction Sequence between map-matching system components

1. Vehicle sends *RegisterSensor* request to SOS.
2. Vehicle starts streaming *UpdateSensor* requests to SOS with latest position.
3. Client begins map-matching by sending an *Execute* request to WPS
4. WPS translates *Execute* request into JSDL and forwards to GridSAM.
5. GridSAM retrieves user credentials from MyProxy service.
6. Executables are staged from head node to a node on the cluster and the map-matching job is started.
7. The map-matcher retrieves recent position observations from the SOS, matches them against the road network stored in the WFS and inserts the matched position back into the SOS. This process continues until a *StopExecuting* request is sent by the client to the WPS.

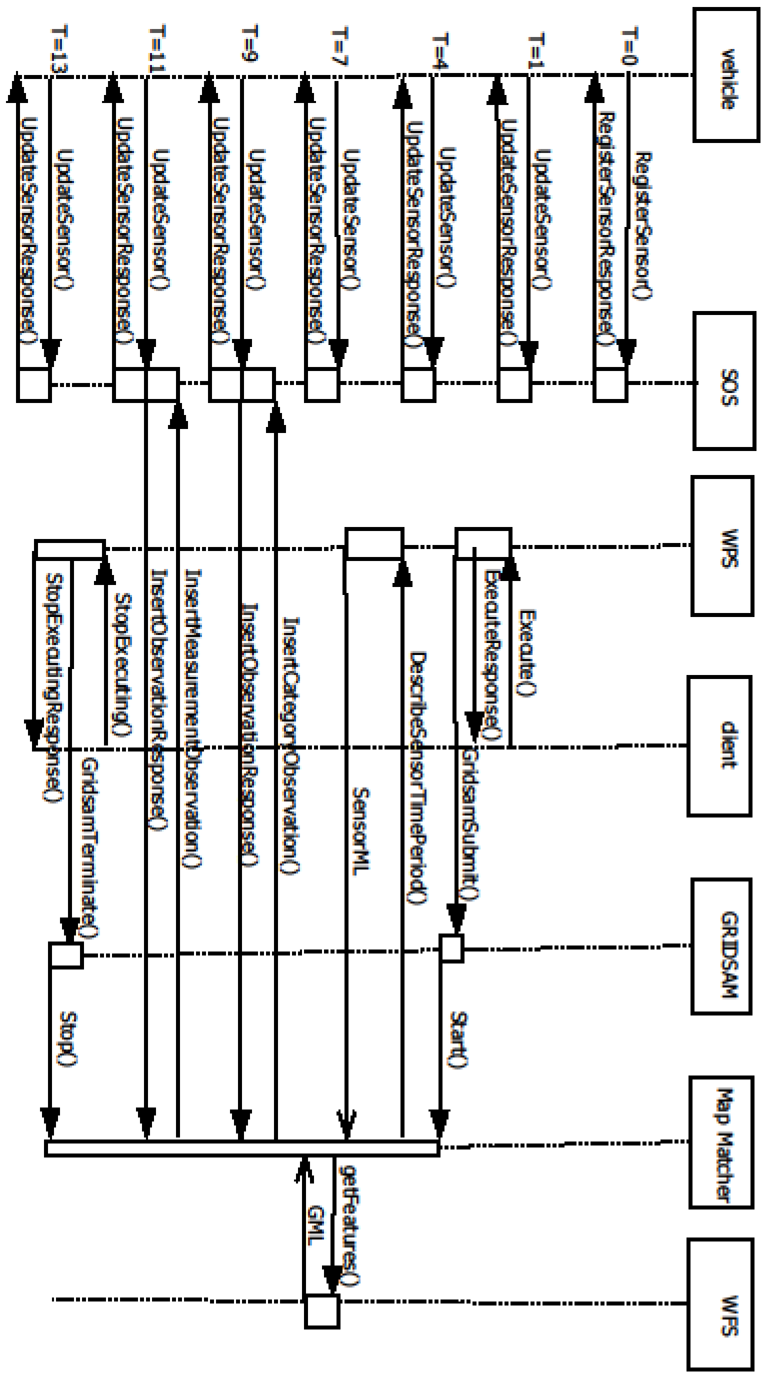


Figure 4.2: Message Sequence Diagram

Figure 4.2: Map-matching message sequence diagram

4.2.5 Review of Map-matching Algorithms

Map-matching is the process of reconciling the users location with the underlying map data (White *et al.*, 2000). Here we consider only global map-matching strategies that are concerned with identifying the most likely road segment within the network, as opposed to local map-matching that considers the position of a vehicle within a road segment (Hummel, 2006).

According to Jagadeesh *et al* (2004) map-matching algorithms can be divided into three main categories. The simplest category includes algorithms that consider only the geometric relationship between the road network and the GPS point, the next category also considers position history and topological information and the final class also uses probabilistic information to define a confidence region in which the vehicle is positioned.

The first category of map-matching algorithm can be further subdivided into four categories (Noh and Kim, 1998). Distance of point to point matching is the simplest of these techniques but it suffers from poor accuracy (Yang *et al.*, 2005). Using distance of point to point, matching is carried out by considering all the points of which the road network is comprised and matching the GPS measurement to the closest point. Distance of point to curve is a slightly more complex variation; matches are made by selecting the road with the shortest distance between any point on the road's sub-segment and the GPS measurement, i.e. the minimum distance from the road line to the GPS measurement. Although an improvement on the previously described technique, Yang *et al* (2005) reveal how distance of point to curve often results in a vehicle jumping from one road to another road running in parallel. The distance of curve to curve technique uses two GPS points and matches the road segment that has the shortest distance from the baseline between the two GPS measurements. Surprisingly, in a study carried out by (White *et al.*, 2000) it was found that curve to curve matching did not consistently outperform point to curve matching. The distance of angle to curve method uses two GPS measurements to calculate the bearing of the vehicle, and the match is made by finding the road sub-segment whose angle deviates from the bearing of the vehicle the least.

The second category of algorithm that incorporates historical and topological information introduces another layer of complexity, examples are provided by Greenfeld (2002) and White *et al* (2000). However, such algorithms have been proven to be fragile because one wrong match can lead to a whole series of wrong matches (Yang *et al.*, 2005).

The third category of algorithm utilises probabilistic techniques and performs better than the first category and can recover more quickly from a wrong match (Jagadeesh *et al.*, 2004). However they tend to suffer from increased computation time. Hummel (2006) describes an algorithm in this category that performs extremely well. It was found that only 0.4% of points were misclassified over a 15000 point sample. A Bayesian statistical method is used to perform the initial map-match, which relies only on the position and orientation of the vehicle. Accuracy is improved by using the road network topology in conjunction with the vehicle's position history to calculate probability distributions for each possible transition from one road to the next and the Viterbi algorithm (Forney, 1973) is used to find the best possible path based on these transition probabilities.

It has been shown that a wide variety of map-matching algorithms exist that use an array of available information and techniques. A trade-off clearly exists between speed of matching and overall accuracy. In this system the Bayesian statistical matching technique described by Hummel (2006) is used that relies only on vehicle position and orientation. For the sake of simplicity, position history and road network topology were not included in this implementation and it is therefore expected that the resulting matching accuracy will be less than optimal. Nevertheless, absolute accuracy of the map-matching algorithm is not the goal of this work; rather it is to demonstrate how real-time data from a collection of independent sensors can be processed in near real-time by running several concurrent processes on the grid.

4.3 Implementation

The implementation and deployment details of the map-matching system are outlined in this section. Figure 4.3 depicts the arrangement of components which are described in the following sub-sections.

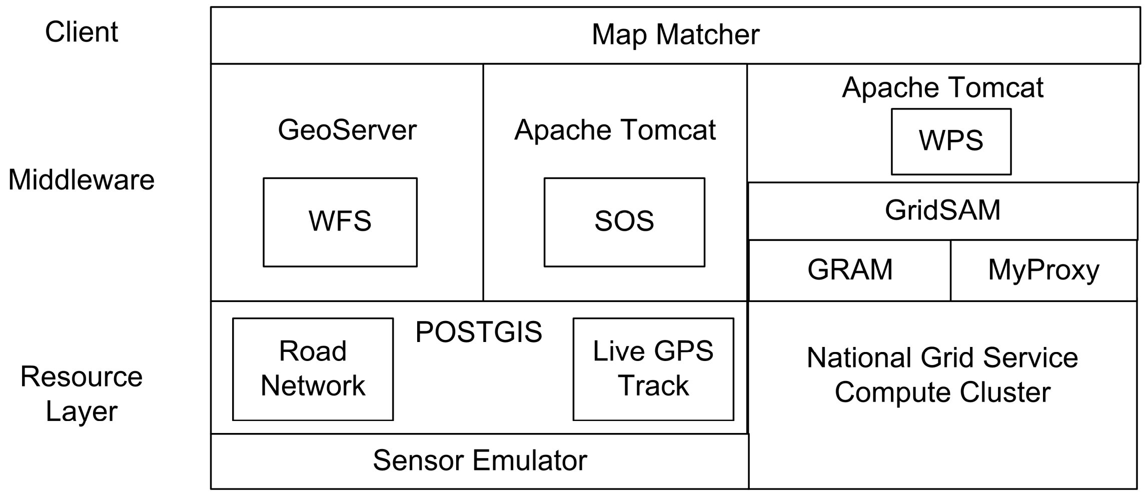


Figure 4.3: Map Matching System Component Diagram

4.3.1 Sensor Observation Service (SOS)

A 52 North SOS was deployed in a Tomcat container using PostGIS as the backend database system. The 52 North SOS complies with the version 1.0.0 schema (Na *et al.*, 2007), implementing the core and transactional profiles in addition to some of the operations in the enhanced profile. Detailed build and deployment steps are set out in Appendix A.

4.3.2 Sensor Emulator

The type of platform envisioned to be installed on each vehicle is a smart-phone or personal digital assistant, as these devices are small, portable, internet accessible and often contain built in GPS receivers. A Java desktop application was developed to emulate such a device, built using a Java Swing graphical user interface, PostGIS JDBC drivers and the Java Topology Suite. The function of this application is to read GPS position and time observations from a PostGIS database and to translate these into XML based *UpdateSensor* requests that conform to the SOS mobile profile (Stasch *et al.*, 2008).

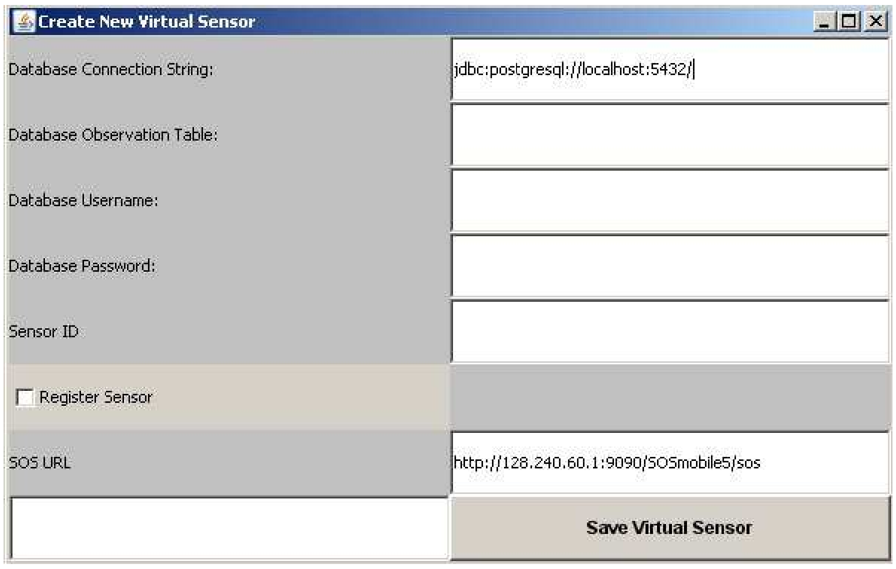
The first stage in developing this application involved loading GPS observations into a PostGIS database. GPS observations were sourced from a variety of vehicles travelling around Newcastle upon Tyne. This included data from the NCC fleet such as road sweeper vehicles, refuse disposal vehicles and work vans. In addition some of the data collected as part of the MESSAGE project (Blythe *et al.*, 2006) was used; this is data sourced from ordinary cars travelling around Gateshead equipped with sensors to measure air quality. The data was delivered in various comma separated text file formats; a comprehensive description of the steps followed to load this data into PostGIS is included in Appendix B.

The purpose of the Sensor Emulator is to enable users to easily emulate a series of in-vehicle GPS devices in order to test the other components in this system. It is anticipated that the in-vehicle mobile devices will update the SOS with their position via web service requests. Therefore the Sensor Emulator was designed to consecutively read GPS observations from the PostGIS database, transform them into SOS *UpdateSensor* requests, and send the requests to the SOS at regular intervals. The application was designed around the concept of a virtual sensor which is essentially a one to one mapping between a table of static GPS observations stored in a database and an SOS sensor. Virtual sensors can be created by specifying the connection parameters of the database from which to read observations, and the address of the SOS at which to publish position measurements. It is assumed that each observation table will conform to the schema in Listing 4.1. Figure 4.4 depicts a screenshot of the graphical interface used to create a virtual sensor. Once a virtual sensor has been created it must be saved; a process that serializes the mapping to a custom virtual sensor file format.

Figure 4.4 shows a screenshot of the application. To initiate the streaming of observations from a database to an SOS instance via a set of web service requests, the user must first select a file system directory containing one or more virtual sensors. When streaming is initiated by pressing the *start* button, observations are read sequentially from the database table of each virtual sensor in the directory, translated into web service requests and sent to their respective SOS instance. The time delay between each web service request can be specified by entering a number of milliseconds in the *delay* textbox, prior to starting the application. Additionally it is possible to use the system's current time, rather than the timestamp specified in the database observation table by checking the *use current time* checkbox. This option enables several vehicle tracks that were sampled at different times to be simulated together. It is possible to suspend, resume, stop and reset streaming. There is also an option to clear all observations from the SOS by connecting to its underlying database and resetting the data model by executing a SQL script.

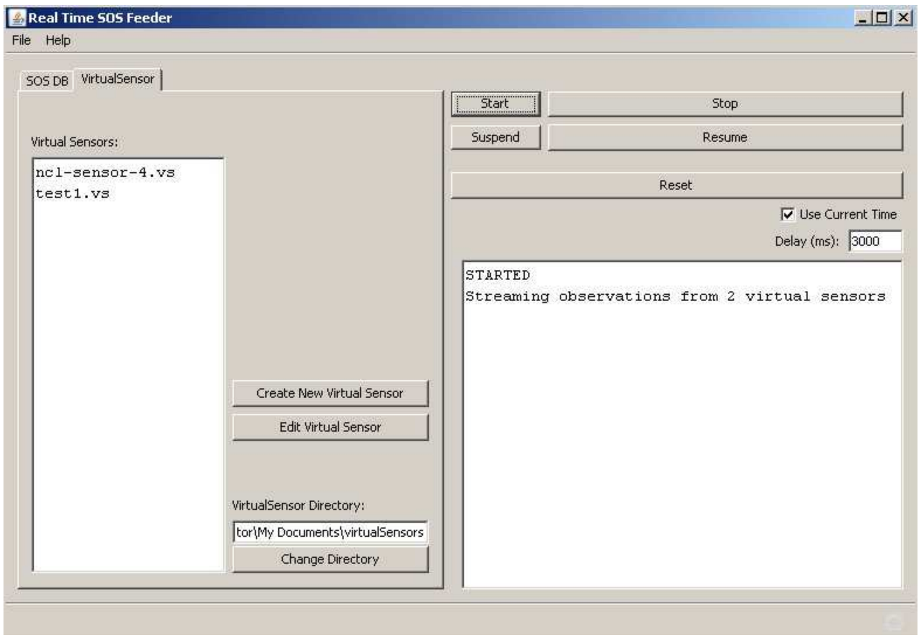
Listing 4.1: Schema of the Observation Table

```
Observations(id  integer,  elevation  real,  time_stamp  timestamp(4)
without timezone, the_geom geometry)
```


Create New Virtual Sensor

Database Connection String:	jdbc:postgresql://localhost:5432/
Database Observation Table:	
Database Username:	
Database Password:	
Sensor ID	
☐ Register Sensor	
SOS URL	http://128.240.60.1:9090/SOSmobile5/sos
	Save Virtual Sensor

Figure 4.4: Screenshot of the graphical interface to create a virtual sensor



```
ncl-sensor-4.vs
test1.vs
```

```
STARTED
Streaming observations from 2 virtual sensors
```

Figure 4.5: Screenshot of the graphical user interface of the Sensor Emulator

4.3.3 Web Feature Service (WFS)

Ordnance Survey MasterMap® Integrated Transport Network™ (ITN) is the definitive road structure dataset for the UK; it details the current topology of the UK's road network. Furthermore, each road link is assigned a unique Topographic Identifier (TOID) which enables the dataset to be stored in a spatial database using the TOID as a primary key. Consequently, ITN was selected to represent the underlying road network in this system. Although ITN data is a commercial product it is available free of charge from Digimap (http://www.edina.ac.uk/digimap) for academic purposes.

Geoserver was used in this system to serve road network data in WFS format. Geoserver is capable of serving spatial data in a number of OWS formats. The served data can be stored in a variety of file or spatial database formats; PostGIS was selected to store the ITN data. Initially the tool OGR2OGR, part of the GDAL/OGR translator library, was used to import the road network data into PostGIS from the GML format (Portele, 2007) in which it is supplied by Digimap. However, it became apparent that this tool did not preserve the TOID of each feature, a shortcoming resulting from the way in which data is delivered by Digimap. Although ITN is a spatially continuous dataset, it is served by Digimap as a spatial tessellation of tiles. When a feature is requested that straddles a tile boundary, the feature is served twice, thus destroying the uniqueness of each feature and preventing the features from being inserted into a database because Codd’s First Normal Form (Codd, 1972) is not adhered to.

Another data loading tool, InterpOSe was selected to import data into PostGIS. InterpOSe is supplied by a commercial organisation Dotted Eyes and automatically discards duplicate features. It contains a wizard based interface to import data from a GML file into a variety of formats, although PostGIS is not supported. This tool was therefore used to convert the road network GML file into ESRI shapefile format. Subsequently OGR2OGR was used to import the shapefile data into a PostGIS database.

Unfortunately the version of PostGIS that was used, version 1.3.5 relies on an underlying Proj4 library (version < 1.6.2) that contains an error in the British National Grid record of the *spatial_ref_sys* table that results in incorrect coordinate transformations. This was corrected by executing the SQL commands stated in Listing 4.2 from the Postgres shell, before importing data into the database. The entire procedure of loading the ITN data in listed in Appendix C.

Listing 4.2: Correction to spatial_ref_sys table in PostGIS

```
UPDATE  spatial_ref_sys  SET  proj4text=  ‘+proj=tmerc  +lat_0=49
+lon_0=-2  +k=0.999601  +x_0=400000  +y_0=-100000    +ellps=airy
+units=m +no_defs +datum=OSGB36’ WHERE srid=27700;
UPDATE   spatial_ref_sys   SET   proj4text   =   '+proj=longlat
+ellps=airy +datum=OSGB36 +no_defs' WHERE srid=4277;
```

4.3.4 Map Matcher

A Java application was developed to perform the actual map matching processing operation. Unlike many of the other components in this system, the map matching program was created as a self contained program rather than a service because it is designed to be run as a grid processing job. The chosen algorithm uses Bayesian classification to assign a GPS observation to a road segment; it is based on the work of Hummel (2006) and is described as follows:

A minimum of two raw observations are required to estimate position on a road network; 1 (X1,Y1) and 2 (X2,Y2) in order to derive vehicle orientation. We can assume that position has been observed by a commercial off the shelf GPS receiver operating in the standalone coarse acquisition mode. As such we can approximate the standard deviation of horizontal position to be 12.5 metres and the standard deviation of orientation, as calculated from the whole circle bearing between X1, Y1 and X2, Y2 to be 15°. Initially, a section of map data is retrieved centred on the most recent observation X2, Y2 by performing a bounding box query on a road dataset. We can now derive the road segment from which the most recent observation was most likely to have been taken using the algorithm detailed in Listing 4.3. A graphical overview of the components in this algorithm is given in Figure 4.6.

A simple Java command line program was developed to implement this algorithm. Furthermore, the program was designed to interact directly with OGC services, thus it is capable of extracting observation geometry and time stamp from a SOS and road network geometry from a WFS.

The map matcher program can be initiated with either five or seven parameters. If the program is initiated with all seven parameters it performs map matching only for the time period between the start time and the end time, it then exits normally. However, if the program is executed with only the five mandatory parameters it runs in real-time mode. In this mode the program performs map matching for each of the observations taken in the past 60 seconds, i.e. between the system’s current time and 60 seconds before the system’s current time. The program then waits until another minute has elapsed and repeats the operation. It continues to run every 60 seconds until it is forced to exit by user intervention.

On initialisation the program contacts the SOS and retrieves the observations for the specified vehicle over the specified time period. This is achieved using the 52North mobile schema (Stasch *et al.*, 2008) which offers a *DescribeSensor* operation which returns a SensorML document detailing the position history of the sensor. The position observations are then parsed into Java Topology Suite format and the time into *java.util.GregorianCalendar* format.

Listing 4.3: Map Matching Algorithm using position and orientation

```
muB:         Standard deviation of horizontal position (metres)
muDeltaPhi: Standard deviation of orientation (degrees)
dist:        Euclidean  distance  between  vehicle  position  and
road               segment (metres)
deltaPhi:    Angular difference between vehicle orientation and
                   orientation of the road element (degrees)
iTemp:       the  cost  associated  with  X2,Y2  belonging  to  the
                   current road subsegment
ii:           the minimum cost of all road subsegments examined
so           far
muB= 12.5
muDeltaPhi = 15
dist, deltaPhi=0
ii = +∞

for each road subsegment
(
dist = shortest distance between X2,Y2 and road subsegment
deltaPhi = difference between road subsegment orientation and
vehicle orientation
iTemp = (dist² / muB) + (deltaPhi² / muDeltaPhi)

      if( iTemp < ii)   {
                        ii = iTemp
                        result = this road subsegment
                        }
}
```

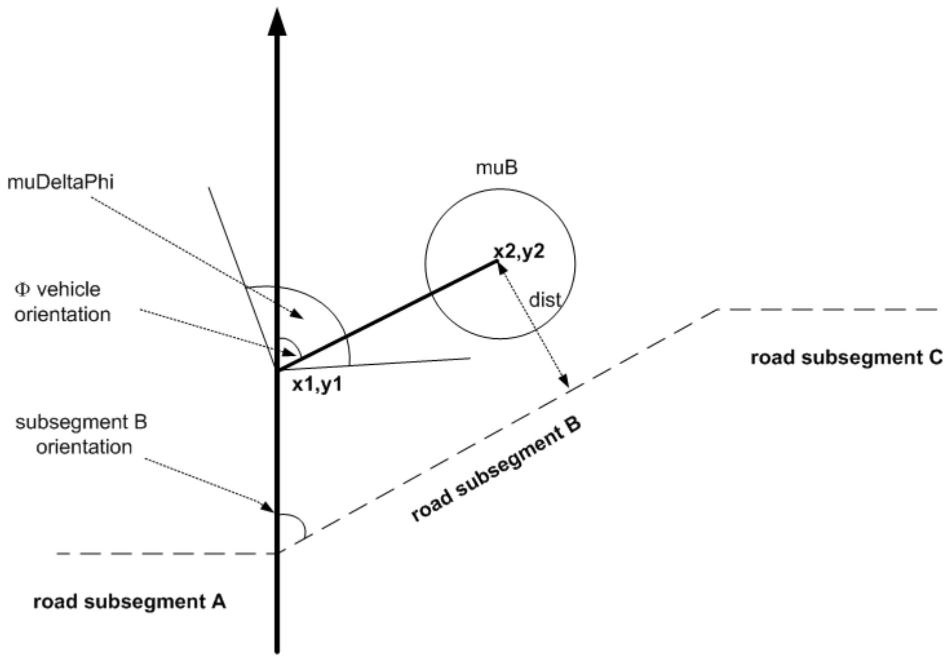


Figure 4.6: Diagrammatic representation of the map-matching algorithm showing the vehicle's current (X2,Y2) and previous (X1,Y1) positions, the

standard deviation of horizontal position (muB), and the standard deviation of orientation (muDeltaPhi) in relation to the position and orientation of road sub-segments

For each observation, the position is converted from WGS84 to OSGB36 and a WFS bounding box query is constructed, centred on the observation. The query is executed on the WFS using the supplied connection parameters, thus retrieving a relevant subset of the road network dataset. The program invokes the map matching algorithm for each observation using the road network data extracted from the WFS and the GPS observations extracted from the SOS. Next, SOS requests are constructed to insert the results of the map matching algorithm, i.e. the road identifier, into the SOS. This is achieved using the transactional *InsertObservation* operation; the road identifier is inserted as a Category Observation (Cox, 2007). As a by product of this operation the orientation of the vehicle is calculated and this is also inserted as a Measurement Observation (Cox, 2007), again using the *InsertObservation* operation.

4.3.5 Web Processing Service (WPS) Proxy

The WPS provides an OGC compliant interface through which to submit geoprocessing tasks. As the goal of this system is to forward processing operations to the grid, the WPS itself does not carry out the processing, it merely translates the execution request into JSDL (Anjomshoaa *et al.*, 2005) and submits it to a grid endpoint to be processed remotely and asynchronously.

52 North provide an open source WPS implementation that complies with the version 1.0.0 specifications. Unfortunately the current specifications have limited support for asynchronous processing; there are no operations defined to pause, cancel or restart asynchronous processing tasks. Recently a number of change requests have been submitted to the OGC to rectify this problem, namely 09_093 (Woolf and Shaon, 2009b) and 09_109 (Woolf and Shaon, 2009c). However, these are likely to undergo a lengthy discussion and modification process before they are included in a revised version of the

specification. It was therefore decided to modify the 52 North implementation to meet our requirements, albeit in a manner that doesn't completely conform to the OGC specifications.

The goals of the modified service are the ability to submit jobs to a grid infrastructure using the *Execute* request, and to add a new operation *StopExecuting* that enables continuous running jobs, such as the map matcher, to be terminated. It should be noted that the 52 North implementation already contains a grid module that enables developers to implement their own task-farm style processing algorithm and to submit it to a Unicore (http://www.unicore.eu) infrastructure. It was opted not to extend this module but to start afresh for the following reasons. Firstly, the embedded grid module, although it carries out processing remotely, returns the results in a synchronous fashion; i.e. embedded in the *ExecuteResponse* document. In our scenario the results are automatically inserted into an SOS and therefore we simply require a reference to our asynchronous job to be returned. Secondly, the embedded module is designed to execute a task-farm style scenario whereas the extension we are trying to implement simply executes one long-running process. Finally, the embedded module is of a flexible design and provides interfaces to enable any grid infrastructure to be plugged in. For simplicity and ease of development it was decided to implement our own interfaces, although it is noted that there would be benefit in streamlining this code for a future production version to avoid code duplication.

Initially, a *GridJobManager* interface was created that enables other developers to plug in their own grid job submission interfaces. Subsequently an implementing class was coded for the GridSAM web service based job submission endpoint. GridSAM exposes both a Java API and a command line API. The Java API initially appeared the most flexible and easy to use; however in practice it proved difficult to use in a service environment as it requires different build and runtime classpaths and a number of system properties to be set. Eventually it was decided to use the command line API through the Java *Runtime.getRuntime.exec* method to manage job submission and termination.

52 North provide an abstract class ‘AbstractAlgorithm’ to enable developers to code their own geoprocessing algorithm. This class was extended by appending a new field to hold the class reference of the job submission class. Additionally, get and set methods were appended and the new abstract class was named ‘AbstractAsynchronousAlgorithm’. The relationship between these classes and interfaces is shown in Figure 4.6 in UML notation.

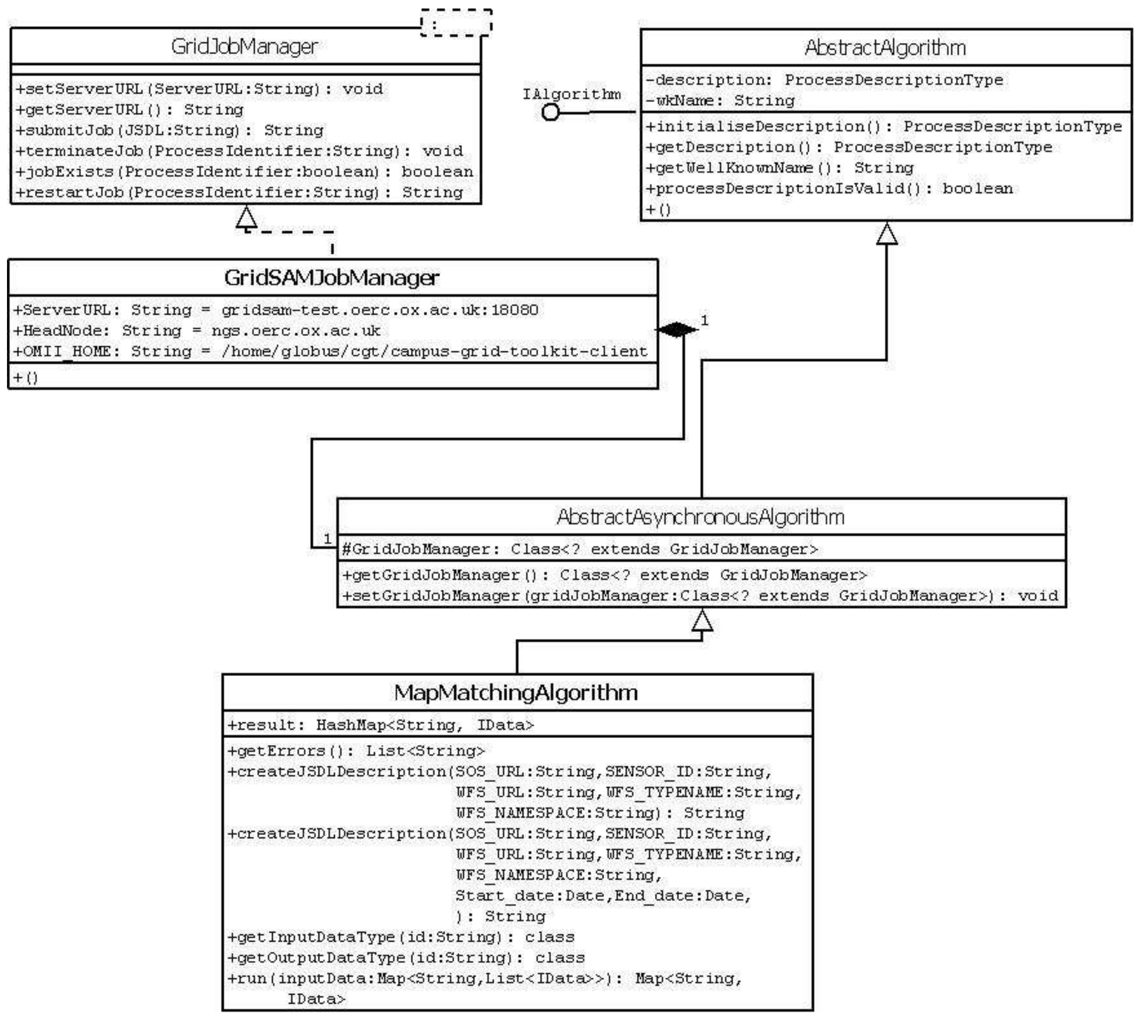

GridJobManager
+setServerURL(ServerURL:String): void
+getServerURL(): String
+submitJob(JSDL:String): String
+terminateJob(ProcessIdentifier:String): void
+jobExists(ProcessIdentifier:boolean): boolean
+restartJob(ProcessIdentifier:String): String
IAlgorithm
AbstractAlgorithm
-description: ProcessDescriptionType
-wkName: String
+initialiseDescription(): ProcessDescriptionType
+getDescription(): ProcessDescriptionType
+getWellKnownName(): String
+processDescriptionIsValid(): boolean
+()
GridSAMJobManager
+ServerURL: String = gridsam-test.oerc.ox.ac.uk:18080
+HeadNode: String = ngs.oerc.ox.ac.uk
+OMII_HOME: String = /home/globus/cgt/campus-grid-toolkit-client
+()
1
1
AbstractAsynchronousAlgorithm
#GridJobManager: Class<? extends GridJobManager>
+getGridJobManager(): Class<? extends GridJobManager>
+setGridJobManager(gridJobManager:Class<? extends GridJobManager>): void
MapMatchingAlgorithm
+result: HashMap<String, IData>
+getErrors(): List<String>
+createJSDLDescription(SOS_URL:String,SENSOR_ID:String, WFS_URL:String,WFS_TYPENAME:String, WFS_NAMESPACE:String): String
+createJSDLDescription(SOS_URL:String,SENSOR_ID:String, WFS_URL:String,WFS_TYPENAME:String, WFS_NAMESPACE:String, Start_date:Date,End_date:Date,): String
+getInputDataType(id:String): class
+getOutputDataType(id:String): class
+run(inputData:Map<String,List<IData>>): Map<String, IData>

Figure 4.7: UML Diagram showing Algorithm and Grid extensions to 52 North WPS

To represent our map matching algorithm, the ‘AbstractAsynchronousAlgorithm’ class was extended with a concrete class ‘MapMatchingAlgorithm’. This class implements the run method which is invoked when a client calls the *Execute* request for the map matching algorithm. The method parses the input parameters and ensures that they are correct, otherwise an exception is thrown.

It then sets the ‘GridJobManager’ to ‘GridSAMJobManager’ and creates a JSDL document by passing the input parameters to a *CreateJSDL(String[] arguments)* method. The job is submitted using this GridJobManager and the resulting process identifier is returned.

Immediately after an *Execute* request is received by the WPS an *ExecuteResponse* document is returned. However, this doesn’t give the WPS a chance to submit the job to GridSAM and return the unique job identifier created by GridSAM to the client. Nonetheless the 52 North WPS has a built in *AfterExecute* method that is run after the execute operation has completed and is used to update the *ExecuteResponse* if the client refreshes their page. A clause was appended to this method to return the job identifier to the client as a web accessible reference. This is an inbuilt feature of the WPS specifications that enables voluminous processing results to be stored at a web accessible location rather than consuming network bandwidth by delivering them directly.

Due to the real-time and continuous nature of our process, once an *Execute* request has been submitted to the service, the process will run eternally. There is clearly a need then to be able to stop this process. To plug this gap in the specifications a *StopExecuting* operation was created. For simplicity this operation can only be submitted as an HTTP GET request. Listing 4.4 gives an example *stopExecuting* GET request for a WPS at http://myWpsServer/wps and a job with process identifier urn:mygridsamjob:id:123. Listing 4.5 details the *stopExecuting* response.

Listing 4.4: StopExecuting Request

```
http://myWpsServer/wps?request=stopExecuting&service=wps&version=1
.0.0& job_id=urn:mygridsamjob:id:123
```

Listing 4.5: StopExecuting Response

```
<StopExecutingResponse>
<Response urn=”urn:mygridsamjob:id:123”> OK </Response>
</StopExecutingResponse>
```

This operation was implemented by extending the Request and Response classes in the 52 North WPS; it requires a single parameter ‘job_id’ which is the unique identifier of the job to be terminated. A *StopExecuting* request calls the *terminateJob* method of the GridJobManager’s implementing class. If successful this will return a very simple *StopExecutingResponse* document (Listing 4.5) that confirms that the job has been successfully terminated, otherwise an exception is thrown. The exact format of the request and response patterns for such an operation is a matter for the OGC WPS Working Group; current change requests indicate that the ability to stop an asynchronous process is more likely to be done using cancel, restart and pause operations. However, in the absence of official guidelines the *StopExecuting* operation provides a temporary solution.

4.3.6 GridSAM Client

GridSAM is an open source job submission and monitoring web service developed by OMII UK (http://www.omii.ac.uk). It supports JSDL and tracks the OGSA-BES job submission standard as well as providing a native interface. GridSAM can be connected to a wide variety of distributed resource managers such as Globus (http://www.globus.org), Condor (http://www.cs.wisc.edu/ condor/), Unicore (http://www.unicore.eu) or PBS; its role is not to carry out processing itself but to provide an OGSA compliant endpoint through which to submit jobs. It is up to the host to connect the service to a distributed resource manager. GridSAM was chosen as a job submission endpoint, primarily because it supports JSDL and OGSA-BES, and because it enables jobs to be submitted to a variety of distributed resource managers without any reconfiguration at the client end. In addition to its OGSA-BES WSDL interface, GridSAM also exposes a native job submission interface through both the command line and through a Java API.

Jobs were submitted to a GridSAM endpoint hosted by the UK NGS at https://gridsam-test.oerc.ox.ac.uk:18443/gridsam/services/gridsam by the WPS proxy component, using the native command line interface. There are two prerequisites for successfully running a job on GridSAM; firstly a valid proxy must be uploaded to the MyProxy server specified in the JSDL and secondly if file staging is required then a GSI-SSH connection must be established between the submission node and the GridSAM head node from which the files are staged. Grid Security Infrastructure SSH (GSI-SSH) is an extension of Secure SHell (SSH); a secure connection protocol between two computing nodes that supports authentication and encryption. The grid enabled version of this protocol includes support for grid authentication and credential delegation. The entire sequence of interaction between components is illustrated in Figure 4.2.

4.4 Results

4.4.1 Functionality Testing

Tests were carried out initially to verify that the map-matching system was working correctly. Two journeys were selected from the available GPS data. Journey 1 was sampled by a car travelling from Washington into Gateshead and back on major arterial roads taking readings every 2 seconds. Journey 2 was sampled by another car monitoring air quality in a residential area of Gateshead taking readings every 5 seconds. It was opted not to test the system on data from council vehicles due to their atypical behaviour; for example refuse disposal vehicles are continually stopping and starting.

Figure 4.8 and 4.9 show the results of applying the map-matching algorithm to Journey 1 and Journey 2 respectively. Sampled GPS observations are displayed as black dots, and matched roads are displayed as black lines. Roads that were matched incorrectly, i.e. roads that weren’t actually travelled on are shown in dark grey. In Figure 4.7 the scale box shows the southern part of the route at an enlarged scale as this contains the majority of the mismatches.

For each observation the map matching algorithm generates one of three possible states; a correct match, a mismatch or a null match. Null matches occur when the algorithm cannot find a road segment that correlates to the GPS measurement and mismatches occur when a measurement is incorrectly assigned to a road segment. A correct match occurs when a measurement is correctly assigned to the road segment from which it was captured. Using only the GPS measurement data it is difficult to determine whether a match is correct or not, even if the actual path of the vehicle is well known (White *et al.*, 2000, Brakatsoulas *et al.*, 2005). For example, at road junctions an observation may be erroneously matched to the vehicle's previous or future road and this would be difficult to detect without validating which road the vehicle was actually on at the time of measurement. As validation data is not available such errors will be ignored and all observations matched to a road on which the vehicle actually travelled are considered correct.

The percentages of correct matches for Journey 1 and Journey 2 are shown in Table 4.1. Journey 2 is comprised of a large number of measurements taken from car parks and therefore a considerable amount of null matches resulted. Because the vehicle was not actually on the road network when these measurements were taken these null matches cannot be considered to be erroneous. Therefore an additional row has been appended to Table 4.1 showing correct matches in Journey 2, excluding the car park observations.

To assess the performance of the system, the length of time between the instant at which the GPS measurement was taken, and the instant at which the matched road was inserted into the SOS, was measured. These results are displayed in Table 4.2.

Table 4.1: Percentage of Correct Matches for Journey 1 and Journey 2

	No. observations	Null matches	Mismatches	Correct Matches	% correct matches
Journey 1	709	1	43	665	93.7%
Journey 2	1172	184	35	953	81.3%
Journey 2 excluding car park observations	898	42	18	838	93.3%

Table 4.2: Time interval between GPS measurement and insertion of observation into Sensor Observation Service

	Sampling Frequency (seconds)	Mean Time Interval (seconds)	Standard Deviation
Journey 1	2	54.30	24.96
Journey 2	5	35.57	13.25

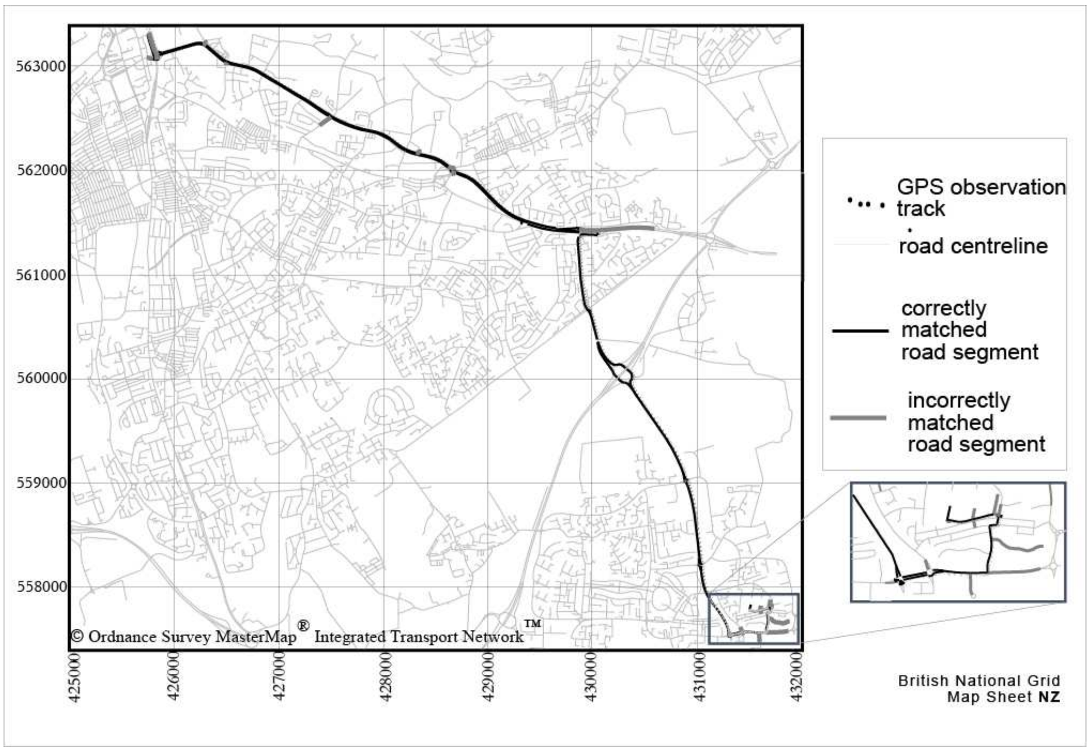


Figure 4.8: Map Matching Results for Journey 1

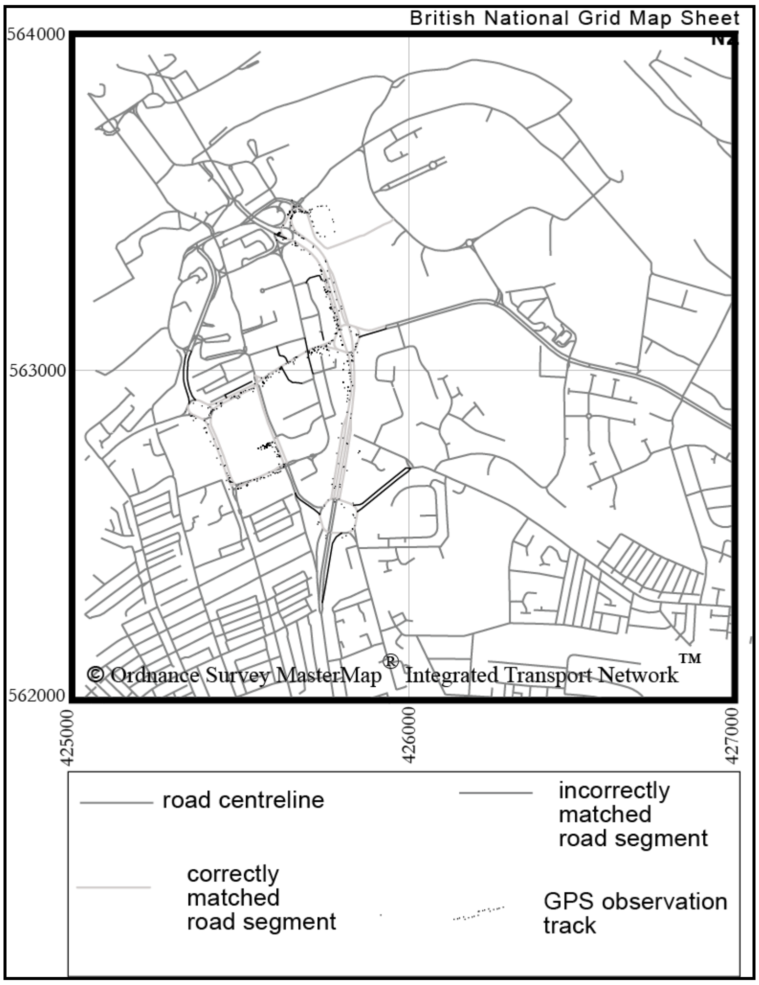

British National Grid Map Sheet
NZ
564000
563000
562000
425000
426000
427000
© Ordnance Survey MasterMap® Integrated Transport Network™
road centreline
incorrectly matched road segment
correctly matched road segment
GPS observation track

Figure 4.9: Map Matching Results for Journey 2

It can be seen from these results that the map-matching algorithm performs well. In each case the algorithm is matching over 90% of observations correctly, and the majority of results are inserted into the SOS within 60 seconds.

In many cases the mismatched observations are surprising, from Figure 4.8 and 4.9 it can be seen that the majority of mismatched roads are side-roads with a very different orientation to the actual path of the vehicle. In each of these cases the incorporation of position history into the algorithm is likely to correct these mismatches.

It can also be seen in Table 4.2 that the interval between GPS measurement and insertion of results is considerably lower for Journey 2. This can be attributed to the lower sampling rate of Journey 2. The map-matcher extracts position observations from the SOS for the previous 60 seconds and matches these before inserting them into the SOS as new observations. The lower sampling rate of Journey 2 means that only 12 positions must be extracted and matched each minute, as opposed to Journey 1 for which 30 positions must be processed.

4.4.2 Scalability Testing

So far it has been demonstrated that the map-matching system meets basic functional requirements for the operation of a single vehicle. The algorithm meets basic performance requirements and has been implemented as part of a service oriented system, thus enabling processing to be executed on a remote grid cluster and invoked through standards compliant interfaces. However, the goal of this work is not to evaluate the performance of the map matching algorithm. Rather it is to demonstrate the applicability of a grid based processing architecture to the problem of multiple real-time data stream instances, and to identify any potential bottlenecks. To this end a number of load tests were conducted to identify weak components and acquire system capacity information. The system was tested against its ability to process multiple data streams concurrently.

In this system a new map-matching instance is launched on a remote grid node every time a vehicle comes online. The scale-out approach adopted in this design necessitates a one-to-one relationship between the vehicle data stream and the map-matcher instance; however it is possible for several vehicles to store their observations in the same SOS repository and for each map-matcher instance to query the same WFS. This many-to-one relationship between vehicle/map-matcher and SOS/WFS is depicted in Figure 4.9. While it is desirable to minimise the number of SOS and WFS instances, there is a finite limit to the number of concurrent requests each of these components can handle, imposed by network bandwidth constraints and limited processing capacity.

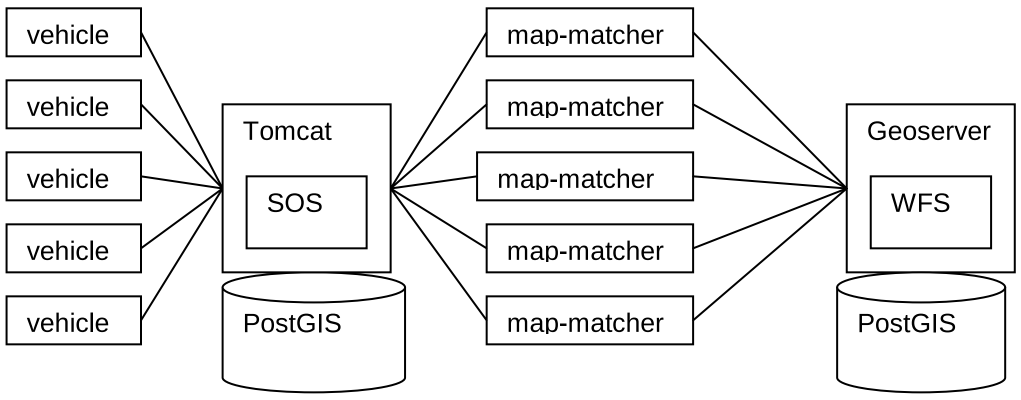


Figure 4.10: Many to one relationship between vehicle/map matcher and SOS/WFS

Tests were carried out to find the maximum number of vehicle data streams that can be assigned to each SOS and WFS without having an adverse affect on performance. Load testing was conducted using JMeter to simulate several concurrent requests to each service and to monitor the response time. In each case JMeter was used to steadily increase the number of concurrent requests over a ramp-up period of 60 seconds after which time the number of concurrent requests was maintained. The results from this test are shown in Table 4.3 and Figure 4.11; response times are given in milliseconds and are averaged over 1500 requests that were sampled after the ramp-up period. The maximum number of concurrent requests that could be maintained was found to be 560 for the WFS and 200 for the SOS; above these figures HTTP error codes were returned. It should be noted that the WFS contained all the road features in the Tyne and Wear output area; 95803 features in total. Better performance was observed with smaller database sizes.

Table 4.3: WFS and SOS Response Time

	Response Time (milliseconds)			
Number of concurrent requests	WFS Get Feature	SOS Describe Sensor	SOS InsertCategory Measurement	SOS Insert Observation
1	316	88	93	101
10	386	88	474	551
20	527	88	957	1098
50	3488	98	2332	2910

100	8921	118	4527	5688
200	20710	1391	10190	10867
300	31999	2918	N/A	N/A
400	43405	N/A	N/A	N/A
560	59963	N/A	N/A	N/A

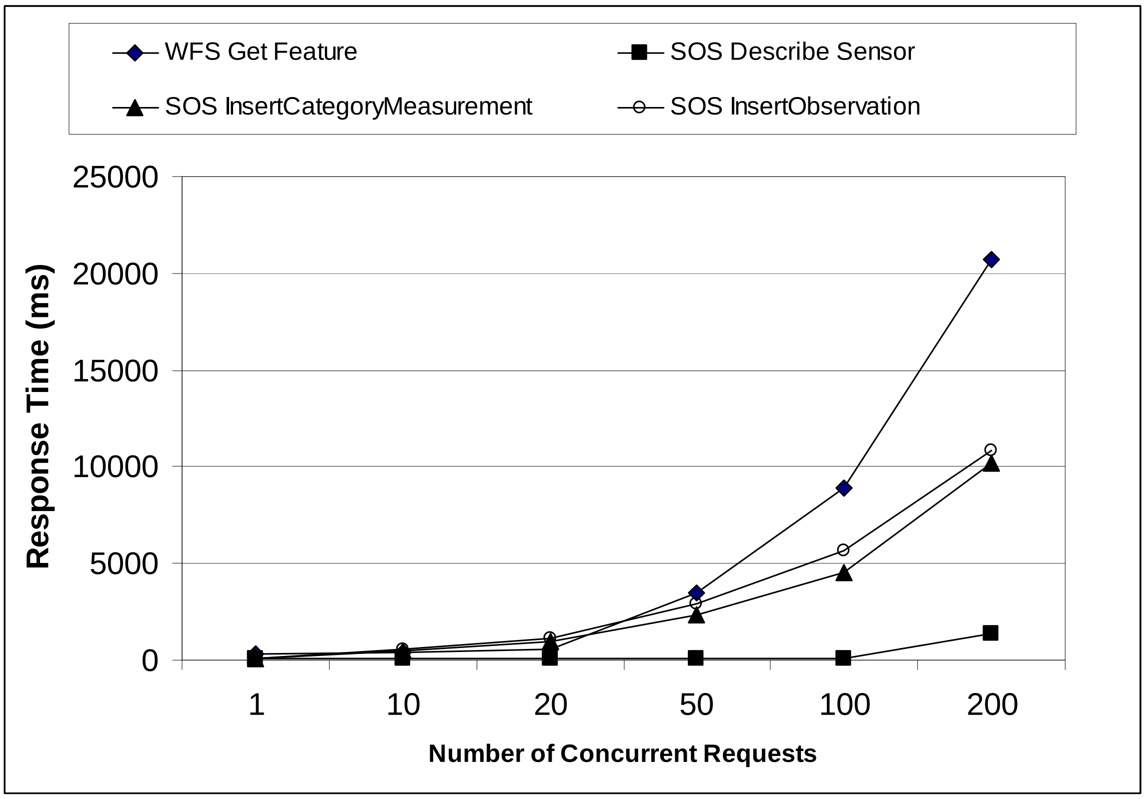


Figure 4.11: Graph showing Response Time of SOS and WFS requests

These response time metrics provide useful information from which to deduce the most appropriate ratio of SOS / WFS to vehicle data streams. Using a 2 second sampling rate the ratio of *describeSensor* requests to other requests is 1:30 because each *describeSensor* request retrieves all the observations recorded in the previous 60 second cycle whereas the other requests are made for each observation.

Because this system operates on a single execution thread there is a danger that time-lag could build in the system. This situation would arise if the entire map-matching cycle were not to complete within the 60 second time period and would result in an ever increasing time interval between each observation being recorded and being processed. To avoid this scenario the map-matcher was profiled to find the average execution time of each processing stage and these results were subsequently used to select the ratio of WFS and SOS to data streams. Profiling was carried out on a single instance of the map-matching program using Netbeans (http://netbeans.org); results are detailed in Table 4.4. Response times from both the WFS and SOS services were disregarded during profiling as each of these services were running locally and so the results are likely to be unrealistic, however, their response times have already been established (Table 4.3). Based on the combined results of profiling and SOS / WFS response times it was decided to allocate 20 data streams per WFS and 10 data streams per SOS; the cumulative time expenditure of this configuration is under the 60 seconds maximum and is summarised in Table 4.5.

Table 4.4: Profiling results for map matcher

Procedure	Number of operations per minute	Processing Time (ms)	Actual Processing Time (ms)
Generate describeSensor request	1	3	3
Parse SensorML	1	29	29
Generate WFS query	30	112	3360
Parse GML and perform match	30	328	9840
Generate insertObservation and insertCategory Measurement requests	30	0	0
Total	92	472	13232

Table 4.5: Map matcher time expenditure (milliseconds)

Processing Time	13232
10 x describeSensor requests	88
10 x 30 x InsertCategoryMeasurement requests	14220
10 x 30 x InsertObservation requests	16530
30 x 20 WFS getFeature requests	15810
Total	59880

This configuration was tested by running the map-matching system over a one hour period for 20 vehicles concurrently and although this number only represents a fraction of the Newcastle City Council vehicle fleet it was deemed sufficient to test this system. This is because the maximum number of data streams assigned to a single component instance is 20, with the exception of the WPS and the GridSAM client that were not replicated as they were unlikely to present a bottleneck because they simply perform job submission. To capture performance data a minor modification was made to the SOS database in that a trigger was added that records for each observation the time difference between insertion into the SOS database and the actual time stamp at which the position measurement was taken; this difference is termed time-lag. The test was initiated by sending a series of WPS *Execute* requests to the WPS. It was found that the first 13 sensors were scheduled promptly within 2 – 3 minutes. However, due to the NGS scheduling policy, the remaining 7 jobs were not scheduled for another 25 minutes. Figure 4.12 and Figure 4.13 show the time-lag results for both SOS servers, each of which was assigned 10 data streams which are labelled as s1 – s10 and s11-s20 in Figure 4.12 and Figure 4.13 respectively. The mean time-lag for SOS Server 1 was found to be 1 minute 51 seconds with a standard deviation of 2 minutes 31 seconds. For SOS Server 2 the mean time-lag was 1 minute 3 seconds with a standard deviation of 1 minute and 4 seconds. The results show that of the initial 20 data streams, only 12 of these were still being processed after the one hour period; 5 from SOS Server 1 and 7 from SOS Server 2. This was found to be the result of deadlock occurring in the database due to a large number of concurrent requests preventing the affected sensors from inserting observations into the database. The map-matcher processes terminated after polling the SOS and finding no observations for the latest 60 second time period.

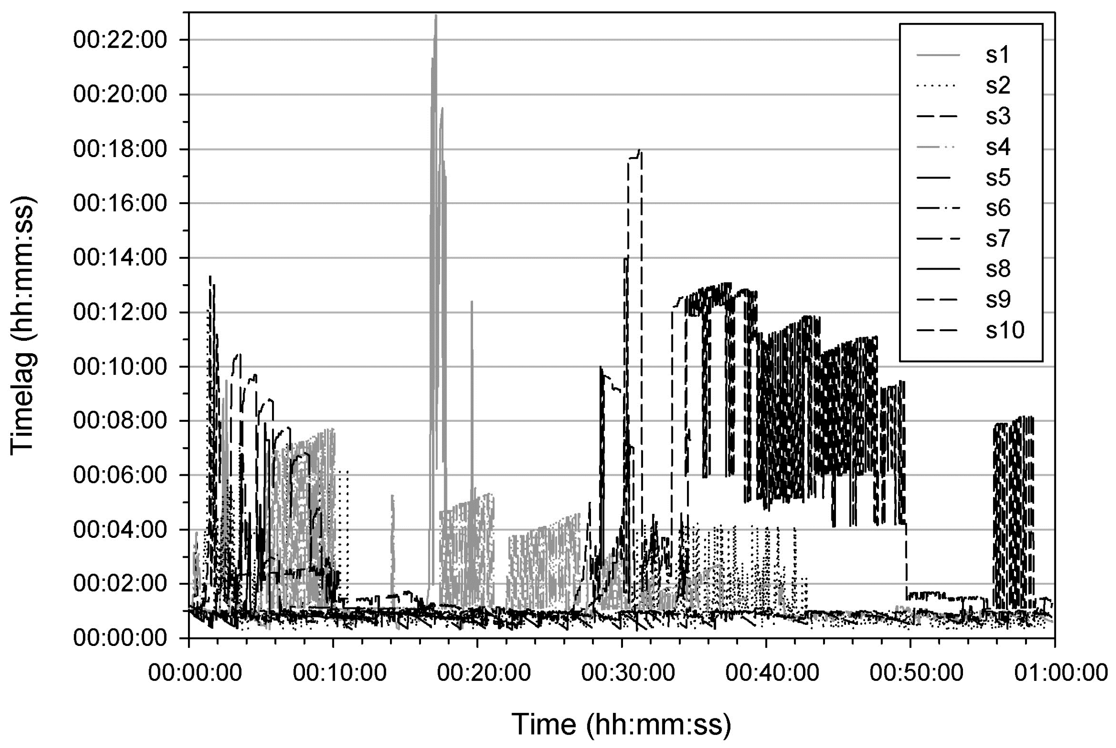


Figure 4.12: Time-lag results for SOS Server 1

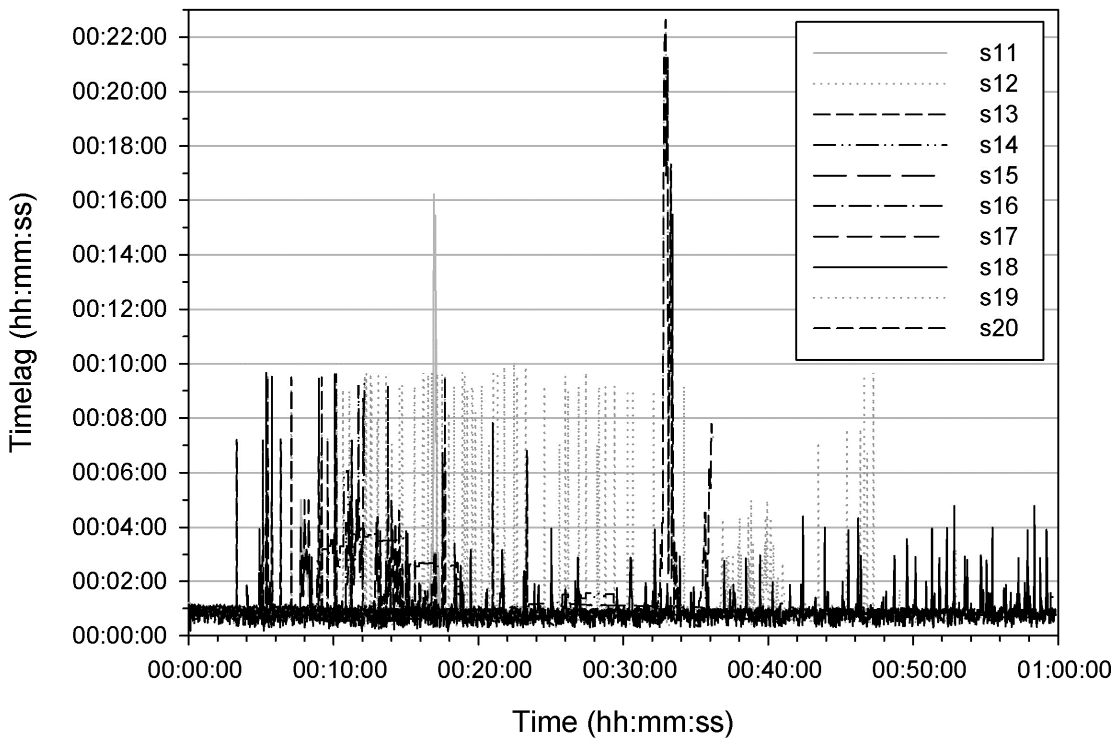


Figure 4.13: Time-lag results for SOS Server 2

Of the remaining data streams it can be seen from Figure 4.12 and Figure 4.13 that the majority of observations are matched within the required 90 second time frame. However, the graphs also contain a number of spikes that represent latency in the processing chain, the most severe of which exhibits a 23 minute delay. These delays can also be attributed to deadlock occurring in the SOS database caused by too many concurrent requests. However, in each case it can be seen that the SOS recovers within a matter of minutes and there is no cumulative effect on time-lag in the medium or long-term.

The graphs also show that the performance of SOS Server 2 is considerably better than that of SOS Server 1; there are two possible causes for this. Firstly, SOS Server 1 is located on the same physical server as the WFS. Consequently both services share a database instance and the increased load on the database is likely to reduce its speed. Also, the increased load on network bandwidth is likely to have a similar effect on the throughput of the SOS in terms of the number of requests it can serve per second. Secondly, due to the delay in scheduling the processing jobs SOS Server 2 experienced a significantly lighter load for the first 25 – 30 minutes of execution as only three map-matching jobs had been scheduled at this stage.

4.5 Discussion

The design, implementation and testing of a real-time grid-based map-matching system has been described in this Chapter. The system is capable of performing map-matching for a fleet of vehicles and can be considered OGC compliant in that it uses compliant data repositories, WFS and SOS, and can be accessed through a compliant WPS interface. The system uses the NGS production grid to carry out processing which is accessed through a GridSAM web service and authenticated using a MyProxy credential delegation service.

The scalability results are surprising, it was anticipated that each SOS and WFS would have handled a much larger number of data streams than was found in testing. The main constraint in this system was found to be an I/O bottleneck at the SOS database that resulted in database deadlock. Marginal increases in performance could almost certainly have been gained by tuning the SOS and WFS server parameters such as connection pool size and Java Virtual Machine settings. Performance could also have been increased by reducing the load on the SOS either by only inserting the road identifier into it rather than also including the vehicle's bearing, or by inserting the results into an altogether separate SOS from the one containing raw position information. However, it seems unlikely that the gains in performance brought from these changes would be sufficient to make the system viable.

To implement the system in its entirety for the Newcastle City Council fleet of 890 vehicles would require a minimum of 89 SOS servers. It could be argued that this hardware requirement negates the benefit of the distributed approach adopted here. The primary driver for this approach is the ability of the system to scale up and down as demand fluctuates whilst minimising the use of computing resources and therefore maintaining a large server cluster to handle SOS transactions is unviable. One solution would be to host these SOS servers in the cloud, which is often cited as a solution to the problem of scaling web applications on-demand by means of resource virtualisation and dynamic provisioning (Buyya *et al.*, 2008, Vaquero *et al.*, 2008). However, considering that the processing cost of the map-matching operation is trivial in comparison to the communication costs of the system (Table 4.5), if this approach were to be adopted it may be easier for each cloud node to carry out the map-matching processing locally rather than porting the computation to a processing grid.

It is anticipated that latency in this system could be reduced through better program design. For example, a multi-threaded program could continually poll the SOS for new observations on one thread, perform the processing on another, and update the SOS with processing results on a third thread.

The concept of using grid computing resources to achieve high throughput processing of near real-time data is original and these results show that it is possible. However, there are a number of issues with running this type of problem on a production grid. Firstly, job scheduling is unpredictable and is dependant on a number of factors such as the cluster availability, fair usage policy, user's priority level and the anticipated size of job. The NGS indicated that they would be willing to prioritise jobs related to the monitoring and mitigation of natural disasters, as these are both time and safety critical and for the greater public good. However, they are not willing to prioritise jobs related to road traffic monitoring and management and so a dedicated grid cluster would be required to implement a production version of this system.

The second issue is that this approach could be considered an inefficient use of resources. The results show that processing is only being performed for a small percentage of the time that the map-matching job is running; the rest of the time is spent waiting for new data to arrive. By assigning one data stream to each processor it is unlikely that the computational load of the data stream will perfectly match the processors capabilities and therefore it is almost unavoidable that the processor will either experience a processing backlog or will spend time idle. In a grid environment resource usage is typically measured by the number of CPU hours used and so it is advantageous to fully utilise each processor whilst a job is running, particularly in the pay-per-usage grids we are likely to see in the future. Thus it can be concluded that the single processor per data stream approach is inflexible, although this style would be useful for a particular type of job that require as much processing to be done as possible in the time available such as iterative convergence problems. An alternative for more computationally intensive problems would be to perform data stream partitioning by sending different sections of the data stream to different processors, although in this case each section would suffer a scheduling delay and there would be no guarantee that the processing jobs would be executed in the order they were submitted.

The third issue is that both the OGC WPS interface and most grid job submission interfaces are not strictly designed to submit open-ended compute jobs. As such, neither interface explicitly exposes a *stopExecuting* operation. However, the OGSA-BES interface does expose operations for managing asynchronous processes such as ‘pause’ and ‘cancel’ and these are sufficient for controlling open-ended compute jobs. Support for these operations is currently being approved by the OGC for inclusion into the WPS specifications.

4.6 Conclusion

Using the grid to process spatial data in near real-time is possible for soft real-time applications but two challenges must be overcome before this approach becomes viable. Firstly, a mechanism for consistently scheduling real-time jobs within an acceptable time-frame must be devised. A potential solution to this problem would be to search a number of grid information services for under utilised compute clusters to submit the processing job to. Alternatively the JSDL language could be extended with an attribute acting as a real-time identifier, thus real-time jobs could be given priority at certain participating grid clusters. In future pay-per-usage grids these clusters could offer cheap processing for non-urgent jobs which could be paused to make way for real-time jobs.

The second challenge is balancing the trade-off between efficient grid resource utilisation and processing latency. Because high throughput grid clusters are designed to process static computationally intensive jobs their scheduling policy is to allocate one job per machine. As can be seen in the case of real-time processing, much of the time the job is running is spent idle and thus several jobs could potentially run on the same processor and a better resource utilisation could be achieved. Alternatively, for data streams that are too computationally intensive to be processed in near real-time by a single processor, data-stream partitioning could be used to allocate different segments of the data stream to different processors.

Performing continuous near real-time processing of spatial data streams is problematic in a distributed service environment because IO bottlenecks occur on spatial data retrieval. Replicating spatial data sources on the cloud presents a possible solution that requires further research.

Chapter 5 Fine-Grained Snapshot Geoprocessing

5.1 Introduction

In this Chapter an exemplar implementation of the FGSG operation as described in Section 3.5.2 is presented. The key characteristic of FGSG is that it requires an entire dataset as input and as such the geoprocessing is most efficiently carried out close to the data to minimise costly network data transfers. The geoprocessing scenario described in this Chapter involves road traffic routing in which real-time FCD is used to weight the cost of travelling on road segments. Furthermore, this real-time data is augmented with traffic travel-time data collected by NCC for the city of Newcastle upon Tyne. This additional dataset serves two purposes; firstly it provides a base set of travel times to use where no real-time data is available, and secondly it provides a means to validate the accuracy of travel times derived from real-time data.

The concept of using FCD to provide real-time travel-time data for use in routing applications has already proven successful (Wang *et al.*, 2008b). The novelty of the work presented in this Chapter is that it incorporates OGC SWE services to provide and filter the real-time FCD and uses a NGS hosted Oracle grid database service to store the road network data, pre-process the FCD and perform routing functions. In conformance with distributed design principles the spatial data and geoprocessing functionality in this system are to be accessed through a set of web services. As such, a web-based map interface to this system is made available that enables end-users to perform shortest path routing queries based on real-time travel time information derived from FCD sensors.

Although a number of studies have previously attempted to make travel-time predictions from real-time FCD combined with travel-time data (Miwa *et al.*, 2008, Lee *et al.*, 2009), this presents a number of difficulties; traffic flow parameters are required to make accurate predictions and such studies are typically location specific. Thus a rigorous travel-time prediction is beyond the scope of this study, which concentrates purely on the architectural challenges of performing near real-time, data-centric geoprocessing operations using a distributed grid environment in a framework based on open standards. Instead of predicting future road traffic levels the work outlined in this Chapter is concerned only with presenting current traffic conditions and making a shortest-path routing function available based on these current conditions. Although the prediction of future traffic levels presents an arguably more interesting topic, the complexity of modelling required to carry out prediction in a rigorous manner is beyond the scope of this thesis.

The remainder of this Chapter is set out as follows. Section 5.2 provides a basic review of real-time traffic routing using floating car data. Section 5.3 outlines the design of a distributed traffic monitoring system using geospatial web services and a relational database system; the implementation of this system is described in Section 5.4. Section 5.5 describes how the system was tested for functionality and scalability and presents the results. A discussion of these results is presented in Section 5.6 and concluding remarks in Section 5.7.

5.2 Review of Real-Time Traffic Routing using Floating Car Data

FCD has been identified as a useful source of live traffic data that has a potential application in real-time traffic routing (Liu and Meng, 2008, Schäfer *et al.*, 2002, Wang *et al.*, 2008b). In comparison to fixed traffic sensors FCD is capable of providing a robust overview of current road traffic conditions at significantly less cost (Lahrmann, 2007). FCD can be used to help motorists avoid congestion and thus has a clear application for the general public as a route-planning tool but is also of particular benefit to emergency services Private companies have been quick to see the benefits of FCD; commercial systems such as the Tom-Tom XL-HD One (Section 2.2.2) source FCD from mobile phone and satellite navigation system users and feed this data back into a real-time traffic repository that can then be accessed through in-car satellite navigation systems.

Road traffic monitoring systems that use FCD are reliant on a significant amount of hardware and communication infrastructure. Such systems are comprised of numerous in-vehicle sensors and a central data repository; as such they are inherently distributed. The major barrier to the widespread use of FCD to obtain accurate road-traffic information is that a significant proportion of vehicles are required to contribute to data collection. For urban areas Cheu *et al* (2002) suggest that 4% to 5% of total vehicles are required to achieve accuracies of 5 kmh^{-1}, 95% of the time, while Huber *et al* (1997) state that 1% to 5% of vehicles are required depending on the level of accuracy required. However, a significantly lower proportion of vehicles, 0.24% (Brackstone *et al.*, 2001) are required for freeway travel time estimation because traffic streams do not suffer interference from traffic control and because there is no interplay between traffic streams from opposing directions (van Lint, 2004). As a result of this constraint the majority of pilot studies that have trialled the use of FCD augment their data with in-situ traffic loop sensors.

Liu and Meng (2008) implemented a system in Shenzhen, China that used 4000 taxis as FCD probe vehicles. This system focussed on obtaining accurate travel times for each road segment. GPS observations were combined with the taxi status, i.e. free, waiting or occupied, and the observations were pre-processed to eliminate irrelevant data. Subsequently the observations were transmitted at approximately one minute intervals to an Oracle database. A web mapping client was developed using Java Server Pages that interacted with the database using ArcSDE and ArcIMS web services. Although not implemented it was suggested that the data could also be delivered to mobile clients using ArcIMS web services to enable access to this data from within vehicles. Wang *et al* (2008b) implemented a similar system using data collected from a small volunteer sample augmented with historically collected in-vehicle FCD. Speed limit data was used to estimate travel-time for road links where no other data was available. Models were used to predict future traffic conditions on each link in the short and long-term based on fuzzy inference systems (Jang and Sun, 1996). Initial results suggest that this approach was capable of determining reasonable routes based on current traffic conditions. However the prediction models relied heavily on domain knowledge and could not be easily transferred to other locations.

5.3 System Design

5.3.1 User Scenario

The purpose of the system set out in this Chapter is to make real-time road traffic information sourced from FCD usefully available to the public. Such information would enable motorists to plan their journeys using the latest available information and would also enable organisations responsible for traffic management to rapidly identify regions of congestion. City council vehicle fleets are particularly useful sources of FCD because they are deployed city-wide and are easy to manage as they fall within a single ownership domain. The system presented in this Chapter relies on map-matched data from a fleet of city council vehicles equipped with on-board GPS receivers. In this respect the work augments that of the previous Chapter which set out a grid-based map matching system.

A filtering mechanism is to be used in this system to poll the repository of live map-matched observations and to detect road change events which occur when a vehicle moves from one road to another. The combined information provided by the road identifier and the time at which the road change event takes place provides enough information to deduce the travel time of each vehicle on each road segment. This data can then be fed into a data repository and be used to weight road segments based on their average travel time. Subsequently routing applications can use this information to find the quickest route to their destination based on the most up to date traffic information.

The major design goal of this work is to provide a scalable system that exemplifies FGSG on a distributed computing architecture in an open standards-based framework. The extent to which this system can scale is the key metric on which the performance of this system will be evaluated; scalability is to be measured in terms of the number of vehicles that can be supported.

5.3.2 Software and Tool Selection

As in the previous Chapter, Ordnance Survey MasterMap® ITN is to be used as the base road network dataset (Section 4.3.3). To evaluate the applicability of

SWE components to real-time FGSG operations a SES (Section 2.3.6) has been chosen as a notification broker. SES is a proposed OGC standard that builds on OASIS WSN specifications (Section 2.3.1) to provide publish / subscribe access to sensor data observations. A prototypical implementation available from 52 North is to be used which builds on a variety of open source components such as the Apache Muse framework (http://ws.apache.org/muse/), the Esper CEP engine (http://esper.codehaus.org/) and the Apache XMLBeans Java XML binding tool (http://xmlbeans.apache.org/). The SES is to be deployed in an Apache Tomcat servlet container. Experimentation with automatic web service code generation tools such as Apache Axis2 (http://axis.apache.org/axis2/java/core/tools/index.html) and the Metro stack (http://jax-ws.java.net/) failed to parse the SES WSDL document and so a custom binding was found to be necessary.

To insert the SES notifications into a database they must be captured, parsed and translated into a SQL *insert* query. Thus a notification consumer service is to be developed that uses the J2EE JAX-WS 2.0 and SOAP with Attachments API for Java (SAAJ) APIs (https://jax-ws.dev.java.net/; https://saaj.dev.java.net/) to build the service interface and parse the SES SOAP messages. Furthermore, Oracle (http://www.oracle.com) JDBC drivers are to be used to interface with the Oracle relational DBMS at the back-end. The notification consumer service is to be deployed in a Glassfish (https://glassfish.dev.java.net/) container, which was selected because it is both open-source and J2EE compliant.

An Oracle database service hosted by the UK NGS is to be utilised to store and process the FCD data. The NGS Oracle service is comprised of a cluster of eight nodes running Oracle Spatial 11g spread over two sites; each node has a dual CPU 3.06Ghz processor and 4GB RAM. A 2TB Storage Area Network is attached to each site and the nodes are physically connected via a fast Myrinet interconnect. Oracle Real Application Clusters (RAC) is used to federate the nodes using a parallel shared-disk architecture (Section 2.4.2), thus providing significantly greater performance and scalability than a single instance. Furthermore, using RAC each node has direct access to the cache on each other node which facilitates high-availability and the execution of queries in parallel (Greenwald *et al.*, 2008). Triggers and stored procedures to manage incoming data are to be coded in Oracle's procedural query language PL/SQL (http://www.oracle.com/technology/tech/pl_sql/index.html).

To facilitate access to the road network data, the real-time travel time data and the associated geoprocessing functionality such as shortest path routing a number of client facing web services are also to be developed. Road network data will be published as a WFS and WMS through Geoserver (http://geoserver.org). To enable web mapping clients to access a visual representation of the road network data in a timely fashion a Web Map Tile Service (WMTS) (Maso *et al.*, 2010) will be made available via Geowebcache (http://geowebcache.org). WMTS are designed to serve cached tile images of map data at a variety of styles and zoom levels to avoid image processing bottlenecks at the server.

Shortest path routing will be made available through a JAX-WS web service that returns an ordered list of road links given the network node identifiers for the route's start and end locations. Another JAX-WS service is to be created that requires a point location as a coordinate pair and returns the closest network node to this location. This service is designed to help end users graphically select valid start and end locations using a mapping client.

To demonstrate the back-end services described above, a web mapping client is also to be developed that enables end users to view the road network data and to perform shortest path routing queries on this data. The mapping client is to be developed in Adobe Flex (http://www.adobe.com/products/flex/) and Actionscript (http://www.adobe.com/devnet/actionscript/) using the OpenScales (http://openscales.org/) open source mapping API. Flex and ActionScript are proprietary technologies but have become rapidly adopted in web mapping applications as they provide a rich end user experience that enables content to be downloaded asynchronously without the need to reload browser pages (Fraternali *et al.*, 2010). The draft HTML5 standard (Hickson and Hyatt, 2008) is likely to supersede existing rich internet application technologies but to date no standard mapping APIs have been developed for it.

5.3.3 Architectural Overview

The proposed system can be logically divided into three main parts; data input, geoprocessing and user interface. The data input sub-system is responsible for filtering map-matched FCD observations from a fleet of vehicles and inserting the results into a database. The geoprocessing sub-system is responsible for organising this data and augmenting it with static travel-time data obtained from NCC and for providing traffic routing functionality based on this static and real-time travel time data. Finally, the user interface sub-system is to enable clients to visualise current traffic conditions and find the quickest route to their destination using a web mapping client. Figure 5.1 shows a UML2 communication diagram that outlines the message flow and the basic associations between components in the system. A full UML sequence diagram is given in Figure 5.2 and a component diagram is given in Figure 5.3.

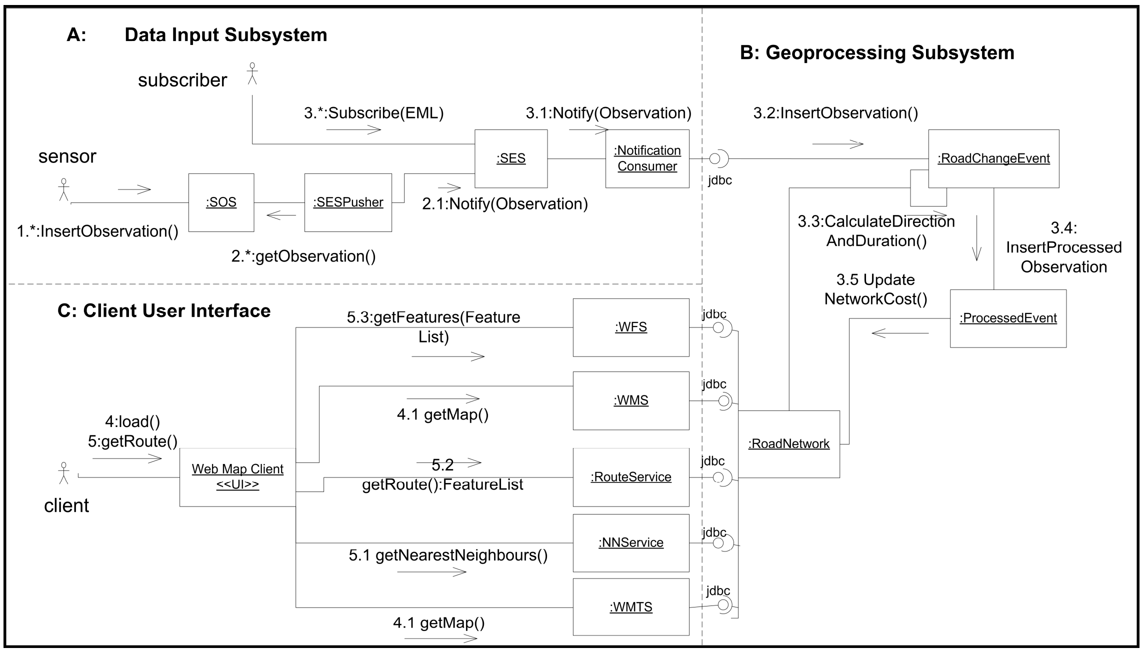
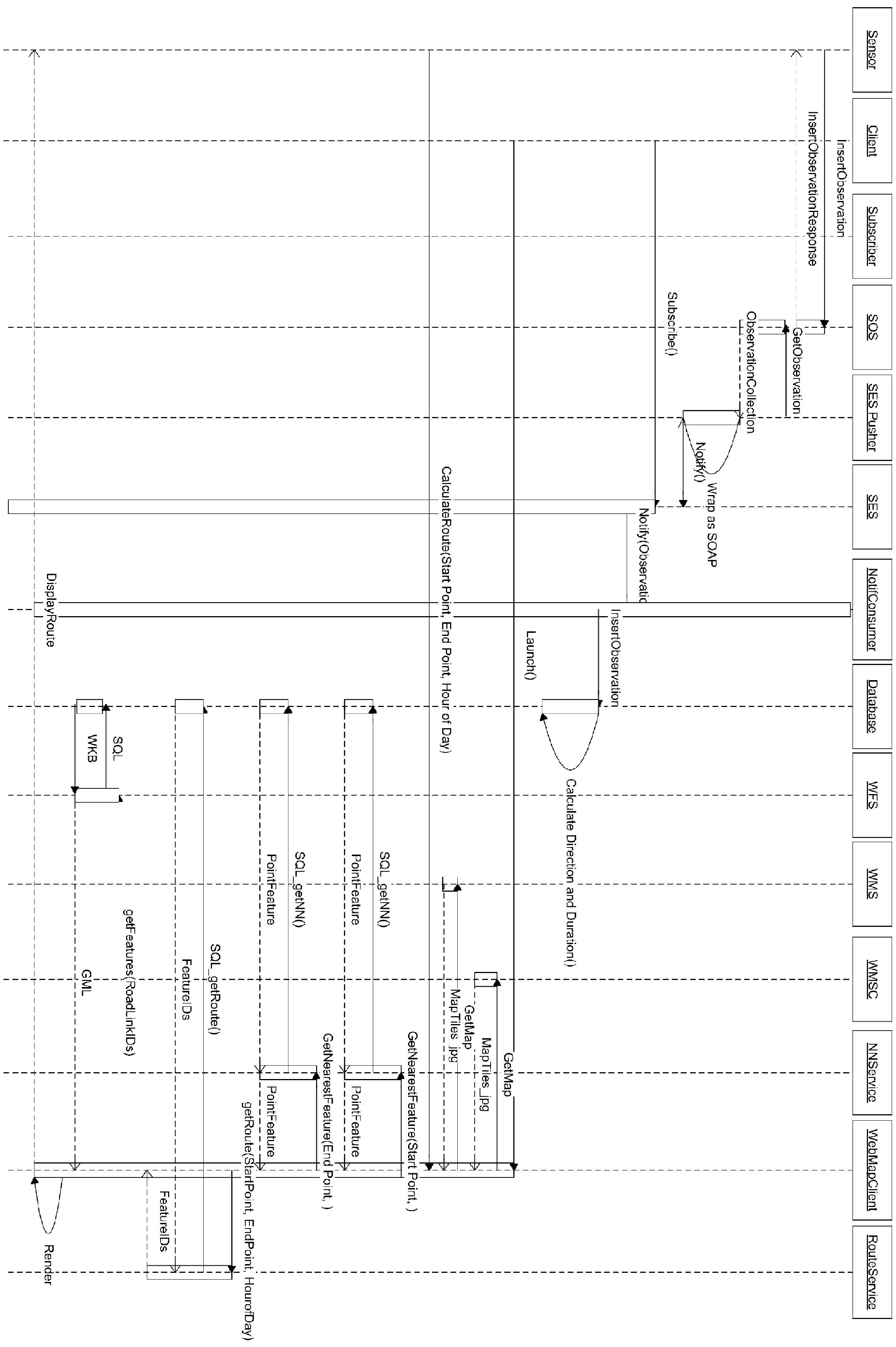


Figure 5.1: A UML2 communication diagram outlining message flow and basic association between system components


Sensor
Client
Subscriber
SOS
SES Pusher
SES
NotifConsumer
Database
WFS
WMS
WMSC
NNService
WebMapClient
RouteService
InsertObservation
InsertObservationResponse
GetObservation
ObservationCollection
Subscribe()
Notify()
Wrap as SOAP
Notify(Observatio
InsertObservation
Launch()
Calculate Direction and Duration()
CalculateRoute(Start Point, End Point, Hour of Day)
DisplayRoute
GetMap
MapTiles_jpg
GetMap
MapTiles_jpg
GetNearestFeature(Start Point,)
SQL_getNN()
PointFeature
PointFeature
GetNearestFeature(End Point,)
SQL_getNN()
PointFeature
PointFeature
getRoute(StartPoint, EndPoint, HourofDay)
SQL_getRoute()
FeatureIDs
FeatureIDs
getFeatures(RoadLinkIDs)
SQL
WKB
GML
Render

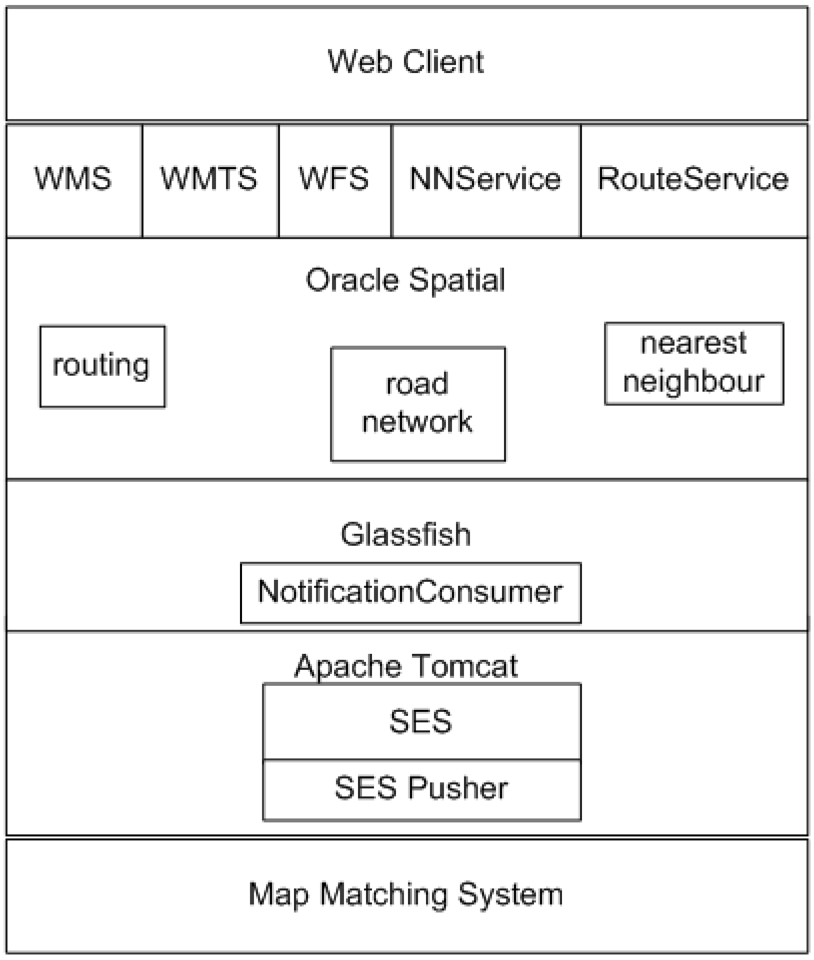


Figure 5.3 Component Diagram of Road Traffic Monitoring System

5.4 Implementation

5.4.1 Data Preparation and Loading

The MasterMap ITN road link and road node datasets were loaded into Oracle Spatial 11g using the SQL Plus and SQL Loader tools. The full procedure for loading these datasets is detailed in Appendix E. Two tables were generated; *ROAD_LINK_POLYLINE* and *ROAD_NODE_POINT* which contained the road link and road node features respectively. Subsequently a spatial network was generated from this road network data comprising a link table *LINK_TABLE*, a node table *NODE_TABLE*, an empty path table *PATH_TABLE* and a path link table *PATH_LINK_TABLE* used to store the link sequence for each path. For each link in *ROAD_LINK_POLYLINE* two road links were created in the spatial network, one representative of each direction of travel. These tables were transformed into a spatial network by inserting a network definition into the user table *USER_SDO_NETWORK_METADATA*. The procedure used to generate the spatial network is outlined in Appendix F.

A spatially referenced travel-time dataset was obtained from NCC that contains the average speed of travel in kmh^{-1} for each direction on major roads in Newcastle upon Tyne, for each hour of the day from 04:00h until 24:00h. This travel-time dataset was loaded into Oracle and joined to the road network link table as columns (*HR_4*,…,*HR_24*). Subsequently the average speed of each road (kmh^{-1}) was converted to average travel-time (s) by dividing the length of each road by its average speed and performing a unit conversion. Twenty additional columns, (*C_4*,…,*C_24*) were appended to *LINK_TABLE* to store the average travel time for each road at each hour of the day. The purpose of performing this join and conversion is to enable shortest path routing calculations to exploit the NCC provided average speed dataset to weight each road segment by travel-time. Appendix F details the process of loading, joining and converting the average travel speed dataset. Figure 5.3 details the resulting database schema after performing the data loading and manipulation procedures described here.

Unfortunately the travel-time dataset provided for this work was incomplete and although it contained travel times for most major roads in Newcastle upon Tyne, only 19% of road segments within the study area had travel-times attached. In order to ensure that an entire road travel-time dataset was available for analysis a spatial interpolation was performed; values were interpolated for each missing travel-time of each road link. However, in order to avoid spatial extrapolation the missing values were only calculated for those features that fell within the bounding box of the features attributed with travel-time values. To this end a new feature table *STUDY_AREA_CLIPPED* was created containing a copy of *LINK_TABLE* but clipped to contain only the features described above, and attributed with additional columns (*T_4*,…,*T_24* and *HRN_4*,…,*HRN_24*) to contain the interpolated speed and travel-time values. The resulting *STUDY_AREA_CLIPPED* table contained 50248 road links of which 9598 contained measured travel times and of which the remaining travel times were interpolated. Appendix F details the SQL commands to create and populate the *STUDY_AREA_CLIPPED* table.

An Inverse Distance Weighted (IDW) interpolation was performed using five nearest neighbours using the formula given in Equation 3. IDW was selected for its simplicity, it is an interpolation method used to provide an estimate *F* of a variable *Z* at an un-sampled location *r*. This is achieved by taking a weighted average of *m* values from the surrounding neighbourhood where the inverse of the distance from the un-sampled location *r* to each of the surrounding points *i,j* is used to weight their respective contribution (Mitas and Mitasova, 1999). In this implementation the distance between neighbouring road segments is calculated using the shortest Euclidean distance. It is noted that the use of network distance would be a more rigorous approach but considerably more computationally expensive (Wang and Kockelman, 2009). Furthermore, existing studies have shown that using Euclidean distance yields satisfactory results for reasonably small networks (Hoef *et al.*, 2006, Kruvoruchko and Gribov, 2004). Interpolation was performed for each travel time column inside the Oracle Spatial database using a PL/SQL procedure that is detailed in Appendix G.

$$F(r) = \sum_{t=1}^{m} W_i Z(r_i) = \frac{\sum_{i=1}^{m} z(r_i) / |r - r_i|^2}{\sum_{j=1}^{m} 1 / |r - r_j|^2}$$

Equation 3

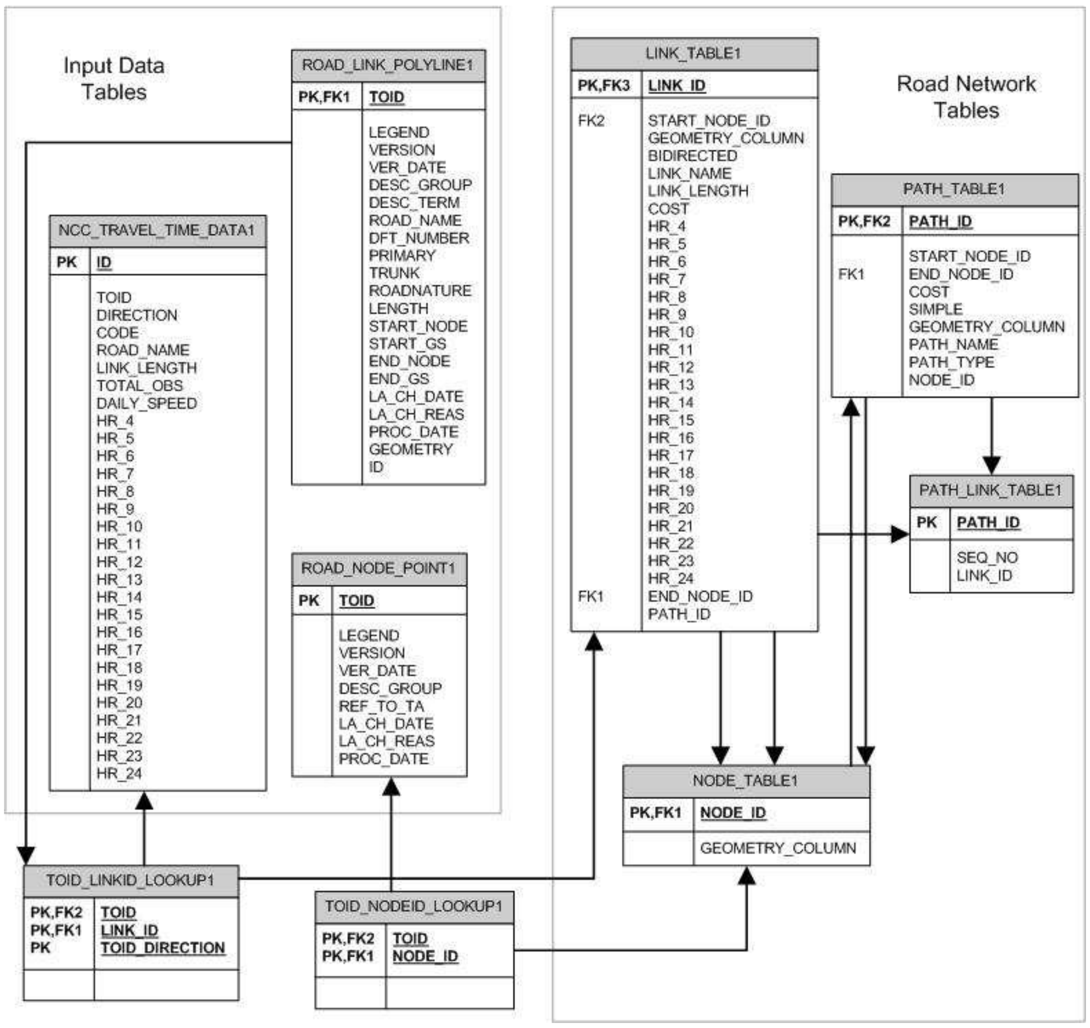


Figure 5.4: Database schema showing the spatial road network tables and the input data tables used to generate the road network

5.4.2 Data Input Subsystem

A simple Java command-line program was developed to poll the SOS for new map matched observations, to wrap these observations in a SOAP envelope and to forward them to the SES for filtering. Figure 5.1 shows that this component provides connectivity between the SES and the SOS.

On invocation the program polls the SOS at a regular time interval using a *DescribeSensor* request to retrieve an O&M *ObservationCollection*. This *ObservationCollection* is parsed into a series of single *Observations* because the current 52 North SES version cannot yet handle whole *ObservationCollections*. Finally, each *Observation* is wrapped in a SOAP envelope as a WSN Notification and sent to the SES via HTTP. Listing 5.1 shows an example WSN Notification containing a map-matched *Observation*.

Listing 5.1: Example WSN Notification produced by SES pusher and sent to SES

```
<env:Envelope  xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:om="http://opengis.net/om/1.0"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<env:Header>
    <wsa:To>http://localhost:8762/ses-main-3.0-
      SNAPSHOT_2010_05_07/services/SesPortType</wsa:To>
    <wsa:Action>http://docs.oasis-open.org/wsn/bw-
      2/NotificationConsumer/Notify</wsa:Action>
    <wsa:MessageID>1259772321843</wsa:MessageID>
    <wsa:From>
      <wsa:Address>http://www.w3.org/2005/08/addressing/role/anony
      mous</wsa:Address>
    </wsa:From>
</env:Header>
  <env:Body>
    <Notify xmlns="http://docs.oasis-open.org/wsn/b-2">
      <NotificationMessage>
        <Topic xmlns:sestopic="http://www.opengis.net/ses/topics"
               Dialect="http://docs.oasis-open.org/wsn/t-
            1/TopicExpression/Simple">sestopic:Measurements
      </Topic>
       <Message>
          <om:Observation gml:id="co_1837"
          xsi:schemaLocation="http://www.opengis.net/om/1.0
          http://schemas.opengis.net/om/1.0.0/om.xsd
          http://www.opengis.net/sampling/1.0
          http://schemas.opengis.net/sampling/1.0.0/sampling.xsd"
          xmlns:om="http://www.opengis.net/om/1.0"
          xmlns:gml="http://www.opengis.net/gml"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:sa="http://www.opengis.net/sampling/1.0"
          xmlns:xlink="http://www.w3.org/1999/xlink">

              </gml:TimeInstant>
             </om:samplingTime>
            <om:procedure xlink:href="ses5"/>
            <om:observedProperty xlink:href=
                  "urn:ogc:def:phenomenon:OGC:1.0.30
            :roadID"/>
            <om:featureOfInterest>
              <sa:SamplingPoint gml:id="position2">
```

```
<om:samplingTime>
<gml:TimeInstantxsi:type="gml:TimeInstantType">
<gml:timePosition>2010-04-13T14:00:08.000+01:00
      </gml:timePosition>

<gml:name>position2</gml:name>
                  <sa:sampledFeature
                        xlink:href="urn:ogc:roadFeature"/>
                        <sa:position>
                  <gml:Point>
                    <gml:pos srsName="urn:ogc:def:crs:EPSG:4326">
                      54.95528   -1.66898
                    </gml:pos>
                  </gml:Point>
                </sa:position>
              </sa:SamplingPoint>
            </om:featureOfInterest>
            <om:result>4000000008044075</om:result>
          </om:Observation>
        </Message>
      </NotificationMessage>
    </Notify>
  </env:Body>
</env:Envelope>
```

A 52 North implementation of the SES was used in this system to filter the map-matched observations and to notify a consumer whenever a road change event is detected. The premise of this system is that every time a vehicle moves onto a new road the SES will detect the change and emit a notification so that the real-time travel-time cost of the previous road can be updated. The 52 North SES parses complex spatial filters encoded in EML and uses an Esper CEP engine to perform the pattern matching. In WSN terms the SES is an OASIS compliant Notification Broker so input and output messages conform to the WS-BrokeredNotification specifications; this WSN functionality is provided through the Apache Muse framework. Both input and output observations are encoded as a WSN Notification. Filters are applied to the SES by sending a WSN Subscribe message containing the filter encoding.

An EML filter was created to identify road change events, i.e. to perform a match every time a vehicle moves from one road, A to the next road B. This was achieved by creating a set of simple, complex and timer pattern filters that collectively identify road change events. Initially a simple pattern, *every* *observation*, was created that matches each observation received by the SES. Subsequent filters *odd* and *even* match alternately; i.e. one matches every odd observation and the other matches every even observation. This enables the comparison of observation events that occur consecutively. The next two patterns *road change 1* and *road change 2* detect adjacent *even* and *odd* events for which the result value, the road's TOID are not equal; an outcome that signifies that a road change event has occurred. Although we are only interested in adjacent *odd* and *even* events the EML syntax necessitates the use of the *BEFORE* operator, hence there are two road change patterns, one that matches *odd* before *even* and the other that matches *even* before *odd*. Two final patterns remain that are used to ensure that the resulting output notification contains the correct information, i.e. the same information that was contained in the input notification. These patterns respectively match when *road change1* and *road change 2* match and output the initial input observation.

The resulting notification contains the unique identifier of the vehicle, the unique identifier of road A and the timestamp of the road change event. The EML filter and associated Subscribe envelope are detailed in Appendix H. This subscription is sent to an SES instance deployed in an Apache Tomcat container.

In order to record the aggregate traffic conditions at a centralised location it is necessary to consolidate the notifications resulting from the SES instances and insert them into a spatial database. Thus a Notification Consumer service was developed to receive road change event notifications from the SES, parse them and insert them into the central database. The JAX-WS API was utilised to develop a one-way service, i.e. a service that does not send a response but simply carries out some business logic when it receives a request. In this case the service listens for notifications from the SES and uses the SAAJ API to extract the required information from the *Notification* document. Subsequently the extracted information such as the observation, road and vehicle identifiers as well as the event timestamp are added to a prepared SQL statement and inserted into the central database. Oracle JDBC drivers were used to provide connectivity to the Oracle instance hosted by the UK NGS. The service was deployed in the Glassfish container.

5.4.3 Geoprocessing subsystem

Within the Oracle database a trigger was developed in PL/SQL to derive higher level information such as the direction of travel, the duration and the cost of travel from the raw notification. The trigger calculates these derived attributes every time a new observation is inserted into the *ROAD_CHANGE_EVENT* table. Initially it is important to determine the direction of travel of the vehicle so that the resulting cost can be applied to the road link representing the correct direction of travel. The direction of travel is calculated by comparing the identity of the node that connects road A and road B. If the connecting node is labelled as road A's end node in the network table then the direction of travel along road A is forwards. Conversely if the connecting node is labelled as road A's start node then the direction of travel is backwards. Duration is calculated by subtracting the timestamp of the previous road change event for the vehicle that submitted the observation from the current event timestamp. Travel time cost is calculated by converting this duration from hours, minutes and seconds into seconds. Finally each of the derived attributes and the original raw observation are inserted into a new table *PROCESSED_EVENTS*. Using a new table to store all the information results in redundancy as each observation is duplicated. However, this is necessary as it overcomes an Oracle constraint that prevents a trigger procedure from updating the table that it is operating on.

Once the vehicle's direction of travel along the road link and the duration it has spent on the road link has been established the next phase involves updating the network cost column. To maximise flexibility it was decided that end users should be able to perform routing queries based on either real-time data or on the static travel speed datasets recorded by NCC. Therefore a further set of spatial networks were defined on the *STUDY_AREA_CLIPPED* table by inserting new entries into the *USER_SDO_NETWORK_METADATA* table for each cost column (*C_4*,…,*C_24*) as well as a new cost column named *COST* to store the real-time data, for which the trigger is responsible for keeping up to date.

To obtain useful and valid routes from the spatial network weighted by real-time cost it is necessary to account for the road links for which no real-time information is available. It is unlikely that at any given time the limited set of available probe vehicles will collectively traverse each road segment in the study area. Therefore it was opted to augment the real-time *COST* column with the interpolated travel time dataset for the current time of day, when no real-time information is available. An additional table *COST_SET_TO(id,hour)* was defined with a single tuple to store the hour of day the *COST* column is currently set to. Furthermore, an additional column, *N_RT_OBS* was appended to the network table *STUDY_AREA_CLIPPED* to store the number of real-time observations that have been captured for each road segment. Once derived duration and direction attributes have been calculated the trigger then attempts to update the real-time *COST* column; this is done as follows.

If *COST_SET_TO* is equal to the current hour of day then only one tuple is updated in *STUDY_AREA_CLIPPED*, i.e. the road-segment for which new travel-time information is available. If this is the first real-time observation for this tuple then the cost is simply set to the calculated duration and *N_RT_OBS* is incremented. Otherwise the existing *COST* value is averaged with the calculated duration.

However, if *COST_SET_TO* is not equal to the current hour of day then each of the tuples must be updated to reflect the current time of day; this will occur for the first observation to be inserted in each calendar hour. Subsequently, for each road link that appears in *PROCESSED_EVENTS* with a timestamp within the past hour *N_RT_OBS* is updated to reflect the number of observations in *PROCESSED_EVENTS* and *COST* is calculated by averaging the calculated durations for these observations. Finally *SET_COST_TO* is updated to indicate that the *COST* column values for which no real-time information is available are set to the current hour. The part of this trigger responsible for updating the

COST column is listed as pseudo-code in Listing 5.2. The entire PL/SQL trigger procedure is detailed in Appendix I.

Listing 5.2: Trigger procedure to update real-time *COST* column

```
IF COST_SET_TO != CURRENT_HOUR {
      UPDATE STUDY_AREA_CLIPPED SET COST = HRN(CURRENT HOUR);
      UPDATE STUDY_AREA_CLIPPED SET COST_SET_TO = CURRENT_HOUR;
      UPDATE STUDY_AREA_CLIPPED SET N_RT_OBS = 0;

FOR EACH RECORD IN PROCESSED_EVENTS{
NO_OBS = number of observations in processed events that relate to
the  same road_segment as this record

UPDATE  STUDY_AREA_CLIPPED  SET  N_RT_OBS  =  NO_OBS  WHERE  TOID  =
RECORD.TOID and TOID_DIR = RECORD.TOID_DIR;

Sum_COST = the sum of RECORD.DURATION for each of NO_OBS;

UPDATE STUDY_AREA_CLIPPED SET COST = sum_COST / NO_OBS WHERE TOID
= RECORD.TOID AND TOID_DIR = RECORD.TOID_DIR;
}

}ELSE{

IF STUDY_AREA_CLIPPED.N_RT_OBS ==0 THEN {
UPDATE  STUDY_AREA_CLIPPED  SET  COST  =  duration  WHERE  TOID  =  toid
AND TOID_DIR = toid_dir;
UPDATE  STUDY_AREA_CLIPPED  SET  N_RT_OBS=  1  WHERE  TOID  =  toid  AND
TOID_DIR=toid_dir;
}ELSE{
UPDATE   STUDY_AREA_CLIPPED   SET   COST   =   ((COST*N_RT_OBS)   +
duration)/N_RT_OBS + 1 WHERE TOID = toid AND TOID_DIR = toid_dir;
UPDATE STUDY_AREA_CLIPPED SET N_RT_OBS= N_RT_OBS + 1 WHERE TOID =
toid AND TOID_DIR=toid_dir;
}
```

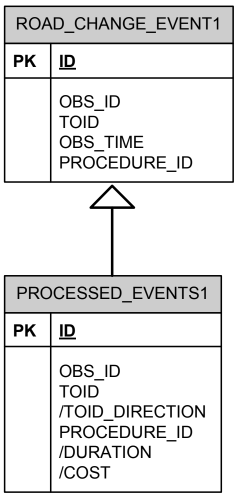


Figure 5.5: UML class diagram showing the parent child relationship between ROAD_CHANGE_EVENT and PROCESSED_EVENTS tables

Shortest-path routing is carried out using the Oracle Network Analysis Load on Demand (LOD) Java API. LOD is a recent feature in Oracle Spatial that enables the analysis of networks that are too large to fit into memory. In order to use LOD the network must be partitioned into segments that fit into memory. Oracle contains a built in function to partition network geometry tables automatically; it performs a spatial partition by recursively bisecting the geometry tables until each partition is of the desired size (Oracle Wang and Gong, 2009). Performance can also be further improved by representing each partition as a BLOB which is defined as a very large data object whose value is composed of unstructured binary data (Shapiro and Miller, 1999). The full procedure to generate and partition the spatial network is detailed in Appendix E.

Network analysis is managed through the *LODNetworkManager* class in the Java LOD API. Initially the *getCachedNetworkIO* method must be called to obtain a handle on the network reader, subsequently analysis can be carried out using a *NetworkAnalyst* object. Analysis is achieved by loading the relevant network partition tables from the database into the user's private session memory as a set of Java objects; each partition is loaded as and when it is needed (Oracle Wang and Gong, 2009, Kothuri *et al.*, 2007). The network tables are locked for editing during analysis to prevent corruption. Data is loaded into private session memory inside the Oracle instance from one of the partitioned network tables at a time on the request of a remote user connected via JDBC.

In this implementation, shortest path network analysis was carried out from a web mapping client described in the following Section. The Dijkstra shortest path algorithm (Dijkstra, 1959) was used to find the least cost route from source to destination by analysing the relative cost of traversing each link using the relevant cost column to weight each link, either *HRN_4*,…,*HRN_24* for static queries or *COST* for real-time queries.

5.4.4 Client User Interface

A JAX-WS web service was developed to access the routing functionality described in Section 5.4.3. The service accepts three string arguments containing the node identifier of the start node and the end node and the cost column on which to base analysis. It returns an ordered list of link identifiers representing the shortest route from the start node to the end node using the selected cost column. Internally the Oracle network analysis LOD API is used to query the Oracle database via JDBC. The service is deployed in a Glassfish v2 container.

Another JAX-WS web service was developed to assist users in finding the nearest node to a particular geographical location. This service accepts an OSGB36 coordinate pair and returns the identifier of the nearest node. Internally this service queries the backend Oracle database by sending a prepared SQL statement via JDBC. The nearest neighbour is identified by using the built in SDO_NN function in Oracle spatial. The prepared statement used to extract this information is given in Listing 5.3. The service was deployed in a Glassfish v2 container.

Listing 5.3: SQL prepared statement to identify nearest neighbour to OSGB36 coordinates <easting><northing>

```
SELECT n.node_id AS RESULT from node_table n where
sdo_nn(n.geometry_column,sdo_geometry(2001,27700,SDO_POINT_TYPE(
<easting>,<northing>,null),null,null),'sdo_num_res=1')='TRUE'");
```

A Style Layer Descriptor document was created for each road network cost column to display categorised views of the road network data based on travel speed using four categories; 0-30 kmh^{-1}, 30-50 kmh^{-1},50-70 kmh^{-1} and >70 kmh^{-1} coloured green, light yellow, dark yellow and red in this order. The Oracle Geoserver plugin was installed to enable Geoserver to use the Oracle road network as a data source and styled map layers were created for each of the cost columns. A Geowebcache instance deployed in a Tomcat container was used to cache each of these layers at 15 zoom levels.

A WFS was deployed using Geoserver to serve road network features in vector format. Using this approach web mapping clients using the Route Service and the Nearest Neighbour Service can request specific features such as road links and nodes from the WFS and display them on a map.

A user interface component was developed as a front end web page to enable end users to visualise the average travel speed on the road network at different times of the day, and to visualise shortest path routes between locations within the study area. A screenshot of the client is shown in Figure 5.6. Open Street Map (http://www.openstreetmap.org/) data is used to display base mapping data, a blue polygon represents the extent of the study area and the WMTS speed layer is shown as an overlay. There are two major user interaction components. On the left hand side of the screen is a navigation panel that enables the user to navigate the map using zoom and pan controls. This panel also enables users to turn each layer on and off, or change the transparency of each layer. On the right hand side of the screen is the routing options control panel. Within this panel is a dropdown menu that enables the user to select the time of day they want to travel. Selecting a new time of day causes the WMTS travel speed overlay to update, and changes the cost column that is used to perform any routing calculations. Alternatively the user can check the ‘use real-time’ box which causes routing analysis to use the real-time cost column to weight the cost of travel on each road segment. Checking this box also changes the WMS speed overlay to only show the road links for which real-time observations are available, all other road links are displayed in grey, although routing analysis will weight these segments with their cost value for the current time of day.

To calculate a route the user has to click on the ‘select start location’ button, this creates a marker on the map and invokes the Nearest Neighbour service which finds the closest road node to this point. This road node is then requested from the WFS and displayed on the map in a different style; the same procedure is carried out to select the end node. Clicking on ‘get route’ invokes the Route service which returns the shortest route between the start and end nodes as an array of road identifiers. This string array is translated into a WFS request which returns the corresponding road features and these are displayed on the map. Clicking on reset removes all the markers and routes from the map.

The user interface was developed in Flex 4 and Actionscript 3 using the OpenScales API. It was packaged as a web application and deployed in a Tomcat container.

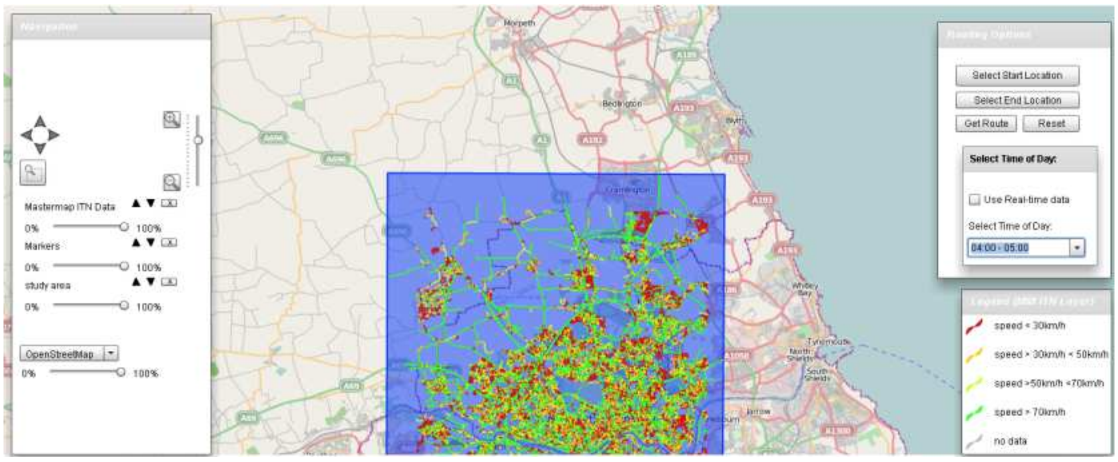


Figure 5.6: Screenshot of the user interface component

5.5 Testing & Results

A thorough testing and verification process was undertaken to ensure that the system as a whole functioned correctly. Initially each component was individually tested to ensure that no logical errors existed in the code and that the interfaces were correctly defined. Subsequently the system was tested in its entirety to ensure that the whole web service workflow ran smoothly and performed the functions required of it; namely to transform a collection of vehicle GPS observation tracks into road network travel-time weightings. A series of load tests were also performed on the system to determine its maximum capacity in terms of number of vehicle sensors and volume of client traffic.

5.5.1 Amazon Machine Image (AMI) Configuration

A collection of 304 vehicle GPS observation tracks were obtained from NCC. The data was sourced from GPS data loggers fitted onboard a heterogeneous fleet of council maintenance vehicles travelling around Newcastle-upon-Tyne. Observations were recorded throughout the day at one minute intervals on 21st September 2010. Stop-start vehicles such as refuse disposal wagons were excluded as their speed of travel does not give an accurate indication of road traffic conditions.

The track data for all vehicles was provided in a single comma delimited text file with geographical coordinates encoded using the ETRS89 reference frame. The data was transformed and loaded into a PostGIS spatial database using a PL/pgSQL script. The loading procedure incorporated steps to remove corrupted observations, transform the coordinates into the WGS84 reference frame and divide the data into a set of tables, each containing a time-ordered sequence of observations from a single vehicle. The procedure is documented in Appendix K.

Given the extent of the scalability issues previously identified with the 52N implementation of the SOS and the Geoserver WFS (Section 4.4.2) it was clear that numerous data input server instances would be required to sufficiently strain the geoprocessing subsystem. Consequently it was decided to deploy the data-input subsystem in the Amazon cloud using EC2 (http://aws.amazon.com). EC2 is Amazon's IaaS product; EC2 virtual machines can be dynamically provisioned on-demand for a relatively low cost.

An Amazon Machine Image (AMI) was built on top of a 64-bit Windows Server 2008 operating system. Each of the web services in the data input subsystem was installed on the AMI; the SOS, SES and WFS. Additionally, the sensor emulator (Section 4.3.2), the SES Pusher (Section 5.4.2), the map-matching program (Section 4.3.4) and the PostGIS database were installed. It was opted not to install the Notification Consumer service on the AMI because this requires JDBC connectivity to the NGS Oracle server which only allows connections from a set of pre-approved IP addresses. Although EC2 instances do retain their IP address until they are terminated it was opted to host the Notification Consumer service off the cloud at a real physical host with a static IP to avoid constant renegotiations with the NGS.

Based on the findings of the previous Chapter (Section 4.4.2) it was decided to run a single Geoserver WFS instance and a single 52N SOS on each EC2 node, and to allocate 10 sensors to each node. SES deployment proved to be problematic because each sensor in this system requires its own SES instance in order to handle stream based observation filtering. It was found that only one SES instance could be deployed per Tomcat container and thus 10 Tomcat containers had to be installed on the AMI, each using a different set of ports. A Java program was written to bulk create a set of 304 virtual sensors (Section 4.3.2) and to register each of these sensors to the SOS. The set of virtual sensors were divided into groups of 10 and placed in different directories, one for each AMI instance. Finally, a Java program was used to create a set of batch scripts, one for each sensor group, that invokes the data input workflow chain. For each of the sensors in the group the sensor emulator program is invoked which streams observations from the database into the SOS. Subsequently the map-matching program is initiated which performs matching by retrieving a road network subset from the WFS, after which the SES pusher is initiated that pushes map-matched observations from the SOS into one of the SES instances.

Once the AMI was configured, EC2 nodes could be launched through the Amazon web management console. It was found that an ‘m1.large’ hardware configuration was required for each instance; this includes 7.5GB of memory and 5 EC2 Compute Units (ECU). ECU is a metric used by Amazon to quantify the compute capacity provided to EC2 instances; 1 ECU roughly corresponds to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor (http://aws.amazon.com).

5.5.2 Estimation of the Probable Route between Non-Adjacent Network Links

This system was designed to process a dense data stream containing an observation every one or two seconds. Consequently the assumption was made at the system design stage that each road change event would occur between two adjacent road links; i.e. links that share a start node or an end node. However, only GPS observation streams of one minute frequency were available from NCC and initial trials with this data showed that a very small proportion of road change events occurred between adjacent road links. The majority of events occurred between road links that were close but not adjacent to each other. Thus it was deemed necessary to estimate the route travelled between each pair of road links in order to correctly determine the actual road links travelled on, the direction of travel of the vehicle and the length of time taken to traverse each road link.

The trigger procedure set out in Listing 5.2 was amended to estimate the path between non-adjacent road links in a road-change event. Each time a road change event occurs between non-adjacent links A and B a shortest path computation is performed between the two links, based on physical distance rather than interpolated travel-time cost. The assumption is made here that the route taken by each vehicle between A and B will be the route with the shortest network distance, thus road length is used to weight each link rather than travel time cost. In order to carry out this localised shortest path computation in an efficient manner a new road network *TEMP_ NETWORK* is defined on two new tables, *TEMP_NODES*, *TEMP_LINKS* containing only the road links and nodes within a 2Km radius of road A. Unless the vehicle is travelling at over 120Kmh^{-1} then the second road link will fall within this radius. Similarly, in all but the most unusual road network topologies the shortest path between the two nodes will also lie within this radius. Once the path taken between the two road links has been established, the time difference between the two observations is divided amongst each of the links in the path, weighted by link length. In this way an estimated cost is assigned to each of the links travelled on. The revised trigger procedure is listed in Appendix I.

There are two major advantages to using this sub-network approach; firstly, to perform a shortest path computation in Oracle it is necessary to read the entire road network into private session memory. Using the PL/SQL API this is impossible for the Newcastle road network as it is too large, although it can be achieved using the Java load on demand API by reading different parts of the network as and when they are required. The second advantage to this approach is that the length of time taken to perform a shortest path computation is significantly reduced as there are considerably fewer links to scan. Performance tests were carried out to compare the relative performance of route calculation on both the mini-network and the full-network using 8 randomly selected routes of various lengths. The mini-network approach was found to outperform the full-network on average by a magnitude of 7.

As road links A and B are not adjacent the direction of travel along each link cannot be determined by simply comparing the identity of their start and end nodes (Section 5.4.3). The road network is directed so the shortest path computation selects the correctly directed links between road A and road B. However, in order to select the correct start node and end node for the shortest path computation it is necessary to perform two comparisons to obtain the path that travels along road A but not road B. Figure 5.7 shows the four possible paths between road A and road B. Path 1 represents the correct configuration as it includes the road links travelled on between T0 and T1, time instants representing the previous and current road change events respectively. To find the correct start and end nodes, two comparisons must be performed. The start node corresponds to the starting node of the longest path, in number of links, between either of road B's nodes and each of road A's nodes. Conversely the end node corresponds to the ending node of the shortest path between either of road A's nodes and each of road B's nodes. As travel time costs are calculated by dividing the time difference between T0 and T1 over each of the road links that lie between road A and road B it is important that this is calculated correctly.

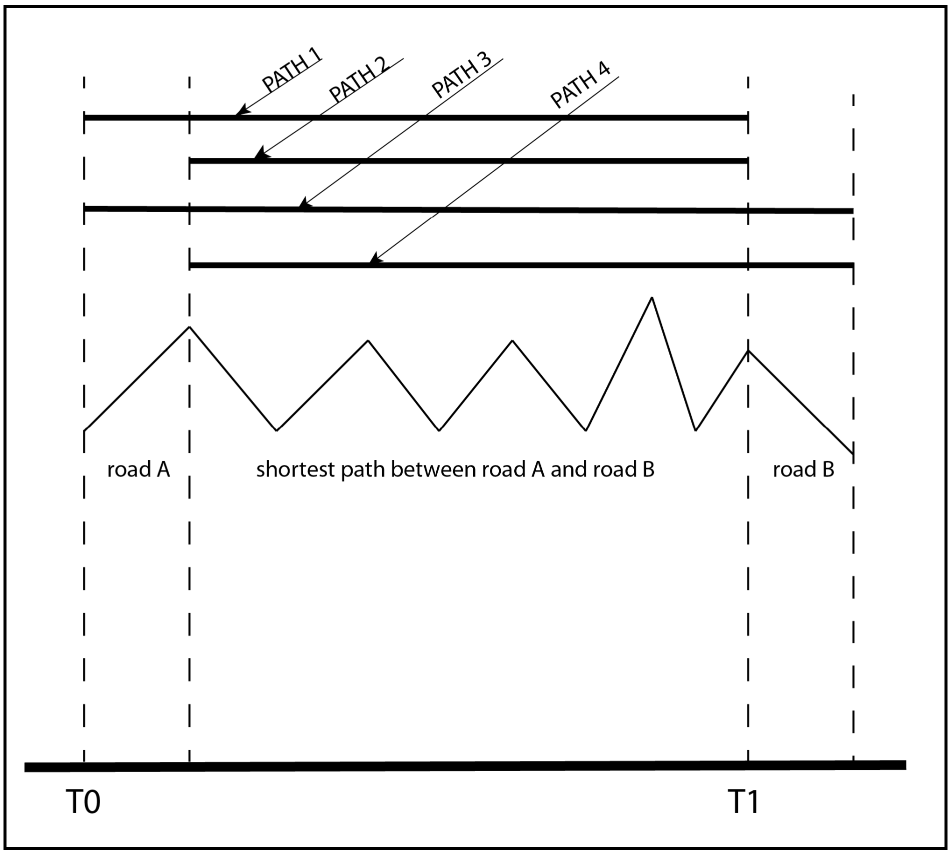


Figure 5.7: Diagram showing the four possible path configurations between road A and road B. Path 1 shows the correct configuration as it represents the distance travelled between road change event at T0 and road change event at T1

5.5.3 Functional Testing for a Single Vehicle

A functional test was carried out to confirm that the system assigns the correct cost weighting to each road link; initially only one observation stream from a single vehicle was analysed. A map is provided in Figure 5.8 that shows the first part of the route travelled by this vehicle which begins its journey on the West of the map and heads in an Easterly direction. GPS observations are depicted as grey squares on the map and their corresponding map-matched road links are depicted as broad dashed black lines and labelled with their identifying TOID. This observation stream is typical of the data acquired from NCC; position is observed only at 60 second intervals and thus it can be seen that no two map-matched roads are adjacent. There are also a small percentage of observations which have not been map-matched due to discrepancies between the position and bearing of the vehicle and those of the surrounding road network.

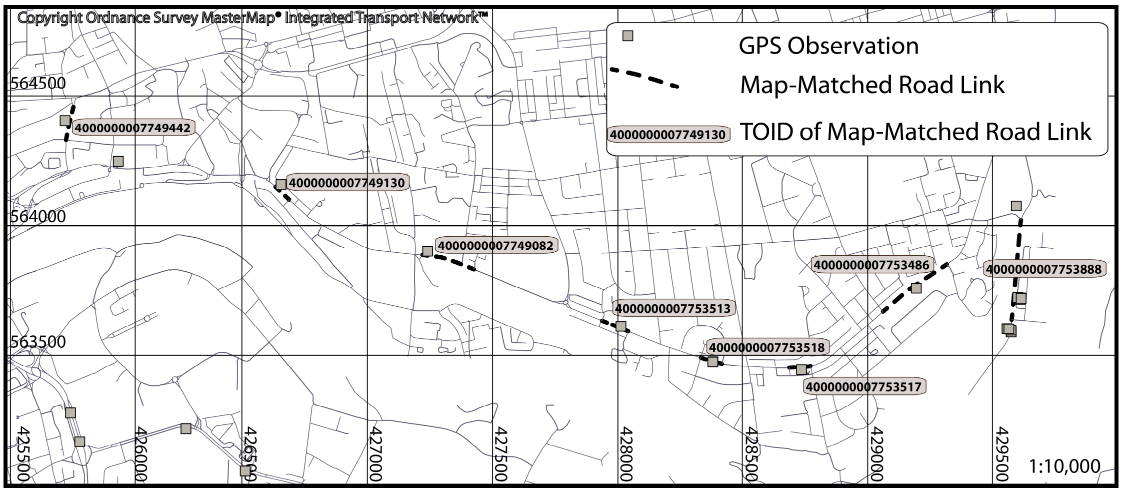


Figure 5.8: Sample vehicle route showing GPS observations and map-matched road links

To verify that the system works correctly the road link travel times for the above route were calculated manually, and then compared to the travel times produced by the system. Table 5.1 lists the raw observations, the map-matched roads and the associated road change events as identified by the SES for the route depicted in Figure 5.8. The duration column denotes the total travel-time cost between road change events. The data shows that the vehicle makes a number of stops at road links 4000000007753817, 4000000007753888, 4000000007753513 and 400000000774913. These stops explain why road change events do not occur at each and every minute interval. Unfortunately there is no mechanism in this system to account for stationary parked vehicles and so these observations will cause an over estimation in travel-time for the road links in question.

Table 5.1: Raw Observations and their corresponding Road Change Events

Raw Observation		Map-matched road	Road Change Event	
Position (OSGB36 WKT)	Time hh:mm:ss	TOID	Time hh:mm:ss	Duration ss
POINT(429528.02668635 563592.364480371)"	07:57:00	4000000007753817		
"POINT(429528.02668635 563592.364480371)"	07:58:00	4000000007753817		
"POINT(429544.97030135 563579.677950211)"	07:59:00	4000000007753817		
"POINT(429544.071589793 563586.535008899)"	08:00:00	4000000007753817		
"POINT(429535.503444385 563591.30072277)"	08:01:00	4000000007753817		
"POINT(429583.836682928 563705.871664461)"	08:02:00	4000000007753888	08:02:04	
"POINT(429581.839862056 563701.221271016)"	08:03:00	4000000007753888		
"POINT(429571.446909098 563707.645304683)"	08:04:00	4000000007753888		
"POINT(429585.017783863 563704.766581186)"	08:05:00	4000000007753486	08:05:04	180
"POINT(429566.335565731 564046.846085716)"	08:06	4000000007753517	08:06:04	60
"POINT(429164.125716246 563743.193043373)"	08:07	4000000007753518	08:07:05	61
"POINT(428702.622021928 563441.609577328)"	08:08	4000000007753513	08:08:05	60
"POINT(428347.396768455 563471.443775008)"	08:09	4000000007753513		
"POINT(428346.764698489 563470.141096373)"	08:10	4000000007749082	08:10:12	127
"POINT(427978.09568244 563603.415897156)"	08:11	4000000007749130	08:11:12	60
"POINT(427202.759552821 563882.99326178)"	08:12	null		
"POINT(426614.32461212 564130.234670907)"	08:13	4000000007749442	08:13:12	120

Figure 5.9 shows the road links that were assigned a new cost value after insertion of the road change events into the database. It can be seen that the route estimated between each GPS observation on the basis of shortest network distance appears to be valid. The calculation of travel-time cost weighting for these road links is given in Table 5.2 for the first four road change events. The first column in Table 5.2 shows the identifier of the previous and the new road links, road A and road B, in addition to the duration between this and the previous road change event. The second and third columns show the identifier and length of each road link travelled upon between road A and road B. In the fourth column the expected proportion of time spent travelling on the road link is calculated by dividing the length of the road link by the total length of road travelled between road A and road B. The fifth column shows the calculated travel-time cost, derived by multiplying the distance proportion by the duration. Finally the sixth column shows the results produced by the system, it can be seen that in each case the correct result was produced.

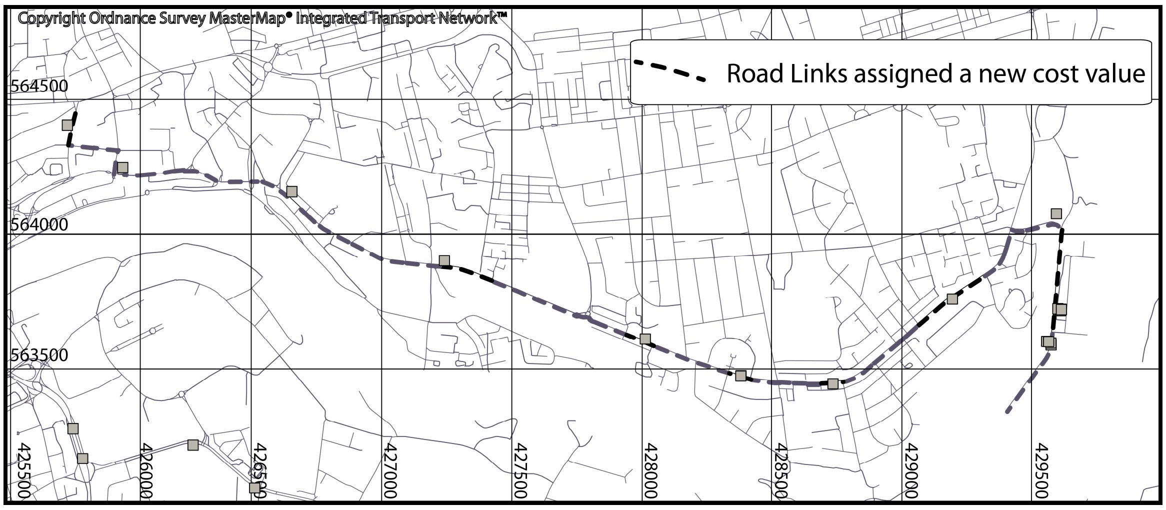

Copyright Ordnance Survey MasterMap® Integrated Transport Network™
Road Links assigned a new cost value
564500
564000
563500
425500
426000
426500
427000
427500
428000
428500
429000
429500

Figure 5.9: Map showing the road links assigned a new cost value by the system

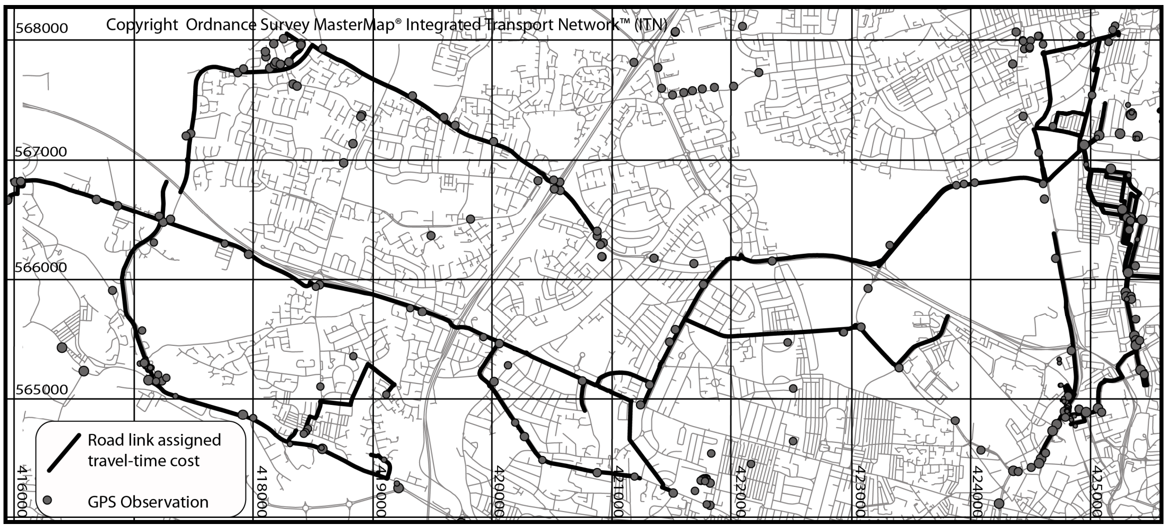
Table 5.2: Road link Cost Calculation from Road Change Events

Road Change Event	Road links between Road A and B	Link Length (m)	Proportion of Distance between A and B	Calculated Cost Duration * proportion	Assigned Cost
EXPLANATION	The road links comprising the shortest path between road A and road B	Length of the road link	(link length/ total link length) * 100	(link length / total link length)* duration	The cost weighting assigned to the road link by the system
Road A: 4000000007753888	4000000007753888	390.9	48.26%	86.869	86.87
	4000000007753889	147.59	18.22%	32.799	32.79
Road B:	4000000008046200	60.34	7.45%	13.41	13.41
4000000007753486	4000000008041716	73.37	9.06%	16.31	16.31
	4000000007753552	26.57	3.28%	5.90	5.90
Duration:	4000000007753534	37.27	4.60%	8.28	8.28
180 seconds	4000000007753520	73.93	9.13%	16.43	16.43
Total:		809.97	100%	180	180
Road A:	4000000007753486	318.27	46.44%	27.87	27.87
4000000007753486	4000000007753533	55.31	8.07%	4.84	4.84
Road B:	4000000007753551	26.24	3.83%	2.30	2.30
4000000007753517	4000000007753550	31.24	4.56%	2.74	2.74
Duration:	4000000007753519	111.66	16.29%	9.77	9.78
60 seconds	4000000007753496	142.58	20.81%	12.48	12.48
Total:		685.30	100%	60	60
Road A:	4000000007753517	86.34	23.91%	14.59	14.58
4000000007753517	4000000007753516	56	15.51%	9.46	9.46
Road B:	4000000007753530	51	14.12%	8.62	8.62
4000000007753518	4000000007753515	60.13	16.65%	10.17	10.16
Duration:	4000000007753514	107.63	29.81%	18.18	18.18
61 seconds Total:		361.1	100%	61	61

5.5.4 Functional Testing for Multiple Vehicles

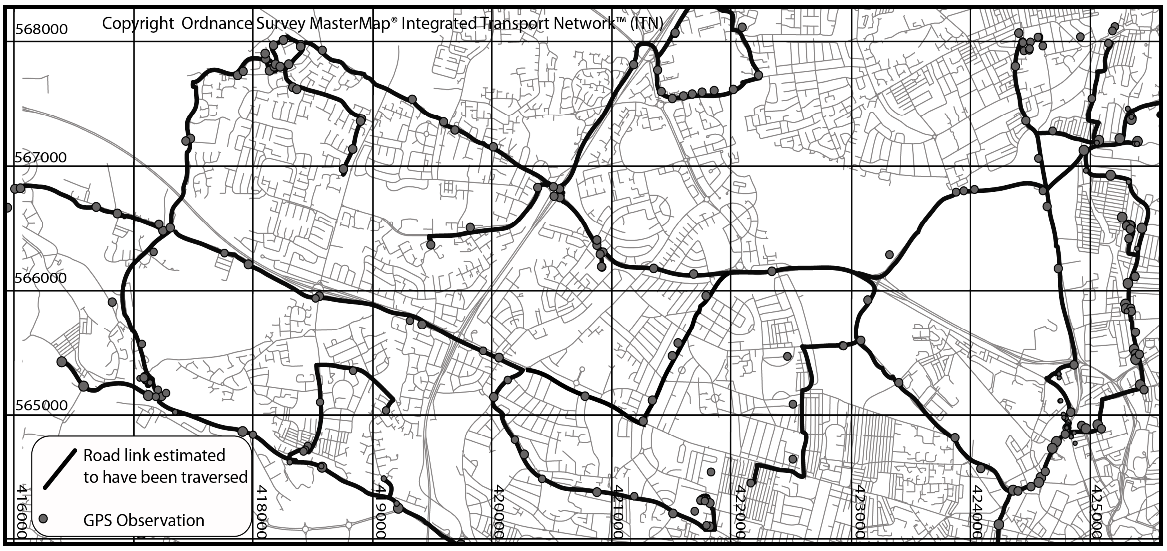
To ensure that the system is capable of estimating travel-time for more than one vehicle another test was performed using a sample of 10 vehicles. The GPS tracks from these vehicles intersect with each other in both time and space; each of the tracks was recorded between 08:00 and 09:00 on 21st September 2010. In order to gauge the systems performance the state of the road link cost column was captured every 15 minutes during this one hour period. The performance of the system can be gauged on two levels; assignment accuracy and travel-time cost accuracy. Respectively these measures refer to the system's ability to calculate travel-time cost for the correct set of road links, and the accuracy of the resulting travel-time costs. Unfortunately the true path taken by each vehicle is unknown, as is the true travel-time cost of each road link. However, by visually comparing the raw GPS observations with the set of road links that have been assigned travel-time costs, it is possible to make a reasonable assessment of assignment accuracy. Likewise, the interpolated travel-time data acquired from NCC for the 08:00 to 09:00 time period provides a best estimate of the actual travel-time on these road links and thus by comparing the observed travel-time values with the interpolated values it is possible to assess the travel-time cost accuracy.

Figure 5.10 displays the road links that had been assigned cost values after running the system for one hour, alongside the raw GPS observations fed into the system during this period. On inspection, it can be seen that a significant proportion of road links have not been assigned a travel-time cost despite an obvious path of GPS observations. The clearest example of this in Figure 5.10 is the cluster of observations between grid reference 421000 567500 and 422200 568500. It was found that in the majority of cases these observations without a corresponding road link occurred during the latter part of the hour period. By removing the clause in the trigger procedure to reset the real-time cost column every hour it was found that these road links were eventually assigned a cost value. Thus it can be concluded that latency in the data input subsystem is largely responsible for this shortfall in assignment accuracy; this is discussed further in Section 5.5.5. There are also some cases for which roads have been assigned a travel-time cost where no vehicles have travelled upon them. There are two obvious examples of this in Figure 5.10 at 420800 565000 and at 420900 565200. The most likely explanation for this is that the road links were map-matched incorrectly. Figure 5.11 shows the road links that should have been assigned a travel-time cost; these links have been derived from visual analysis of the raw GPS observations.



Copyright Ordnance Survey MasterMap® Integrated Transport Network™ (ITN)
568000
567000
566000
565000
416000
418000
419000
420000
421000
422000
423000
424000
425000
Road link assigned travel-time cost
GPS Observation

Figure 5.10: Map showing the road links assigned a travel-time cost and the raw GPS observations



Copyright Ordnance Survey MasterMap® Integrated Transport Network™ (ITN)
568000
567000
566000
565000
416000
418000
419000
420000
421000
422000
423000
424000
425000
Road link estimated to have been traversed
GPS Observation

Figure 5.11: Map showing the estimated set of road links that should have been assigned a travel-time cost and the raw GPS observations

To assess the accuracy of the resulting real-time travel-time costs, each real-time cost value was compared to the interpolated travel-time cost value for the same road link and the same time period of 08:00 to 09:00. For each 15 minute period the mean and standard deviation of the absolute difference between the two travel-time costs were calculated; the results are summarised below in Table 5.3. It can be seen that there is very little change in the accuracy of the real-time system over the hour period.

Table 5.3: Summary statistics for the absolute difference between interpolated and real-time travel-time costs

Sample Time	08:15	08:30	08:45	09:00
No. of Assigned Roadlinks	75	233	317	401
Mean of the absolute difference between interpolated and real-time travel time (seconds)	10.06	9.29	12.96	12.36
Standard Deviation of difference between interpolated and real-time travel-time (seconds)	12.36	11.74	17.30	19.57

Each road link's travel-time cost value is calculated by averaging the travel time of each vehicle that has travelled along it, thus it is anticipated that the accuracy of each link will increase proportionally to the number of vehicles that have travelled on it. Table 5.4 presents the mean absolute difference in travel-time cost again, but broken down by the number of real-time observations for each road link. This clearly shows that as more vehicles travel upon a road link the real-time travel-time cost becomes significantly closer to the interpolated value.

Table 5.4: Mean absolute difference between interpolated and real-time travel-time costs by number of real-time observations

No. of real-time observations	Mean absolute difference in travel-time cost (seconds)			
	08:15	08:30	08:45	09:00
1	8.91	6.87	9.19	9.96
2	1.11	2.42	3.01	2.12
3			0.58	0.08
4			0.17	0.17

5.5.6 Scalability Testing

The system was subjected to an increased data load and its performance was monitored. The load was increased by launching new AMI instances, each of which performs data input for 10 sensors in a shared nothing configuration. As such the expected points of failure in the system are the Notification Consumer and the database which must handle all the database insert transactions. The Notification Consumer was monitored by logging the time delay between each notification and its associated road change event. Within this time period a complex processing chain is executed; the observation is map-matched, inserted into the SOS, pushed to the SES where it is filtered and forwarded to the Notification Consumer. This portion of the processing chain takes 3 to 4 minutes. Subsequently the Notification Consumer waits for a free connection and then inserts the observation into the database. Due to database atomicity constraints (Section 2.3.2) the connection is not released until the database's internal trigger procedure has returned; tests show that the average processing time of each trigger procedure was found to be 45 seconds although this figure varies significantly depending on whether the implicated road links are adjacent. It can be seen from the results in Table 5.5 that the delay between observation and notification increases proportionally to the number of sensors. This suggests that the database insert and associated trigger procedure is the source of a bottleneck.

Table 5.5: Time Delay between Road Change Event and Notification

No. of Sensors	No. of Road Change Events per hour	Average time delay between road change event and notification(mm:ss)
10	67	06:02
20	94	14:40
50	195	18:51

Another aspect of the database design that does not scale well was found to be the exclusive lock required by each vehicle on the temporary network tables used to estimate the vehicle's path between known positions. This is likely to account for a significant portion of the bottleneck as each vehicle's path must be processed sequentially. As a result of this bottleneck another flaw in the system was observed. It was found that as the number of sensors is increased, observations from the same sensor are not necessarily inserted into the database in the same order that their notifications occurred. This is problematic as the trigger procedure relies on observations being inserted in their correct order. Travel-time cost is calculated by comparing the time stamp of the latest observation with that of the previous observation from the same sensor, which is retrieved from the *PROCESSED_EVENTS* table. A clause in the trigger prevents new observations from being inserted if the previous observation from the same sensor has a timestamp that occurs after the new observation, as this would result in a negative cost value. Clearly this is unsatisfactory because as the number of sensors is increased the proportion of inserted travel-time observations is reduced. Solutions to this problem are discussed in Section 5.6.

Although the scalability of this system from the data input side presents an interesting problem the major focus of the work in this Chapter is fine-grained snapshot geoprocessing. In this system, fine-grained snapshot geoprocessing occurs primarily when an end-user invokes a routing query through the web mapping client. In the scenario presented in this Chapter it is anticipated that a high volume of end-users require the use of this real-time routing service. The focus of this Section is determining how well the nearest neighbour and shortest-path web services scale cope with an increased workload, rather than attempting to increase the availability of the client application itself.

For the shortest-path service 100 start and end nodes within the study area were randomly selected; for each set of nodes a SOAP request to the shortest-path service was constructed. Using Apache JMeter a large set of users was simulated using concurrent execution threads; on each thread one of the 100 shortest path requests was randomly selected and sent to the service. For the nearest neighbour service the same procedure was followed, although randomly selected coordinates from within the study area were used in place of start and end nodes. The results are displayed in Table 5.6 and Figure 5.12. Surprisingly the response time for each of these services is similar, despite the greater computational complexity of the shortest-path service. These results are promising and show that the services scale well to a large number of users;

only a 14 second delay is experienced when 500 different requests are made simultaneously.

Table 5.6: Response Time of Shortest Path Routing and Nearest Neighbour Web Services

No. Threads	Shortest Path Response Time	Nearest Neighbour Response Time
1	544	532
10	569	558
20	609	615
50	1476	1265
100	2758	2306
250	7426	6867
500	12731	14113

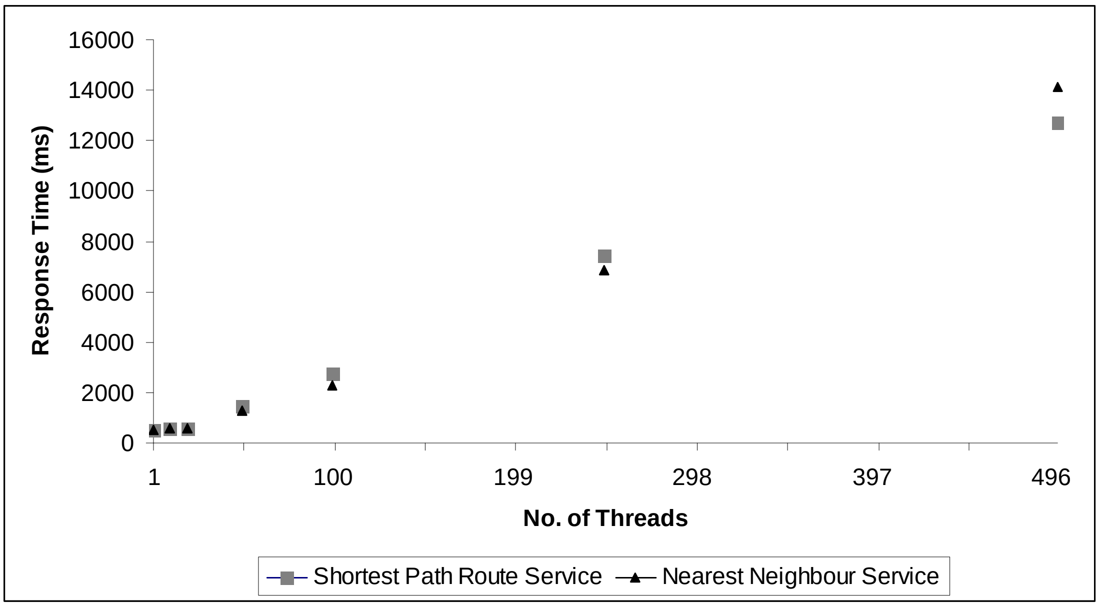


Figure 5.12: Response Time of Shortest Path Routing and Nearest Neighbour Web Services

No attempt was made to test the scalability of the Geoserver WFS and WMS or the GeoWebCache WMTS as these services are not anticipated to present a bottleneck in the system.

5.6 Discussion

In this Chapter a road traffic monitoring system has been designed and implemented that incorporates elements of data stream geoprocessing and fine-grained snapshot geoprocessing. The system is composed of geospatial web services and a parallel relational database hosted by the NGS. Additionally the Amazon EC2 cloud infrastructure has been utilised to deploy the system on a large scale.

This system has successfully demonstrated how real-time geospatial sensor data streams can be filtered and processed using a complex geoprocessing workflow. A variety of open standards have been incorporated into this system including SOS and SES elements from Sensor Web Enablement, WMS, WMTS and WFS elements from OWS and WSN from the OASIS framework. Furthermore SOAP based and RESTful services have been seamlessly combined into a unified workflow. It was found that the SES SOAP bindings and the adherence of the SES interface to WSN specifications presented a useful alternative to the traditional OGC RESTful interface. The key advantage of this interface was that the Notification Consumer service could be easily constructed using the Metro JAX-WS stack (https://jax-ws.dev.java.net/) and a standard WSDL document published by OASIS. This ease of deployment provides a good example of the benefits of creating SOAP bindings for OGC services.

It was observed that integration of the SOS and the SES could be improved. In this system the SES is made aware of new observations in the SOS through a bridge program, the SES pusher. This program polls the SOS every two minutes using a RESTful HTTP request, parses the resulting observations, encodes them as a WSN notification and sends them to the SES. There are a number of problems with this approach. Firstly, this process introduces latency as there may be a delay of up to two minutes before the SES is aware of new observations at the SOS. Secondly, the process of parsing and reformatting the observation document is computationally expensive which increases the use of computational resources and could also introduce a processing bottleneck. Finally, this approach adds an additional layer of communication which contributes to latency and which may also produce a bottleneck if observations are voluminous.

One solution to this problem would be to move the SES forward in the processing chain; raw observations could be fed directly into the SES which would forward these observations to both the SOS and the map-matcher program. The map-matcher program could then output map-matched observations as notifications to another SES which both archives the observations in the SOS and forwards road change events to the Notification Consumer program. Alternatively an integrated service could be developed that publishes both an SOS and SES interface, thus enabling both push and pull access to observations from a single data service. An integrated service would greatly simplify sensor web workflows but is unlikely to be developed unless the SES is approved as a SWE standard.

The use of a parallel spatial relational database to perform the bulk of geoprocessing in this system produced mixed results. Road network data provides a good fit to the relational data model and there is a clear performance advantage to carrying out fine-grained geoprocessing operations such as shortest path and nearest neighbour analysis in close proximity to the physical data store. However, insertion of new observations into the database produced a bottleneck. In part this was due to the system design that appended a block of pre-processing to the insert transaction, thus monopolising database connections for a lengthy period of time. Another contributing factor was that insertions were being made on an individual basis; aggregating a collection of new observations and performing a bulk insert is likely to have reduced the bottleneck although it would have further increased latency.

An interesting question is whether relational databases are a suitable storage medium for real-time sensor observations. The ACID guarantees (Section 2.4.2) of a relational database ensure that data consistency is maintained but at the expense of availability (Lynch and Gilbert, 2002). As a result it is difficult to constantly update a data aggregate stored in a relational database and to query it concurrently because records are locked while they are being updated.

Furthermore, sensor data is notoriously unreliable and systems need be robust to corrupt and erroneous observations, which violate ACID constraints. The new movement of NOSQL databases show promise for storing sensor data as they have more relaxed consistency rules (Leavitt, 2010). However, to date such systems are only capable of the most trivial spatial analysis operations.

The concept of sensor-network databases (Madden, 2002, Govindan *et al.*, 2002) also present an interesting solution to this problem. Sensor-network databases are capable of running analysis on a network of sensors without storing the data in a centralised location; queries are processed in-network. Although these sensor-network database systems have been successfully used for wireless sensor networks (Gaynor *et al.*, 2004), it would be difficult to implement for this traffic monitoring system. Our aggregated dataset is not an aggregation of raw observations; rather it is a derived phenomenon of approximated road link travel times. As such it would not be possible to retrieve this information directly from sensors as a pre-processing chain must first be executed.

Part of the problem with the data input system design was that it relied on observations being inserted into the database in the order that they occurred. It was found that this condition did not hold true as the number of input sensors was increased. A simple solution to this problem would be to add another layer of abstraction to the SES. In its current state the SES emits a road change events to the Notification Consumer containing only the timestamp of the event and the identifier of the current and previous road links. This could be improved by including the timestamp of the previous road change event, thus providing enough information for the trigger to calculate travel-time costs for each road link regardless of the order in which the road change events arrive at the database.

5.7 Conclusion

This system has demonstrated how fine-grained snapshot geoprocessing can be incorporated into an end-to-end monitoring and prediction system. The system did not scale well to a large number of data input sensors due to a bottleneck caused by the insertion of processed observations into the database. Aggregation of observations before insertion into the database presents a potential solution to this problem. NOSQL databases may also present a solution as they do not adhere to the strict consistency rules of traditional relational databases. This technology has not yet reached maturity and has poor support for spatial data, although it does present an interesting topic for further research.

No attempt was made to scale this system over a larger geographical area. It is anticipated that the shortest-path geoprocessing operation would scale well in this regard because it operates on a set of partitioned tables and utilises a load-on-demand approach to processing. However, the problem of scaling over multiple vehicles would have to be solved before this could function as an effective system.

Chapter 6 Coarse-Grained Snapshot Geoprocessing

6.1 Introduction

This Chapter details the design and implementation of a geoprocessing system that executes a CGSG operation in parallel. The operation is an implementation of the Spatial Reclassification Kernel (SPARK) image processing algorithm which has been modified to run on Amazon's EC2 Elastic MapReduce service.

Applying the SPARK algorithm to a classified image has been shown to improve the precision of thematic classification by translating broad land cover classes such as trees or buildings into more specific land use classes such as residential housing or industrial wasteland (Barnsley and Barr, 1996). The SPARK algorithm operates by passing a kernel window over a classified image and comparing the spatial frequency and arrangement of pixels in each kernel window to a set of predefined land use templates.

We have found that the SPARK algorithm is a good fit to the MapReduce programming model and that its execution time can be significantly reduced by applying the presented MapReduce (Section 2.4.3) approach. A major goal of this work is to evaluate the effectiveness of the cloud infrastructure at performing parallel CGSG operations in an efficient and scalable manner.

6.2 Background and Context

6.2.1 Elastic MapReduce

Amazon offers an Elastic MapReduce service (http://aws.amazon.com/ elasticmapreduce/) that enables the elastic deployment of MapReduce jobs on their EC2 infrastructure using Hadoop (http://hadoop.apache.org), a popular open-source java implementation of the MapReduce framework. Elastic MapReduce fits into the PaaS category of distributed system as it encompasses a software framework, Hadoop MapReduce, as well as hardware resources;

EC2 and S3. Amazon S3 (http://aws.amazon.com/s3/) is a set of web services that provide redundant and scalable data storage as a service. MapReduce jobs are elastically deployed onto EC2 instances using S3 as a back-end data resource. As such entire workflows can be executed remotely on the Amazon infrastructure without consuming any local computational or data resources.

Key benefits of MapReduce include the automatic handling of fault-tolerance, load balancing and data distribution, thus it shields the developer from many of the complexities of parallelisation. Hence the framework offers a relatively straightforward way to develop task-farm style distributed applications. However, MapReduce has attracted criticism on the basis that it is inferior to parallel relational DBMS for many applications and has even been described as a ‘major step backwards’ by prominent members of the parallel database community (Dewitt and Stonebraker, 2008b, DeWitt and Stonebraker, 2008a, Stonebraker *et al.*, 2010). The main criticisms levelled at the MapReduce model by Stonebraker *et al* (2010) are summarised here. Firstly, MapReduce does not make use of indexes or columns to rapidly access data items of interest; instead it uses a brute-force approach that requires each data record to be scanned in its entirety. This has the effect of reducing performance for query intensive operations on relationally structured data. Secondly, MapReduce natively operates on text files and thus each record must be parsed before it can be operated on. Conversely parallel DBMS store typed data and so the parsing stage can be omitted in each processing workflow. Thirdly, MapReduce schedules tasks to each worker node at runtime using a fixed data granularity corresponding to the storage block size. This is considerably less efficient than the approach taken by parallel DBMS in which tasks are scheduled and optimised at compile time by means of a distributed query plan. Fourthly, parallel DBMS use streaming to transport data between nodes whereas MapReduce writes intermediate data structures to disk between the *Map* and *Reduce* stages, thus introducing another IO bottleneck into the workflow.

Despite these criticisms, Stonebraker *et al* (2010) concede that MapReduce works well for certain types of operation. Notably, MapReduce excels at Extract, Transform, Load (ETL) operations that extract data from heterogeneous data sources, performs a transformation and loads into a database. Additionally it is considered a useful tool for processing operations on non-structured data. An overview of MapReduce work in the geospatial domain is provided in Section 2.4.3.

The work presented in this Chapter fits into the raster processing category but rather than performing a simple raster algebra operation it attempts to execute an algorithmic workflow by applying a processing kernel to an image and comparing the similarity of resulting kernel windows with a set of predefined land-use templates.

Figure 6.1 outlines a component diagram showing the basic components of Amazon`s Elastic Map Reduce.

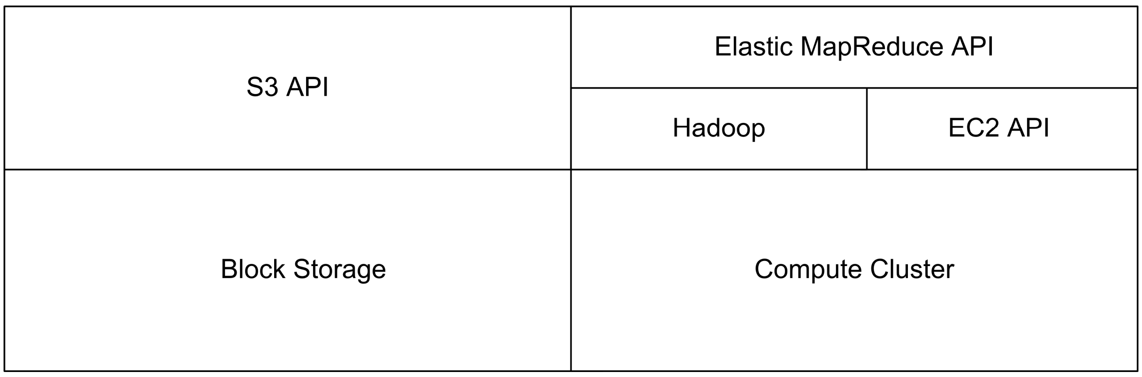


Figure 6.1 Component Diagram of Elastic Map Reduce

6.2.3 The Spatial Reclassification Kernel (SPARK) Algorithm

The remainder of this Chapter describes the implementation of the SPARK algorithm using MapReduce on the Amazon cloud. Originally developed by Barnsley and Barr (1996) this algorithm reclassifies satellite sensed imagery to improve land-use type inference.

Classification algorithms typically do not perform well in urban areas due to the large number of spectrally distinct land-cover types in close proximity to each other. Barnsley and Barr (1996) have presented a reclassification algorithm, SPARK that translates broad spectral land-cover types such as building, trees, water, grass and tarmac into more specific land-use categories such as agricultural, residential and industrial. This is accomplished by passing a kernel over the classified image in which each kernel window is compared to a set of pre-defined land-use templates and the central pixel of each kernel window is reclassified to the most similar land-use type. The similarity between each kernel window and the set of pre-defined land-use templates is determined through the examination of frequency and spatial arrangement of pixels in each window. The SPARK process is described here in 5 steps:

1. Perform an Initial Classification of a Satellite Sensed Image into Land-cover Types.

Classification is the process of identifying the real-world land-cover type of each pixel in a remotely sensed image, and assigning the pixel a new value to indicate this land-cover type. Classification techniques fall into two major categories referred to as supervised and unsupervised classification respectively. Supervised classification requires a-priori knowledge of the study area in the image; training areas in the image exemplar of each land-cover type must be identified manually. The remaining pixels in the image are grouped into one of the land-cover types defined by the training areas based on their spectral similarity. Unsupervised classification algorithms perform a similar process but without manual user intervention; land-cover types are inferred based on the spectral separability of pixels in the image. A variety of methods exist for performing both supervised and unsupervised classification; a full treatment is given in Mather (2004).

2. Define a set of Land-use Templates

Identify the major land-use types in the study-area using a-priori knowledge, and for each of these land-use types select a training area on the classified image that is representative of each land-use type. Define the size of kernel window to be used in the analysis, and then for each land-use class take a template for this window size from a random location within the polygon or set of points that have been recognised as belonging to the given land-use class.

The random location forms the centre pixel of the template window and the adjacency events are derived relative to this random location.

3. Calculate Adjacency Matrices

For every given kernel window and land-use template a corresponding adjacency matrix must be defined. An adjacency matrix is an immutable matrix with width and height equal to the number of land-cover types defined in the classified image. Each adjacency matrix contains the frequency and type of adjacency event that occurs between different land-cover type pixels in a window. Adjacency events refer simply to a pair of contiguous pixels; for example if two pixels classified as grass occur next to each other in a window then it can be said that a grass-grass adjacency event has occurred. Adjacent edges and adjacent vertices in each window are counted in this way but only one adjacency event exists for each pair of pixels, thus two adjacent pixels grass and tree would only result in a single adjacency event grass-tree, not grass-tree and tree-grass. Figure 6.2 shows the adjacency events in a 3x3 kernel window. Adjacency-events are grouped together in an adjacency matrix *M* that stores the frequency f_{ij} with which pixels from land-use *i* and pixels from land-use *j* are adjacent (Equation 4).

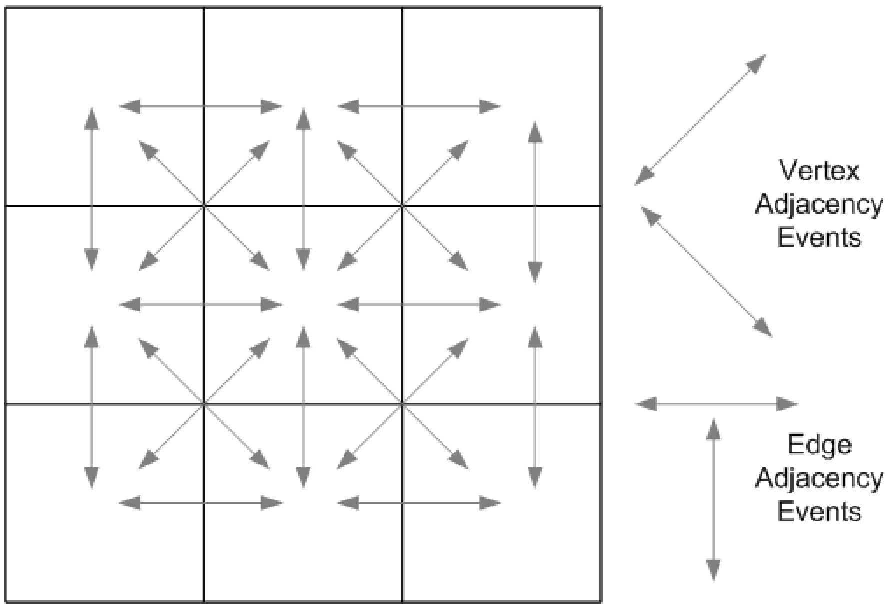


Figure 6.2: Adjacency Events in a 3x3 Kernel Window [adapted from Barnsley and Barr (1996)]

$$M = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix}$$ **Equation 4**

4. Calculate Kernel Window and Template fit

The similarity between a kernel window and a land-use template is measured by comparing their respective adjacency matrices and normalising the result. For a given classified image the similarity between an image adjacency matrix A$_{ij}$ and a template adjacency matrix T$_{ij}$ can be expressed as an index between 0 and 1. This index is referred to as Δk and can be calculated using Equation 5, where *C* is the number of land-use types and *N* is the total number of adjacency events in the window, i.e. 20 for a 3x3 kernel or 72 for a 5x5 kernel. The resulting metric, Δk, represents the degree of coherence between the two windows and ranges from 0 which indicates no similarity to 1 which indicates an exact match.

$$\Delta k = 1 - \sqrt{\frac{1}{2(N^2)} \sum_{i=1}^{C} \sum_{j=1}^{C} \left\{ A_{ij}(f) - T_{ij}(f) \right\}^2}$$ **Equation 5**

5. Perform Reclassification

A new output image is defined to represent the reclassified study-area. Each pre-defined land-use template is mapped to a byte value in the output image; e.g. industrial 0, residential 1, agricultural 2. The location of each pixel in the reclassified output image corresponds to the central pixel of each kernel window on the input image. Thus, for each kernel window the corresponding pixel in the output image is assigned to the byte value of its most similar land-use template; i.e. the template that has the maximum value of Δk for the corresponding kernel window.

6.3 Design & Implementation

The SPARK algorithm is suitable for coarse-grained parallelisation because it involves repetitive computation on independent subsets of the classified image. The significant portion of computation in this algorithm is the calculation of an adjacency matrix and Δk for each kernel window. For example, let us consider a simple processing scenario involving the reclassification of a 3000 x 5000 pixel image using a 3x3 kernel window and 5 land-use templates. Computation of an adjacency matrix and Δk must be performed for every pixel except for those on the outside perimeter of the image due to incomplete kernel window information; this amounts to (3000 x 5000) – 15996 = 14,984,004 pixels. Adjacency matrix construction involves 20 comparison operations for each 3x3 kernel, and comparison with each of the 5 land-use templates involves at least another 10 arithmetic operations. Thus (14,984,004 x 20) + (14,984,004 x 10 x 5) = 1,048,880,280 instructions must be processed even for this simple example. The work presented here will concentrate on parallelising the calculation of adjacency matrices and Δk as set out in steps 3 and 4 in Section 6.2.3 with the aim of reducing the overall processing time. An overview of the workflow described in the following text is outlined in Figure 6.3.

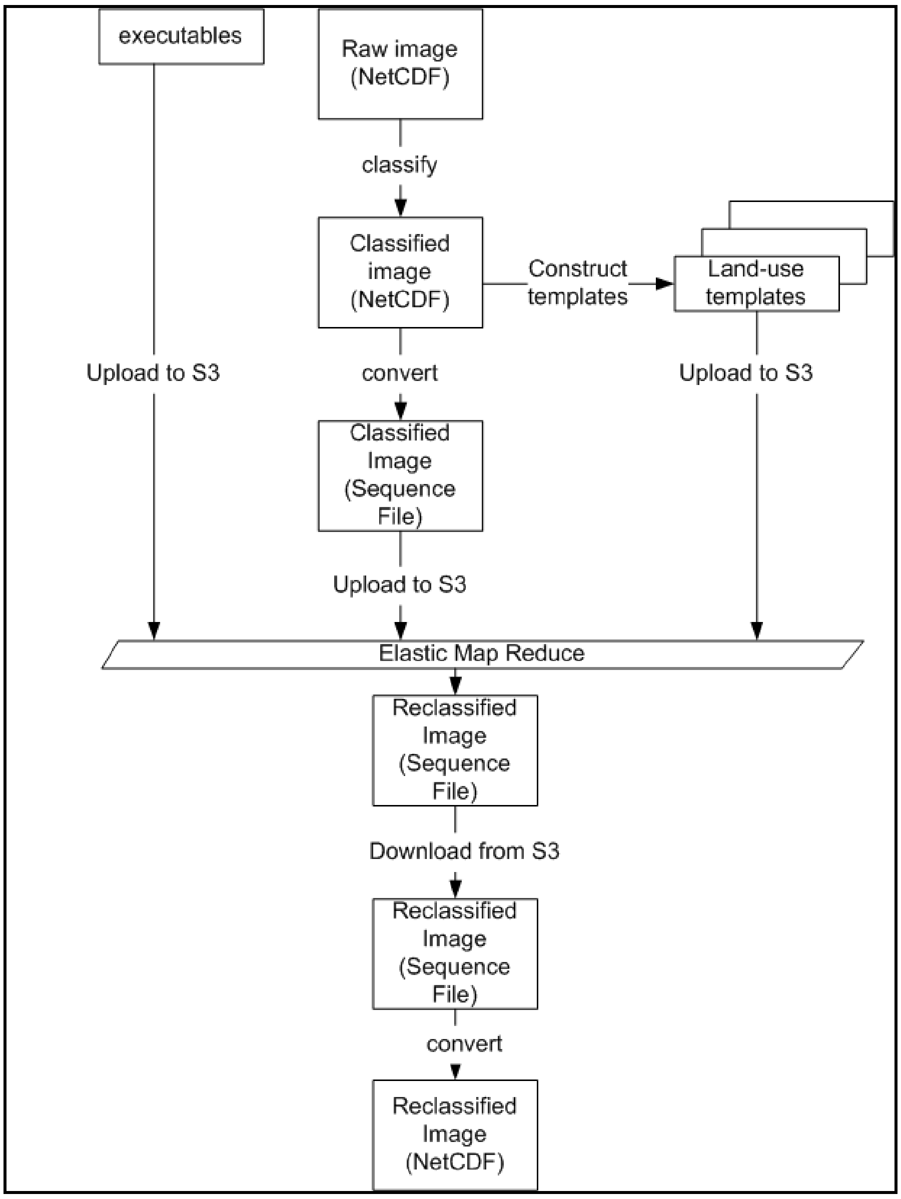


Figure 6.3: Pre and post processing stages in the MapReduce SPARK workflow

6.3.1 Data Partitioning

Hadoop automatically divides input data files into chunks referred to as input splits and allocates these amongst available processors. The size of an input split typically corresponds to the file block size which defaults to 64MB in the Hadoop File System (HDFS) although it is possible to customise both the input split size and the file-block size. Input data can be passed to Hadoop in text or binary format but must be comprised of key-value pairs that are referred to as records. Thus data partitioning involves two major elements; selecting an appropriate input split and file block size, and selecting the type of data item to encode in each record.

The input data for the SPARK algorithm is an image of the study area, classified by land-use. In the first instance it was opted to partition this dataset into kernel windows. For a given kernel window of dimension *n* pixels, every pixel *P* in the image with the exception of those that are less than (*n*-1)/2 pixels from the edge of the image, has a corresponding kernel window *W*. The kernel window *W* is simply a square subset of the image, centred on *P* with a width and height of *n* pixels. A custom java object *Window* was designed to represent each kernel window, comprised of a two-dimensional byte array to store the pixel values, and an *IntPair* field, comprised of two integer values denoting the Cartesian coordinates of the window kernel's central pixel *P* in relation to the image. The *Window* object implements Hadoop's *Writable* interface which enables it to be serialized and deserialized internally by *Map* and *Reduce* functions in Hadoop. The *IntPair* field implements Hadoop's *WritableComparable* interface, an extension to the *Writable* interface that enables the results to be sorted. Using the Unidata NetCDF java library (http://www.unidata.ucar.edu/software/netcdf/) a java method was written to read a classified image in NetCDF format into an array of *Window* objects and to write these objects to a *SequenceFile*, a binary encoded file format specified by Hadoop that can be used as input to a MapReduce job. For each kernel window a record was written to the SequenceFile containing the kernel window's image coordinates as an *IntPair* in the key field and the serialized *Window* object in the value field.

This partitioning method was chosen because it prepares the input data for subsequent processing operations; the calculation of each adjacency matrix and Δk can easily be performed by the *Map* function by simply deserializing each *Window* object. However, this approach to data partitioning was found to be extremely inefficient due to the considerable expansion in file size from the raw image to the SequenceFile. For a 9x9 kernel window the transformation of a 12MB NetCDF file into SequenceFile format resulted in a SequenceFile over 2GB in size which took over 10 hours to upload to S3 from a standard broadband internet connection. Consequently an alternative partitioning strategy was sought; it was decided to write larger blocks of data to each SequenceFile record in an attempt to reduce the amount of data redundancy in the SequenceFile.

A new partitioning method was devised with a lower level of data granularity. This was found to be considerably more efficient in terms of storage volume and upload speed; it is described as follows. For a kernel window dimension of 3 pixels, a classified image of width 5000 pixels is subdivided into two-dimensional blocks of data of 5000 pixels width and 3 pixels height. Each block is serialized into a java object referred to as a *RowSet* containing a two dimensional byte array of 3 x 5000 pixels and a text field indicating the position in the image of the block's central row. A SequenceFile record is written for each *RowSet* object containing the *x* coordinate of the central row as the key, and the serialized *RowSet* as the value. Using this approach for a 12MB NetCDF image and a 9x9 kernel window the generated SequenceFile was reduced to 108MB in size and took only 42 minutes to upload.

The downside of this approach is that each *Map* function has to convert the *RowSet* into an array of *Window* objects and generate an adjacency matrix and Δk for each of these kernel windows, thus reducing the data granularity of the *Map* task. For an image of width *x* pixels and height *y* pixels, Equations 3 and 4 relate the SequenceFile storage volume S_v to the image size and the kernel window dimension *n* for the *Window* method (Equation 6) and the *RowSet* method (Equation 7) given an arbitrary key size of *k*. It can be seen that the *RowSet* method reduces the storage volume of the resulting SequenceFile by a factor of *n*.

$$S_v = n^2 k(x-n+1)(y-n+1) \qquad \textbf{Equation 6}$$

$$S_v = nyk(x-n+1) \qquad \textbf{Equation 7}$$

As a consequence of this more compact file format the amount of computation to be performed in each InputSplit is considerably greater, resulting in a lower data granularity and a reduced ability to exploit as many processors. To compensate for this it was decided to reduce the file block size and input split size from 64MB to 8MB, thus enabling a job with a 108MB input file to be distributed amongst 14 processors rather than 2.

6.3.2 Hadoop Configuration

Hadoop provides a mechanism referred to as a Distributed Cache that makes a small set of auxiliary files available to each MapReduce process. This was used to supply each MapReduce process with a copy of land-use templates with which to compare each *Window*. A further set of *Window* objects were created to store the land-use templates and these were initialised with the template's byte code value in the key field, and a land-cover pixel arrangement exemplar of the template's land-use category in the byte array. An additional mapping file was also distributed to each processor containing a mapping of each land-use type to its corresponding byte code value in the image.

Each land-use template *Window* object was serialized and uploaded to an Amazon S3 storage bucket, as was the SequenceFile and the mapping file. A java archive file containing the SPARK logic encoded as Hadoop Map and Reduce processes (Section 6.3.3) was also uploaded to S3.

6.3.3 The Map and Reduce Functions

A custom map function was written to read each record from the SequenceFile containing *RowSet* key-value pairs into an array of *Window* objects, and to transform these *Window* objects into a different set of key-value pairs as set out in the following steps:

1. Calculate the adjacency matrix for the kernel window
2. Calculate the adjacency matrix for each land-use template
3. Compare the window kernel's adjacency matrix to the adjacency matrix of each land-use template to produce a value for Δk.
4. Emit a new key-value pair for each land-use template using the window kernel's image coordinates as the key and a new key-value pair as the

value, in which the key is the byte value of the land-use type represented by the template and the value is Δk.

A custom reduce function was written to combine records with the same key, and to emit a single key value pair containing the identifier of the window and the byte value identifier of the land-cover template with the greatest value of Δk, i.e. the land-cover that is most similar to the kernel window. The map and reduce processes using the approach described here are detailed in Listing 6.1.

Listing 6.1: SPARK Map and Reduce Functions

$$map(RowSetIdentifier, RowSet) \rightarrow list(WindowIdentifier, (TemplateIdentifier, \Delta k)$$
$$reduce(WindowIdentifier, list(TemplateIdentifier, \Delta k)) \rightarrow list(WindowIdentifier, TemplateIdentifier)$$

6.3.4 Output Conversion

Hadoop produces an output file for each reducer process and because each window's image coordinates are stored as *IntPair* objects which implement *WritableComparable* the results are sorted by image coordinates. A java method was written to combine the output files and generate a new NetCDF file containing the re-classified image.

6.4 Testing & Evaluation

6.4.1 Test Scenario

A multispectral (XS) SPOT-1 HRV image of South East London (4195 x 2995 pixels) was selected to test this system. Using ERDAS Imagine software (http://www.erdas.com/) the image was geometrically corrected to fit the British National Grid using a nearest neighbour resampling method. The image was classified using a Maximum Likelihood supervised classification that identified six land-cover classes; water, grass, crops, forestry, small buildings and large buildings. The classified image was converted into NetCDF format using the GDAL library (http://www.gdal.org/) and the FWTools package (http://fwtools.maptools.org/). The full pre-processing, classification and format conversion procedures are detailed in Appendix J. The confusion matrix for this land-cover classification is shown in Table 6.1, and the classified image is shown in Figure 6.4.

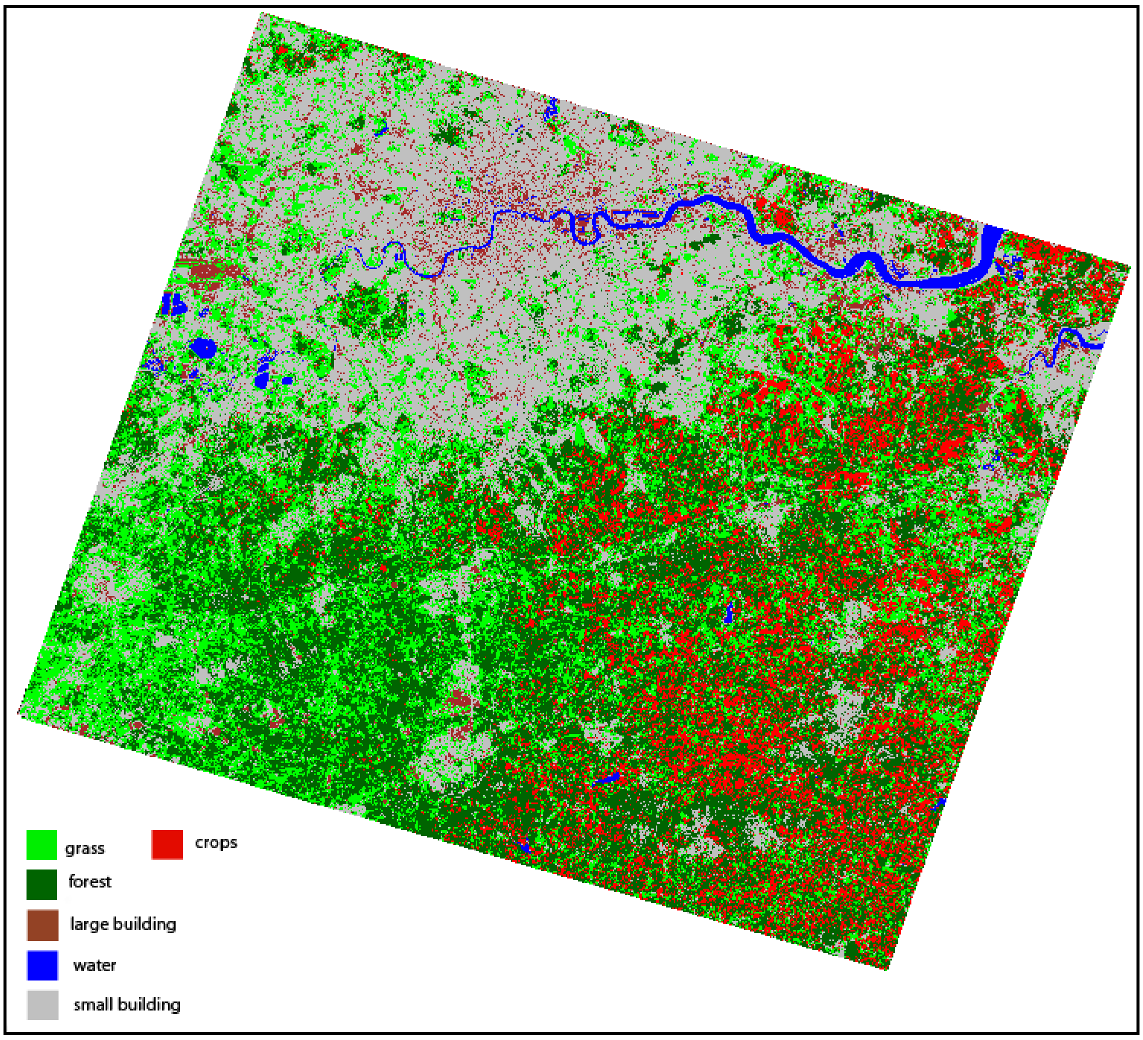


Figure 6.4: Supervised Classification of a SPOT-1 HRV image of South East London

Table 6.1: Confusion Matrix for Land-cover Classification

Land Cover Type	unclassified	small building	large building	forest	crops	grass	water
Unclassified	218	0	0	0	0	0	0
small building	0	39	3	2	0	3	1
large building	0	2	6	0	0	1	0
forest	0	3	0	23	10	26	0
crops	0	0	0	4	11	0	0
grass	0	1	1	2	0	39	0
water	0	0	0	0	0	0	5
Total test pixels	218	45	10	31	21	69	6
Overall Kappa Statistics = 0.7684							
Overall Classification Accuracy = 84.75%							

Following the approach set out by Barnsley and Barr (1996), nine land-use templates were created from the classified image; low-density residential, medium-density residential, woodland, arable farmland, permanent pasture, water, commercial/industrial, vacant / fallow land and unclassified. The templates were sampled at random from large training areas in the image using a-priori knowledge of the study area gained from large scale Ordnance Survey mapping; a 9x9 pixel kernel size was used. The land-use templates are detailed in Appendix K.

A wizard-based Java tool was developed to facilitate the preparation and uploading of input files and to handle job invocation and monitoring. Using this tool the job is prepared and invoked over seven stages; definition of land use templates, definition of mapping file, conversion of input file from NetCDF to SequenceFile, upload of input files to S3, submission and monitoring of job, download of result files and conversion back to NetCDF format.

6.4.2 Results

The resulting re-classified image is shown in Figure 6.5. An accuracy assessment was performed on the image which shows that the reclassification was successful. A confusion matrix for the reclassification is given in Table Table 6.2.

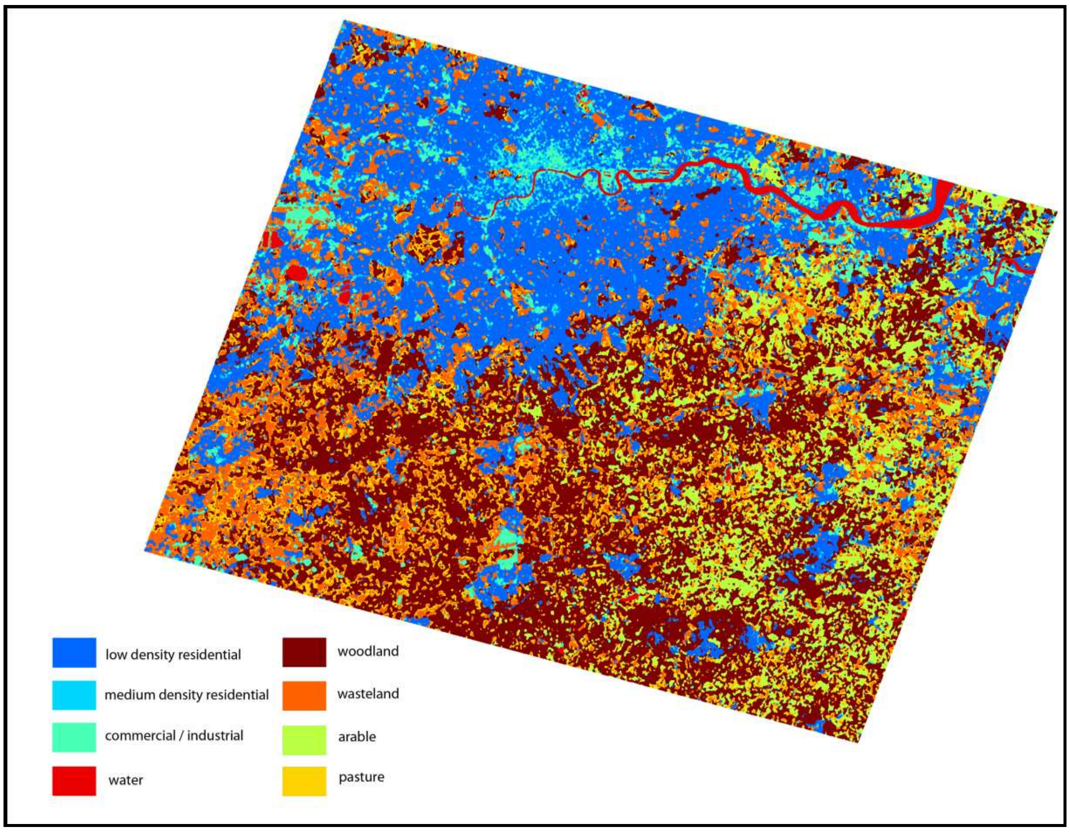


Figure 6.5: SPARK Re-classified Image of South East London

Table 6.2: Confusion Matrix for Land-Use Reclassification

Land-Use Type	**Unclassified**	**Water**	**Arable**	**Pasture**	**Woodland**	**Wasteland**	**Low density residential**	**Medium density residential**	**Commercial**
Unclassified	188	0	0	0	0	0	0	0	0
Water	0	3	0	0	0	0	0	0	0
Arable	0	0	9	1	0	0	0	0	0
Pasture	0	0	0	26	0	1	0	0	0
Woodland	0	0	7	13	35	5	0	2	0
Wasteland	1	0	1	4	1	15	1	1	0
Low density residential	3	0	0	2	0	1	45	20	3
Medium density residential	0	0	0	0	1	1	0	6	1
Commercial	0	1	0	0			0	0	2
Total test pixels	192	4	17	46	37	23	46	29	6

Overall Kappa Statistics = 0.7551
Overall Classification Accuracy = 82.25%

The algorithm was computed several times on different processing architectures in an attempt to measure the performance improvement resulting from parallelisation. The tested architectures include stand-alone Hadoop running locally on an Intel Core i-3, 2.13 GHz processor and Hadoop running on different sized Amazon EC2 instances using Amazon Elastic Map Reduce. Amazon measure computational power in EC2 compute units, one of which is approximately equivalent to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor (http://aws.amazon.com); specifications of the different Amazon instance types are detailed in Table 6.3. To provide a meaningful indication of performance the algorithm was also executed on a local standalone processor as a native java job. Results are detailed in Table 6.4 and Figure 6.6. Note that the processing time referred to in Table 6.4 includes upload and download times, not solely execution time.

Table 6.3: Amazon EC2 Instance Type Specifications

Instance Type	Instance Name	EC2 Compute Units	Memory (GB)	Default No. Concurrent Map Tasks Per Node	Default No. Concurrent Reduce Tasks Per Node	I/O Performance	Elastic MapReduce price per hour in EU / Ireland ($USD)
Small	m1.small	1	1.7	2	1	Moderate	0.015
Large	m1.large	4	7.5	4	2	High	0.06
Extra Large	m1.xlarge	8	15	8	4	High	0.12
High Memory XL	m2.xlarge	6.5	17.1	4	2	Moderate	0.09
High Memory Double XL	m2.2xlarge	13	34.2	8	4	High	0.21
High Memory Quadruple XL	m2.4xlarge	26	68.4	16	8	High	0.42
High CPU Medium	c1.medium	5	1.7	4	2	Moderate	0.03
High CPU XL	c1.xlarge	20	7	8	4	High	0.12

Table 6.4: Processing Time of the SPARK algorithm on increasing numbers of processors for a 4195 x 2995 pixel image and 9 land-use templates

Platform			Processing Time					
Hadoop / Native	Processor	No. Processors	Conversion to SequenceFile	Upload to S3 from local machine	Execution Time	Download from S3	Conversion to NetCDF	Total
Native	Intel Core i-3 2.13Ghz	1	N/A	N/A	1:13:20	N/A	N/A	1:13:20
Hadoop	Intel Core i-3 2.13Ghz	1	00:00:30	N/A	1:37:00	N/A	00:1:20	1:38:50
Mr EC2	Medium EC2 High-CPU	1	00:00:30	00:42:00	01:06:00	00:01:55	00:1:20	1:51:45
Mr EC2	Medium EC2 High-CPU	4	00:00:30	00:42:00	00:28:00	00:01:55	00:1:20	1:13:45
Mr EC2	Medium EC2 High-CPU	8	00:00:30	00:42:00	00:14:00	00:01:55	00:1:20	00:59:45
MR EC2	Large EC2	1	00:00:30	00:42:00	00:46:00	00:01:55	00:1:20	01:31:45
MR EC2	Large EC2	2	00:00:30	00:42:00	00:45:00	00:01:55	00:1:20	01:30:45
MR EC2	Large EC2	4	00:00:30	00:42:00	00:20:00	00:01:55	00:1:20	01:05:45
MR EC2	Large EC2	8	00:00:30	00:42:00	00:08:00	00:01:55	00:1:20	00:53:45
MR EC2	X Large EC2	1	00:00:30	00:42:00	00:30:00	00:01:55	00:1:20	01:15:45
MR EC2	X Large EC2	2	00:00:30	00:42:00	00:28:00	00:01:55	00:1:20	01:13:45
MR EC2	X Large EC2	4	00:00:30	00:42:00	00:14:00	00:01:55	00:1:20	00:59:45
MR EC2	Small EC2	1	00:00:30	00:42:00	04:06:00	00:01:55	00:1:20	04:51:45

MR EC2	Small EC2	2	00:00:30	00:42:00	04:05:00	00:01:55	00:1:20	04:50:45
MR EC2	Small EC2	4	00:00:30	00:42:00	01:28:00	00:01:55	00:1:20	02:13:45
MR EC2	Small EC2	8	00:00:30	00:42:00	00:46:00	00:01:55	00:1:20	01:31:45
MR EC2	Small EC2	16	00:00:30	00:42:00	00:24:00	00:01:55	00:1:20	01:09:45
MR EC2	Small EC2	20	00:00:30	00:42:00	00:25:00	00:01:55	00:1:20	01:10:45

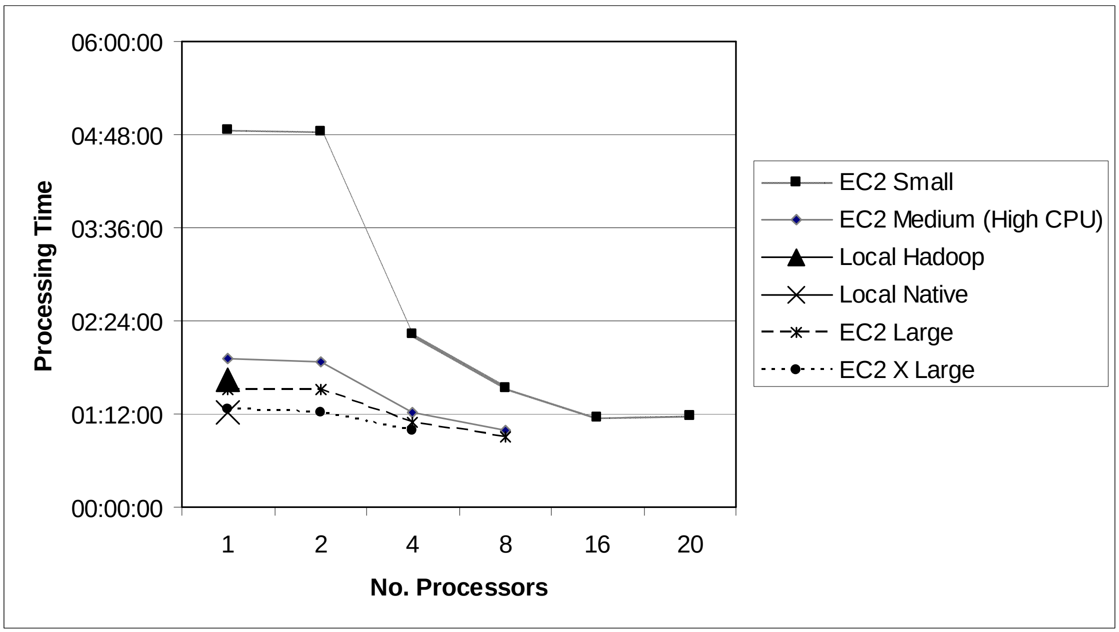


Figure 6.6: Processing Time of the SPARK algorithm for a 4195 x 2995 pixel image and 9 land-use templates using different Elastic Map Reduce configurations

It can be seen in Figure 6.6 that for each processor type, as the number of processors increase, the marginal increase in performance diminishes. Furthermore, when using more than 4 processors, there is very little difference in processing time for each instance type except for EC2 Small. Consequently, the deciding factor when selecting the appropriate processing type for a given job may come down to cost. In Figure 6.7, processing time is plotted against the total execution cost for the given EC2 processor types. It can be seen that at the $0.24 price point 8 Medium EC2 High CPU gives the best performance at 14 minutes, compared to 4 Large EC2 ($0.24, 20 minutes) and 2 XL EC2 processors ($0.24 28 minutes). Similarly at the $0.12 price point, 4 Medium EC2 High CPU computes in a favourable 28 minutes compared to a single XL EC2 which computes in 30 minutes. It can thus be concluded that the Medium EC2 High CPU instance type offers the best cost performance, although the single Large EC2 instance type offers the best value for money at the cheapest price point of $0.06, and also offers the best overall performance at 8 processors with an overall processing time of 8 minutes.

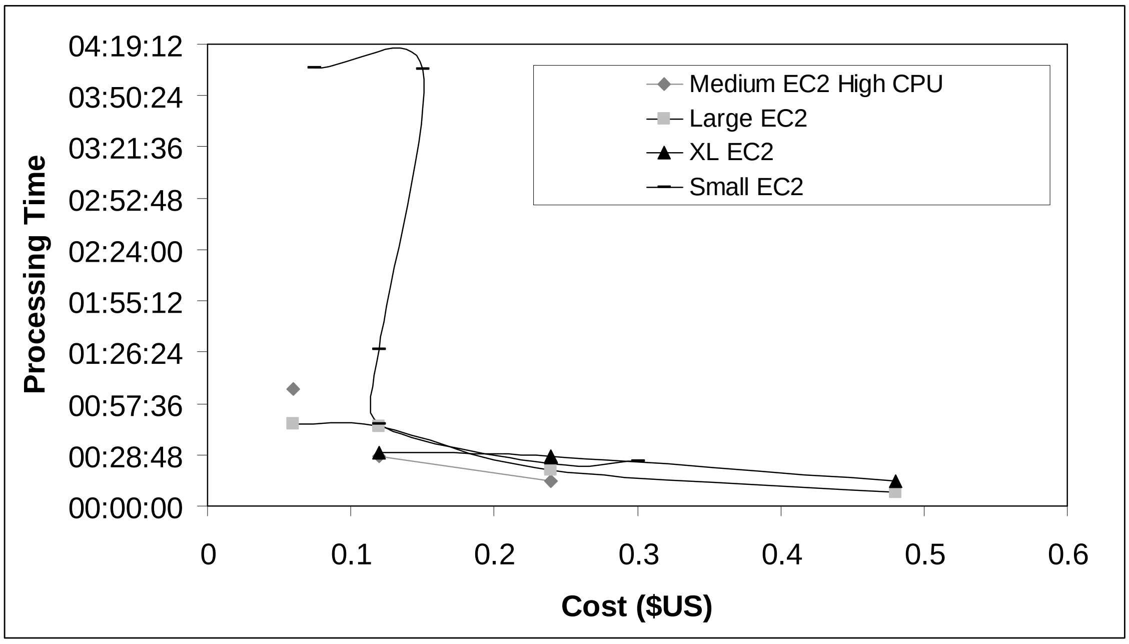


Figure 6.7: Graph Showing Cost Performance of Different EC2 Instance Types

6.5 Discussion

Overall the results show that MapReduce is capable of significantly reducing the execution time of the SPARK algorithm. The best results were achieved when the algorithm was run on 8 large Elastic MapReduce nodes which reduced the execution time to just 8 minutes, compared to 73 minutes using native java code. In the context of time critical scenarios it is likely that this reduction in execution time is sufficient to render the algorithm useful in many situations.

To provide a meaningful comparison to local code execution, the total processing times in Table 4 include data pre-processing, upload, download and conversion steps. It can be seen that despite reducing the SequenceFile volume by using a RowSet data partitioning strategy (Section 6.3.1) , the data upload still occupies a significant portion of total processing time to the extent that it almost negates the benefit of parallelisation for a job of this size. However, upload speeds from EC2 instances were found to be several orders of magnitude quicker than from a local machine; the upload time of 42 minutes for a 108MB SequenceFile was reduced to 3 minutes when uploaded from an EC2 instance. Consequently it can be argued that *processing* data on the

Amazon cloud is only viable for data that is also *stored* in the Amazon cloud when dealing with large datasets.

The results detailed in Figure 6.6 and Table 6.4 show that the processing time of the SPARK algorithm on a single local processor took only 73 minutes using native Java code but increased to 97 minutes using a locally executed Hadoop MapReduce instance. It seems likely that a large portion of this increase in processing time is the result of automatic fault tolerance measures built into MapReduce such as the calculation of checksums after each file block read; a measure that is designed to ensure data integrity. Other Hadoop mechanisms that could reduce performance include the reading of intermediate files from disk using a pull based approach (Dewitt and Stonebraker, 2008b) as well as the performance penalty suffered by running master and slave processes on the same node.

Figure 6.6 shows that executing the job on small EC2 nodes is considerably slower than using the more expensive instance types when using a small number of processors. However, this gap narrows as the number of processors increases. An out of memory exception prevented the scaling of larger instance types above 4 processors for x-large, and 8 for medium (high-cpu) and large. This is a result of Amazon's cluster configuration parameters that set larger instances to run a greater number of mappers and reducers concurrently, as specified in Table 6.3. Additionally, each process runs by default inside its own Java Virtual Machine and thus consumes a significant amount of memory. However, no attempt was made to adjust these configuration parameters given that an acceptable run-time of 8 minutes, on 8 large instances was achieved. Nonetheless it seems likely that execution time could be reduced further by fine tuning the cluster and job configuration parameters.

It was possible to translate the SPARK algorithm to the MapReduce paradigm with relative ease; this can be attributed to a number of factors. Firstly, because of the coarse-grained nature of the SPARK algorithm, individual subsets of the data aggregate could be processed independently of each other. This greatly simplified the task of parallelising the algorithm using MapReduce as individual data subsets could easily be mapped to key-value pairs. Secondly, the SPARK algorithm can broadly be considered as a unary transformation that operates primarily on one dataset. It should be noted that although a number of auxiliary datasets were required for the SPARK algorithm in the form of land-use templates, each of them was small enough to be distributed to each Map process without generating a significant IO bottleneck. It would be more difficult to translate a binary algorithm to the MapReduce paradigm because each Map process is only capable of processing a single input file. A number of methods have been suggested to overcome this problem such as the default Hadoop join which involves combining two datasets into a single input file using a common key, and the Map-Reduce-merge (Yang *et al.*, 2007) method that involves combining multiple MapReduce jobs in a workflow chain. These techniques require considerable effort to code and therefore simple coarse-grained binary operators such as raster intersection and union may be better achieved using a different distributed processing environment such as Condor (http://www.cs.wisc.edu/condor). Thirdly, the SPARK algorithm can neatly be divided into Map and Reduce processes. The calculation of Δk for each kernel window corresponds to a simple *Map* transformation and the process of finding the template with the maximum value of Δk corresponds to a simple *Reduce* operation. Although this feature of SPARK simplifies the coding of the algorithm in MapReduce it is not strictly required as MapReduce jobs can be defined without a *Reduce* stage.

For this work the Amazon cloud infrastructure proved to be suitable; MapReduce functionality was provided through an easy to use interface and a respectable speed up was achieved. However, some problems are envisioned for using both Elastic MapReduce and the cloud in general as a generic geoprocessing tool. The MapReduce environment is quite restrictive and as discussed above, it only appears to be suitable for certain types of computational job. Even for computational jobs that are well suited to this approach the environment is restrictive; the results outlined in Section 6.4 show that a considerable proportion of the total processing time was spent uploading data to cloud storage in the voluminous SequenceFile format as there was no mechanism to perform the data expanding transformation closer to the data source. Due to data locality the transfer times between EC2 nodes and S3 storage is reputedly much quicker than transfers from local machines to S3 (Murty, 2008). Consequently, the concept of co-locating data and processing capabilities is gaining popularity. There is a current drive in academia, realised through projects such as the Open Science Data Cloud (Grossman *et al.*, 2010), to host large scientific datasets in data centres with fast network connections that are capable of providing both persistent storage and computational analysis. It seems likely that large geospatial datasets such as satellite imagery archives will eventually be hosted in such an environment and this is expected to greatly expedite their processing and analysis (Gray *et al.*, 2005).

Another issue is that the level of performance achieved in a cloud cluster is unpredictable and can vary depending on the current workload the cluster is subject to (Armbrust *et al.*, 2009). This unpredictability of performance may present a problem for geoprocessing operations that have a hard real-time deadline. This can clearly be seen in the results set out in Table 6.4; a MapReduce job assigned 20 processor instances takes one minute longer to process than the same job assigned only 16 processor instances. Although the example presented in this Chapter considers the geoprocessing of a static dataset, there are numerous scenarios in which such datasets are required to be processed rapidly such as emergency disaster response. A potential solution to the lack of predictability in performance is to use cloud-aware scheduling (Schad, 2010) which compensates for poor performance as a result of an increased overall workload by allocating the given job to either faster processors or to a larger number of processors.

The third issue with geoprocessing in the cloud is the lack of interface and API standards in cloud computing models such as SaaS, PaaS and IaaS which could have severe consequences for cloud users in terms of vendor lock-in, a scenario that could result in an inability to transfer data or software between providers (Nelson, 2009). Commercial infrastructure providers have little incentive for standardisation (Buyya *et al.*, 2008) although it does seem likely that specifications from standards bodies such as the Open Science Grid (http://www.opensciencegrid.org), the Open Grid Forum (http://www.ogf.org) and the Open Cloud Consortium (http://www.opencloudconsortium.org) will eventually become adopted in scientific and academic systems.

6.6 Conclusion

This chapter has provided a working example of a CGSG system deployed in the Amazon cloud. The system successfully demonstrated how a unary, coarse-grained geoprocessing operation could be implemented in the MapReduce paradigm. However, MapReduce is only a suitable candidate for certain types of processing operation; namely coarse-grained operations on a single dataset. Currently, a lack of predictability in performance impedes the usage of the cloud as a generic geoprocessing platform; this is particularly the case for applications with a hard real-time requirement. Furthermore, the lack of standardised web service interfaces provide a barrier to adoption of cloud technologies as users are unwilling to become tied in to a particular proprietary system.

Chapter 7 Discussion

7.1 Introduction

This study set out to develop an appropriate conceptual and implemented framework in which open standards in grid computing, sensor web and geospatial web services could be combined, as a technological basis for monitoring and prediction of geospatial phenomena in the Earth systems domain. The results show that it is possible to significantly improve the performance and scalability of geoprocessing tasks in Earth systems monitoring and prediction using distributed computing. The framework set out in this thesis has outlined three styles of geoprocessing that occur in monitoring and prediction systems, and has demonstrated how they can be implemented in a manner that overcomes the limitations imposed by single processor architectures through the use of parallelisation techniques. Furthermore, it has been demonstrated that open web service standards in the geospatial and distributed computing domains can be integrated despite apparent incompatibilities in web service policy, style and message semantics. The goal of this chapter is to draw on the relevance of these findings, to address the limitations of this study, and to relate this work to other research in the field.

7.2 Harmonisation Issues

It was hypothesised that integrating SWE, OWS and distributed computing standards could facilitate the development of high performance, scalable and loosely coupled applications that rely on live sensor data, such as monitoring and prediction systems. Various methods of integrating near real-time sensor data into distributed geoprocessing workflows have been considered throughout this thesis. The three implementations (Chapters 4-6) have highlighted some important issues relating to standards integration and interoperability. In this work we have only considered open standards published by the major standards organisations in each domain; the OWS and SWE frameworks published by the OGC, the OGSA framework published by the OGF, and the WS-Notification and WSRF specifications published by OASIS. Currently these frameworks represent consensus on best practice in industry and academia in their respective domains. With regards to cloud services the discussion refers to Amazon Web Services (EC2 and S3) which, in the absence of formally recognised standards have become a de-facto industry standard.

7.2.1 OGC-OGF Harmonisation

There has been considerable progress in integrating OGC and OGF standards in recent years but there is still a long way to go before these frameworks are fully harmonised. One of the main issues is that full adoption of WS-I by OWS and SWE has not yet been realised. WS-I standards are of significance because they provide a common messaging and interface format for both OGC and OGSA standards.

Compatibility between WS-I tools and OGC schema is another issue that has yet to be resolved. WS-I tools greatly simplify the process of grid service development as they enable service and client stubs (service implementation) to be automatically generated from WSDL documents (service interface). It was found (Section 5.3.2) that standard web service tools such as Apache Axis2 (http://axis.apache.org/axis2/java/core/tools/index.html) and the Metro stack (http://jax-ws.java.net/) were unable to parse the SES WSDL document and so were unable to generate client stubs. This suggests that there is still a certain level of incompatibility between OGC schema and web service tools. This was also experienced in the OGC SOAP Interoperability Experiment in which the same issues were encountered (Sonnet and Savage, 2003). As the SES is not an official OGC standard its WSDL may not follow specific OGC conventions but it is expected that the forthcoming SOS version 2.0 specification (Broering *et al.*, 2010) will provide a more stringent test of compatibility. Several of the problems encountered in the SOAP Interoperability Experiment could only be rectified by altering the schema or by combining all the schemas into a single file. These problems suggest that there is still work to be done on either improving web service tools or in improving OGC schema compatibility with such tools.

Progress has been made in integrating OGSA job execution standards such as JSDL and OGSA-BES with the WPS. Woolf and Shaon's (2009a) JSDL-WPS profile extends the capabilities of WPS job execution in a grid environment by enabling end users to select their hardware requirements and thus the speed at which a given task will run. In the map-matching system (Chapter 4) it was deemed unnecessary to integrate JSDL into WPS because provided that incoming observations could be processed more quickly than the data arrival rate there was no performance advantage to be gained from running the task on a faster machine. Consequently the JSDL was hard wired into the WPS in an opaque manner. This had the effect of shielding the WPS user from the unfamiliar JSDL schema, albeit at the expense of flexibility in resource provisioning.

7.2.2 Improvements to SWE Data Services

With regards to SWE data services it appears that there is considerable scope for existing and proposed standards to be consolidated. For example, in order to interface the SES with the SOS in the road traffic monitoring system (Chapter 5) it was necessary to create a bridge program to poll the SOS for new observations and forward them to the SES as SOAP encapsulated WSN compliant notifications. Although the adopted approach was relatively straightforward it is noted that interoperability between these components could be significantly improved. There is no reason why both the SOS and SES service interfaces could not be implemented by a single service instance that provides both push and pull based access to observations. This would have the potential to greatly simplify the road traffic monitoring system, as three components, the SOS, SES pusher and SES, could be reduced to one. Furthermore this would enable overall system latency to be reduced as observations could be forwarded to consumers as soon as they arrived rather than waiting for the SOS to be polled. According to the OWS-7 engineering report (Fairgrieve, 2010) there is currently ongoing discussion within the OGC as to whether asynchronous filtering and notification should be incorporated into the SOS specification.

Certain design features of the SES encourage better performance and interoperability compared to the SAS design. The incorporation of CEP into the SES enabled voluminous observations to be condensed into a smaller amount of relevant information, thus significantly reducing the processing and messaging overheads further on in the geoprocessing chain. In terms of interoperability the inclusion of WSN specifications is likely to improve compatibility with other services in both the web and grid domain, given their widespread adoption.

7.2.3 Improvements to the WPS

The WPS specification is not designed for running continuous open-ended jobs. In the map-matching system the WPS interface was modified to enable continuous jobs to be managed. Although the method of launching a grid process for each sensor resulted in resource provisioning inefficiency (Section 7.4.1) it did enable real-time observations to be processed relatively quickly as job scheduling delay was only encountered when starting the job rather than for every observation. It is suggested that for continuous jobs the *Execute* operation be replaced by *startExecuting* and *stopExecuting* operations to provide a clearer management interface for continuous processes (Section 4.3.5).

Rather than providing the WPS with all the input data on invocation, a URL reference to the SOS repository was provided so the process could dynamically poll this repository for new observations. A proposed extension to the WPS (Woolf and Shaon, 2009b, Woolf and Shaon, 2009c) to handle asynchronous jobs includes the ability to pause and cancel processes, which proved to be useful for controlling long-running processes. For example, the existing *Execute* operation could be used to start a long-running job and the proposed cancel operation could be used to stop the job. In the Sensors Anywhere discussion paper (Uslander, 2009) this style of execution is referred to as cyclic and it is suggested that a total duration or total number of cycles be provided as an *Execute* input parameter rather than allowing the process to run continuously. This time-out concept presents an improvement to the approach adopted in the map-matching system because it prevents a job from running eternally in the event of user carelessness or accidental loss of the job's unique identifier.

7.2.4 OGC Services using IaaS and PaaS

Despite the current lack of standards in the cloud computing domain, it presents a promising solution to the performance and scalability problems facing OWS and SWE services. Presently cloud computing is driven by large IT companies for commercial gain. Many organisations consider this a disincentive to cloud adoption, preferring instead to use community driven and standards-based grid infrastructures. By their very nature cloud data centres must be large to benefit from economies of scale (Buyya *et al.*, 2008). However, cloud infrastructure does not necessarily have to be operated by corporations and we are likely to see the cloud model adopted by academia in the future. Current open source cloud software such as Eucalyptus (http://www.eucalyptus.com) and OpenNebula (http://www.opennebula.org) present a solution for organisations to host their own private cloud and it is likely that emerging academic cloud test beds will be based on these.

Usability is a primary motivation to use cloud technology. Commercial cloud infrastructures offering IaaS and PaaS are significantly easier to use than grid services. From an end user's perspective, access to a grid service requires a valid grid certificate which must be installed correctly and this alone requires some degree of expertise in grid security. Grid portals and Problem Solving Environments go some way to alleviate this problem by providing a user friendly interface to grid applications, but these also add another layer of complexity to grid application development. From the developer's perspective, expertise in SOAP based web services is mandatory in order to develop and deploy applications on the grid. Furthermore, almost all grid middleware can only be deployed on Linux based platforms. In comparison, applications built on IaaS and PaaS do not require distributed computing expertise to use. Additionally, cloud service providers present easy to use web based management consoles to manage the virtual machine lifecycle, as well as providing a set of well documented RESTful APIs to the developer.

Standardisation in IaaS and PaaS is important for cloud users. For example, in the road-traffic monitoring system the Amazon EC2 IaaS was used to scale-out the data input subsystem by building an AMI containing the required software and data. It would have been very difficult to port this AMI to a different infrastructure provider because the AMI file format is not an open standard. However, the adoption of the proposed Open Virtual Machine Format (Section 2.3.6) would solve this problem. IaaS could benefit from standardisation in three areas; web service lifecycle management interface, virtual machine image file format, and the adoption of a common security model. Major GIS vendors are beginning to develop cloud based GIS platforms and this trend is likely to continue. For example, it is already possible to deploy ESRI’s ArcGIS on EC2 instances, and ERDAS Apollo is available as a cloud service for a monthly subscription.

7.3 Performance Issues in Distributed Monitoring and Prediction

Monitoring and prediction systems collect vast quantities of data which must normally undergo some filtering and analysis before it can be used in decision support scenarios. In the implemented systems job scheduling (Chapter 4), data I/O (Chapters 4-5) and data transfer (Chapter 6) were found to be the major bottlenecks. The problems encountered and potential solutions to these are discussed below.

7.3.1 Job Scheduling

The timely scheduling of grid compute jobs is problematic for on demand monitoring and prediction applications that have a hard real-time deadline. In itself job scheduling is an NP-hard problem (Moreno, 2003) in which queued jobs must be allocated to available resources while minimising some user cost function. Sufficient resources are not always available and significant scheduling delays may result. A 28 minute scheduling delay was experienced in the map-matching system which allowed hundreds of GPS observations to go unprocessed because their processing window had ended before the job was launched. For mission critical monitoring and prediction applications it is often necessary for all observations to be processed within a short time window (Kopetz, 1999). The majority of geohazard early warning and post-disaster management systems fall into this category and for such systems the significant delay experienced here would be unacceptable.

If there are insufficient available resources to run all the jobs then scheduling delays are unavoidable. However, grid interoperability has made it possible to balance a job workload not only within a compute cluster, but also between clusters and in some cases between grids (Yagoubi and Slimani, 2007). Recruiting from a wider pool of resources in this way improves the chance of obtaining an available processor. As yet, resource sharing amongst cloud providers is not widespread despite the economies of scale it provides. It seems likely that more cooperative academic clouds will emerge in the future but this is dependant on the definition and uptake of open standards.

Many scheduling systems enable computing resources to be reserved in advance for a specific time of day (Xing *et al.*, 2004). This capability is likely to be of use only for monitoring and prediction applications for which the computational requirements are known in advance, rather than for event driven applications. Service Level Agreements (SLA) are a bilateral agreement between the service provider and consumer (MacLaren *et al.*, 2004) which have emerged as a more flexible way of negotiating scheduling priorities. The OGF's WS-Agreement specification (Andrieux *et al.*, 2007) is used for negotiating SLAs; one of its goals is to provide assurance to the consumer of the level of service they can expect. For monitoring and prediction systems SLAs have the potential to guarantee quality of service in terms of job scheduling delay (Baranski and Schäffer, 2010).

Job scheduling algorithms often require an estimate of each job's execution time in order to optimise scheduling (Malarvizhi and Uthariaraj, 2009). Due to their continuous style of execution the compute jobs in the map-matching system would not be able to provide this information, as the length of job is not known at the time of job submission. At the NGS, jobs are scheduled using a fair share policy; those users who have used the least computing resources in the recent past are given priority. For event driven monitoring and prediction applications this is likely to pose a problem because the level of access to computational resources will be restricted after a major computational event has occurred.

It can be concluded that despite difficulties in job scheduling it is possible to use grid processing for near real-time geoprocessing applications, provided that a SLA is in place that guarantees a minimum provision of service. The standardisation of resource negotiation through the uptake of WS-Agreement is likely to be of benefit to monitoring and prediction applications as this provides a mechanism through which they can be assured a particular level of service. Given that the processing time of a job is often required in advance by scheduling algorithms it may be concluded that continuous running compute jobs are not always a satisfactory solution for data streaming applications. This problem can be circumvented by scheduling numerous finite jobs as opposed to a continuous job but is likely to suffer a performance penalty due to scheduling overheads.

7.3.2 Data I/O

According to Szalay and Blakeley (2009) data access is becoming the major limiting factor in computing systems. This view was reinforced by the scalability results from both the map-matching and traffic monitoring systems. Poor performance can be attributed to hardware constraints such as slow disk seek times and low I/O bandwidth. Such issues are exacerbated when there is a lot of competition for storage resources, and in database systems this often manifests in the form of deadlock which impedes performance further. Database deadlock occurs when two or more transactions are each waiting for locks to be released that are held by the other (Connolly and Begg, 2005). In the map-matching system it was found to occur in the SOS database and caused a 23 minute processing delay. The unpredictable but generally poor response time observed in the WFS and SOS at an increasing number of user requests suggest that bottlenecks are the combined result of database deadlock and more general I/O constraints.

Poor availability in non-transactional data stores is commonly alleviated by replicating the data store amongst several servers and load balancing the incoming requests in a round-robin fashion (Cardellini *et al.*, 2002). The WFS in the map-matching system could have been replicated in this way because it is used in a read-only manner, but it is rarely possible to replicate a transactional data store because it violates data consistency in accordance with Brewer's CAP theorem (Lynch and Gilbert, 2002). Strong data consistency is important in the map-matching SOS. For example if a *describeSensor* query were performed on one data replica before *insertObservation* requests from another data replica were made consistent, then many observations would go unprocessed. An alternative to replicating the data source is to minimise the number of transactions at the data store. This is achievable in data streaming applications by filtering out irrelevant observations before committing them, or by aggregating observations before insertion as this reduces the overall transaction lock time.

On the other hand, some spatial data services may tolerate eventual consistency. Cloud services provide an *eventually consistent* platform from which high availability spatial data services can be published. For example, Amazon S3 has adopted the eventual consistency model and the Google App Engine API provides both strong and eventually consistent data access. Sensor data services that handle a large volume of incoming observations but that are only occasionally subjected to batch queries are likely to be well suited to an eventually consistent data store. A SOS backed by a distributed, document oriented (NOSQL) CouchDB database (http://couchdb.apache.org) has recently been developed as part of the GeoCENS project (Liang *et al.*, 2010). This database has a strongly consistent data model but is still likely to provide a significant performance advantage over relational databases as it has a lock-less update model. Lock-less update enables updates to be committed on a first come first served basis, with transactions either being committed or failing completely. A performance comparison between the GeoCENS SOS and relational SOS such as the 52North implementation would make an interesting piece of future work. Similarly, the development of an eventually consistent SOS using cloud services and performance comparison to its strongly consistent counterparts would also be worthwhile extension to this thesis.

In the road traffic monitoring system the insertion of new observations into the database was found to present a bottleneck which was caused by executing a complex pre-processing trigger on the insertion of each observation. The trigger determined the direction of each vehicle in relation to the road, the travel-time spent on each road, and estimated the path taken between non-adjacent road links. A potential solution to this bottleneck would be to create multiple read-only replicas of the road network database, and load balance the pre-processing queries between replicas. In this way the pre-processing overhead would be separated from the transactional updates to the road network travel cost.

It was found that the road network dataset used in the traffic monitoring system was too large to load into main memory and this prevented analysis from being performed on the road network. This was solved by horizontally partitioning the road network tables using regular data decomposition, and using the load-on-demand API provided by Oracle to analyse each network partition sequentially. A secondary motivator for this approach was to bolster query performance in the pre-processing trigger. However, it was not possible to compare the performance of the trigger over partitioned and non-partitioned tables due to the lack of available dense GPS tracks. As only sparse GPS tracks were available, the trigger was redesigned to incorporate shortest path analysis to determine the path between non-adjacent road links, which meant that partitioned road network tables had to be employed to run the analysis.

It can be concluded that it is difficult to scale-up transactional data sources in a distributed architecture and that they can often present a considerable bottleneck. Eventual consistency may be appropriate for some sensor data applications and an eventually consistent SOS has been suggested as an interesting future research topic. Data de-clustering through horizontal partitioning goes some way towards alleviating data access bottlenecks (Cruanes *et al.*, 2004) but in the road traffic monitoring application, queries over the partitioned road network still suffered from poor response times.

7.3.3 Data Transfer

Data transfer is a significant problem in distributed systems because bandwidth improvements have not kept pace with improvements in storage capacity (Gray *et al.*, 2005). It is recognised that co-locating data and computation is the preferred solution for processing large datasets (Skillicorn, 2002). In the context of sensor geoprocessing this suggests that sensor data repositories should be hosted in grid or cloud data centres to enable analysis to be performed in close proximity to the data. The key advantage of geoprocessing at the data source is that costly network data transfers are minimised, thus providing a performance advantage over the alternative method of sending data to a remote geoprocessing service. However, this performance advantage can only be realised if the operation is data-reducing in nature, i.e. the resulting feature set is smaller than the original dataset (Friis-Christensen *et al.*, 2007).

The approach taken in the road traffic monitoring system was to co-locate data and computation in a parallel relational database. The relative merits and shortcomings of this approach are discussed in detail in Section 7.2.3. An alternative approach to database geoprocessing is high-level gridification whereby data and processing services are hosted on the grid. Besides fast interconnects between storage and computational nodes, GridFTP can be used to rapidly transfer data between locations using multiple parallel streams (Allcock *et al.*, 2003), and OGSA-DAI can be used to federate access to multiple databases. This approach was exemplified in the SEE-GEO project (http://www.edina.ac.uk/projects/seesaw/seegeo) in which spatial extensions to OGSA-DAI were developed to provide access to grid-based data services from a standard web service client. However, this method requires a significant development effort.

The same end result of close proximity between data and processing services can be more easily implemented on the cloud infrastructure as exemplified by the SPARK MapReduce work (Chapter 6). The main obstacle in this implementation was the excessive time taken to upload a classified satellite image to cloud storage. Network bandwidth is not responsible for this bottleneck, as the observed download speed for the re-classified image was 28 times faster than upload speed. This discrepancy between upload and download speed is a phenomenon known as asymmetric communication, and in many cases is artificially introduced by internet service providers to account for the imbalance between upload and download volume of typical internet users (Bose *et al.*, 2003).

In this thesis it has largely been assumed that HTTP can be used for the transportation of data from sensor devices and data services to geoprocessing services. However, the uploading of data to distributed storage within reasonable time constraints presents a considerable challenge considering the asymmetric communication phenomenon. To avoid upload bottlenecks it is necessary for organisations that carry out computational geoprocessing on large data sets with a near real-time requirement to migrate the entire workflow to a data centre, or to host their own data centre in the form of a private cloud or compute cluster. Gray and Patterson (2003) stated that the cheapest and fastest solution to upload very large datasets was to send a disk to the data centre via postal services; this is still the case today and Amazon have began to offer such a service for uploading data to S3. This method is not viable for near real-time datasets although it does present a useful alternative for the occasional transportation of large data archives. Extremely poor transfer time from a standard web client to S3 was observed in the SPARK MapReduce work to upload a file representing a single satellite image. For real-time analysis of satellite imagery it would thus make sense to couple satellite receiving stations with data centres, either by locating them in the same place, or by creating a fast network link between them.

For geoprocessing tasks that operate on small data items there is less of a case for co-locating data and computation. For example, in the map-matching architecture the actual algorithm was processed in a computational grid, but spatial data was retrieved from web components exposing OGC compliant interfaces. The system was expected to withstand the physical separation of data and processing components because in each case only small volumes of data were being transferred. Indeed, data access was found to be the limiting factor in this system rather than bandwidth constraints. This data access bottleneck was the result of excessive load being placed on the spatial databases behind the data services, so locating the data services on the grid in this case would have had little effect on performance. Instead, addressing the database bottleneck through techniques such as de-clustering (Section 2.4.2) is likely to have achieved better results.

7.4 Methodologies for Real-time Distributed Geoprocessing

A review of parallel geoprocessing techniques was conducted in Section 2.3. It was found that most of the techniques assume a static dataset and that the introduction of real-time data poses a new set of challenges. In this Section an evaluation of the typology, and the techniques employed within the implementation Chapters is conducted in an attempt to highlight generic methods of distributing real-time geoprocessing operations.

7.4.1 Data Stream Geoprocessing (DSG)

It was hypothesised that processing multiple streams of geospatial data from a sensor collection could be achieved by allocating one processor to each sensor data stream. This approach was used to develop a grid based map-matching system to process multiple streams of vehicle GPS observations concurrently. The major advantage of this approach was its simplicity; minimal software development was required to process numerous streams of data concurrently. Although this technique did prove to be effective it was noted that for sparse data streams the monopolisation of grid computing resources for long periods using this configuration was likely to be wasteful. Conversely, for dense data streams the workload could overburden a single processor and result in the development of a processing backlog. Thus it can be concluded that the single sensor stream per processor approach is valid for some DSG applications but is likely to be unsatisfactory in the majority of cases due to its lack of flexibility.

The problem of resource provisioning for stream-based processing systems is often complicated because unpredictability in the rate of data arrival is typical (Babcock *et al.*, 2002). This difficulty is further compounded when attempting to perform processing on grid systems because grid resource changes and failures are common (Kalogeraki *et al.*, 2008). In the Data Stream Management System (DSMS) literature a variety of methods are suggested to overcome this problem of resource provisioning such as load shedding and data stream partitioning.

Load shedding is a commonly used technique in which some of the incoming observations in a data stream are dropped to ensure that a processing backlog does not develop. Essentially load shedding sacrifices the quality of observations in terms of loss ratio or sampling rate to ensure a shorter processing delay (Tu *et al.*, 2006). Although this trade off may be acceptable for some applications it is likely to be unacceptable for mission critical monitoring and prediction systems.

An alternative to load shedding is data stream partitioning, a technique that parallelises the processing of a data stream using either a functional decomposition or a data decomposition. Functional decomposition is relatively easy to achieve using a processing pipeline. Data decomposition is more difficult, particularly if there are dependencies between the data items in a stream. In the simplest case, each data item can be processed independently of each other. In a more complex scenario, the processing of each data item relies on information from a window of preceding data items. Finally, the most complex scenario involves the detection of complex events, i.e. events that are an abstraction of other events (Luckham and Schulte, 2008). These different levels of dependence are respectively termed atomic transformations, stream-dependant transformations and event correlations. In the examples implemented in this thesis, the map-matching algorithm can be considered a stream-dependant transformation, because the determination of vehicle orientation requires both the current and the preceding data item in the stream. In the traffic monitoring system, road change event detection falls into the event correlation category because it is impossible to predetermine the number of data items (map-matched observations) that will occur between road change events. For example, in a given time window or count window there is no guarantee that the vehicle will move onto a different road.

Cherniack *et al* (2003) present two data stream partitioning approaches termed box sliding and box splitting that are used in the Aurora DSMS. The former approach uses a functional decomposition whereas the latter uses data decomposition. Query execution in DSMS is achieved using a processing pipeline which can be conceptualised as a chain of boxes in which each box represents a process. In a distributed DSMS the processing pipeline may span several processors although more than one box may be located on each processor. Box sliding is the practice of relocating a box from processor 1 to another processor that sits immediately before or after processor 1 in the pipeline. Shifting a box upstream is recommended if a data reducing operation is being performed, whereas shifting a box downstream is useful if the operation is data expanding in nature. The process of box sliding is a useful method of balancing the load in a processing pipeline. Box sliding can be applied regardless of the level of dependency between data items, but the processing operation must be composed of multiple independent stages. Figure 7.1 demonstrates upstream box sliding.

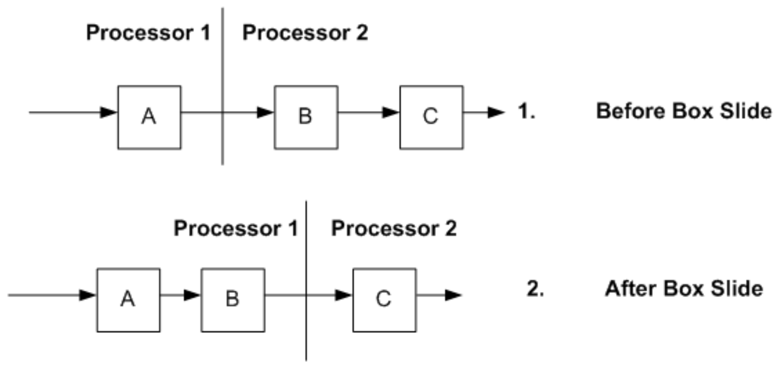


Figure 7.1 Upstream box sliding: Process B is moved from Processor 2 to Processor 1 [adapted from Cherniack *et al.*, (2003)]

Box splitting is the practice of duplicating a box to another processor and diverting some of the load from the original box to the newly duplicated box. Box splitting requires that a new box be placed on each side of the split process; the upstream box is a filter that divides the data stream amongst the split processes, and the downstream box merges the results. Box splitting is depicted in Figure 7.2. It should be noted that to maintain the integrity of the data stream the order of observations should be preserved, thus the merge operator is also required to sort the observations into their original order.

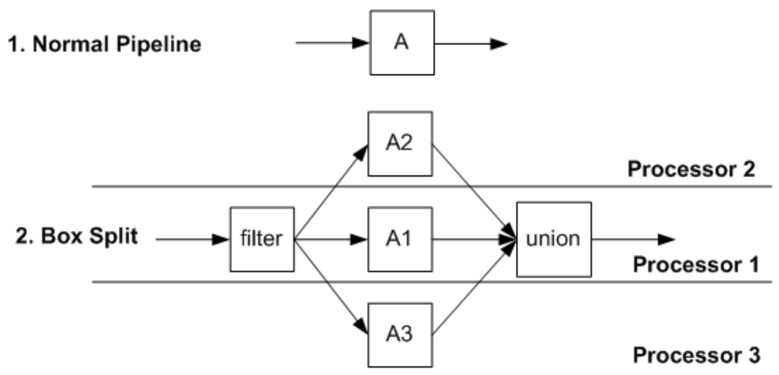


Figure 7.2 Box Split: Process A is duplicated on Processor 2 and Processor 3, the filter operator equally allocates incoming observations amongst the three processors [adapted from Cherniack *et al.*, (2003)]

Atomic transformations can easily be processed using the box-splitting method; the Aurora (Carney *et al.*, 2002) and Borealis (Abadi *et al.*, 2005) distributed DSMS both use this approach. A requirement of this approach however, is that to achieve a significant speed-up the cost of the computational phase must outweigh the cost of merging the results (Brito, 2008). Processing stream-dependent transformations is more difficult because it requires incoming observations to arrive or to be retrieved in order. Data stream sources can be unpredictable and thus there is no guarantee that observations will arrive in the correct order (Tatbul *et al.*, 2003); this is particularly the case when observations from multiple streams are retrieved from a single endpoint. For example, in the map-matching system this was experienced when the SOS was overloaded by incoming observations and deadlock occurred in the database causing some observations to arrive out of order. Commonly, this problem is mitigated by buffering the stream before processing (Babcock *et al.*, 2002, Abadi *et al.*, 2005) as this allows delayed observations to arrive before they miss their processing window.

A potential problem with box-splitting is that the procedure of creating a new instance of the process may incur a significant delay attributed to job scheduling, dependant on the current level of usage at the grid cluster (Section 7.3.1). In DSG systems observations will continue to accumulate while job scheduling is taking place so an important requirement is that sufficient storage is available to ensure observation persistence. Mission critical monitoring and prediction systems may not be able to tolerate such scheduling delays in which case a prioritised scheduling policy or a dedicated compute cluster may be necessitated. On the NGS it is possible to reserve CPU time but this requires advance knowledge of the amount of resource required, which is not always possible for streaming applications.

Brito (2008) presents optimistic parallelisation as a technique to accomplish complex event detection in parallel. The problem with processing such tasks in parallel is that consecutive observations or events may be processed out of order resulting in an incorrect solution being computed. In optimistic parallelisation observations or events are scheduled for processing based on their timestamp; processing begins as soon as the likelihood of the item being processed successfully becomes acceptable. Using this method items may be processed out of order, but the output from the process is not committed until preceding items have been processed. If a preceding item affects the outcome of the item that was processed out of order then it is recalculated. Although this technique wastes some compute cycles it is capable of processing data streams more quickly than is possible using sequential methods and ensures that the correct output is generated.

The system presented in Chapter 4 highlighted the inefficiencies of the single sensor per processor approach. Grid computing was used in the map-matching system to process numerous data streams in parallel. However, for very dense data streams, or for computational processing, it may also be necessary to parallelise the processing of each stream. Possible solutions include load shedding, box-splitting, box-sliding and optimistic parallelisation. In this regard, the DSG category of geoprocessing operation could be further subdivided into the aforementioned atomic transformations, stream-dependent transformations and event correlation to better reflect the possible methods of parallelisation.

7.4.2 Fine-grained Snapshot Geoprocessing (FGSG)

Two methods of performing FGSG were suggested in Section 3.4. For fine-grained geoprocessing operations with a relatively simple computational complexity it was proposed that a spatial database be used to perform the computations. However, for more computationally intensive operations the use of the message passing paradigm was suggested.

The approach taken in the road traffic monitoring system was to position data storage and processing together in a parallel relational DBMS. The key merit of this approach is its simplicity to implement. Additionally, the expression of geoprocessing operations in SQL is advantageous because of its declarative style; query execution is kept separate from task definition. As a result, process optimisations can be performed by the query interpreter using indexes, without altering the query statement.

However, there are also a number of drawbacks to geoprocessing at the database. Firstly, many geoprocessing tasks operate on raster and coverage datasets but relational database geoprocessing is biased towards relationally structured data. Nevertheless this is changing, and support for rasters and coverages amongst commercial and open source spatial databases is growing. For example, Oracle Spatial can store raster data and supports a wide variety of processing operations (Kothuri *et al.*, 2007) and similarly PostGIS 2.0 is set to include full raster support (Racine, 2010).

Secondly, the expression of particular geoprocessing algorithms in SQL can be problematic. For example, it is not possible to declare variables, loop through a feature set, or use exception handling in SQL and this severely restricts its use as a geoprocessing language. Consequently Simple Features for SQL

(Herring, 2006) and the SQL/MM (Stolze, 2003) specifications concentrate primarily on low-level geoprocessing operations. In some cases database vendors have applied their own extensions to the SQL language to make complex geoprocessing functionality available. For example, it is possible to perform nearest neighbour analysis using SQL commands in Oracle Spatial. Alternative procedural and object oriented database programming APIs provide a more powerful interface that enable higher level operations to be performed, but these have not been standardised.

Another problem with database geoprocessing is that certain operations may need to reference data from sources external to the database, or the database may simply be unable to handle the memory or computational requirements of a particular operation. Furthermore, this approach introduces a tight-coupling between data and processing operations which goes against the principles of distributed architectures. For example, SQL requires table and column names of a particular dataset to be hard-coded into the task definition. Recent work by González Cortéz and Leduc (2010) has attempted to overcome this constraint using Gearscape Geoprocessing Language, a geoprocessing language that implements spatial SQL. The presented approach decouples the geoprocessing script from the data using tables and literals as parameters.

Overall the database geoprocessing technique can be considered effective. Improvements could be made to existing spatial database implementations by building in more high-level geoprocessing functionality. The nearest neighbour functionality in Oracle Spatial was found to be extremely useful, and this could be adopted by other spatial database vendors. Similarly, other fine-grained operators such as Theissen polygon creation and line-of-sight analysis could be made available as pre-defined functions. Such extensions would make an interesting topic for further research, particularly if they were designed to take advantage of distributed databases by running on several processors concurrently. Additionally, further investigation into the use of MPI for fine-grained geoprocessing is required. No attempt was made to use MPI in this work because of the large amount of development work involved. However, the development of a standard library of parallel geoprocessing tools that takes advantage of MPI is likely to be useful considering that MPP clusters on which to run such functions are now widely available through grid and cloud computing interfaces.

7.4.3 Coarse-grained Snapshot Geoprocessing (CGSG)

The parallel geoprocessing of coarse-grained tasks has reached a high level of maturity. Several tools and frameworks are available to facilitate the execution of embarrassingly parallel problems on cluster, grid and cloud platforms. Numerous examples of using such tools to perform coarse-grained geoprocessing in parallel are presented in the literature and in nearly all cases a significant speed-up is achieved.

As stated in Section 3.4, snapshot geoprocessing shares many similarities with static geoprocessing and the main difference is that CGSG operations are triggered by a real world event, rather than through manual invocation. In the parallel SPARK implementation (Chapter 6) no attempt was made to define a scenario, or to implement a service based framework in which the system was triggered by a real-world event. The main reason for this omission was due to the amount of manual user input required to prepare the initial classified image and the predefined land-use templates.

The MapReduce programming model proved to be suitable for CGSG. With regards to the creation of tools that enable parallel processing operations to be rapidly developed, a geospatial MapReduce format converter would appear to be a useful asset. For example, a tool to translate common image file formats into Hadoop SequenceFile format would reduce the development effort of writing MapReduce geoprocessing code. The proposed tool could offer a number of partitioning strategies that roughly follow the approach taken in Section 6.3.1. For example, partitioning options could include both overlapping and non-overlapping *Windows*, *RowSets* and *ColumnSets*. A similar tool could be developed to convert a set of output files back into a recognised image format. A raster algebra MapReduce tool, MrGIS is currently being developed by Chen *et al* (2008) and this is expected to contain some similar functionality.

Unfortunately MapReduce is not a standardised framework and this is likely to impede future work on the integration of MapReduce with geoprocessing. Recently, the MapReduce patent was granted to Google Inc. (Dean and Ghemawat, 2010) and this potentially puts the future of open source MapReduce projects such as Hadoop in jeopardy. Although it seems unlikely that the patent will be vigorously enforced it still presents a significant disincentive to the use of MapReduce in open geoprocessing systems.

7.5 Conclusion

It can be concluded that interoperability problems in grid based geospatial monitoring and prediction systems could be mitigated by making recommended changes to SWE data services and the WPS interface. Furthermore, compatibility between OGC schema and web service tools needs to be addressed. IaaS and PaaS technologies have the potential to improve performance and scalability in real-time monitoring and prediction systems but standard interfaces must be adopted if a long lasting benefit is to be maintained.

Performance constraints in real-time geospatial monitoring and prediction systems are primarily caused by job scheduling bottlenecks and data transfer and data access issues. For jobs with a finite runtime scheduling delays can be mitigated using SLAs. However, continuous running jobs present a problem in this regard as the job lifetime is not known in advance and therefore cannot be made available to the scheduling algorithm. Data access bottlenecks commonly manifest in the form of database deadlocks but can be mitigated by minimising data access through observation aggregation, or by using an eventually consistent data store. Data transfer bottlenecks can be avoided by performing processing close to the data where possible; consequently the use of data centres to store and process data is increasing.

The monitoring and prediction systems implemented in this work enabled important interoperability and performance issues to be highlighted. However, in each implementation the methodology undertaken represents only one of many possible approaches and unsurprisingly certain deficiencies in the implemented systems have been highlighted. It was found that in DSG the single stream per processor approach is not universally applicable, and in the FGSG system, no attempt was made to implement or evaluate the effectiveness of the MPI/ MPP approach. Nonetheless the simple geoprocessing typology set out in this work has on the whole proven to be an effective means of determining the appropriate processing architecture and methodology for real-time geospatial monitoring and prediction applications.

Chapter 8 Conclusion

8.1 Thesis Summary

This research has been driven by technological advancements in two fields; distributed computing and sensor web. The integration of these fields has been shown to benefit geospatial monitoring and prediction systems by enhancing their ability to process and analyse observations in real-time. In Chapter 2 background was provided on the key technological components to this work; namely distributed computing, parallel processing and geospatial web services. In particular, Chapter 2 focuses on how distributed computing technologies can be used to solve issues of scalability and performance in real-time geoprocessing workflows for Earth systems monitoring applications. Within Chapter 2 objectives 1 – 3 are addressed.

Chapter 3 set out the conceptual foundation for this thesis. Within this chapter an attempt was made to classify geoprocessing operations in relation to distributed computing architectures. The classification was intended to facilitate system architects in the design of geospatial monitoring and prediction tools and can be considered a first step towards the development of a generic distributed geoprocessing toolbox. Three distinct classes of geoprocessing operation were identified; DSG, FGSG and CGSG. DSG involves the processing of an unbounded stream of input such as a set of observations from a single sensor through time. Conversely, snapshot geoprocessing refers to the processing of data from one or more sensors at a given instant in time. Snapshot geoprocessing is further subdivided in this classification into coarse-grained and fine-grained operations; coarse-grained operations can be easily parallelised whereas fine-grained operations cannot. The process of developing and evaluating a system representative of each operation type has shown that distributed computing can be integrated with the sensor web and has also highlighted a number of interesting scalability and interoperability issues (Section 7.2 and Section 7.3). Objective 4 is addressed within Chapter 3.

Chapter 4 presents a scalable map-matching (DSG) system that uses ongoing grid compute jobs to process a continuous stream of real-time GPS observations from a fleet of vehicles. This system seamlessly integrates OGC and OGSA web services into a real-time geoprocessing workflow; geoprocessing is carried out on grid compute nodes which are interfaced to OGC services using low-level gridification.

In Chapter 5 a real-time road traffic monitoring system is developed that uses FCD acquired from a fleet of GPS equipped council vehicles to estimate travel-time. Through a web mapping interface this system enables clients to query the fastest route between any two points in the study area based on real-time road traffic information. This provides an example of a FGSG operation in which a parallel relational database is used to perform the bulk of the processing and Amazon EC2 IaaS is used to load test the system.

Chapter 6 explores the use of cloud computing and the MapReduce paradigm to perform a coarse-grained snapshot geoprocessing operation. The operation is an image processing algorithm that reclassifies a satellite image on the basis of the spatial frequency and arrangement of pixels. Collectively Chapters 4-6 address objectives 5 and 6.

Chapter 7 attempts to answer the main research questions of this thesis relating to performance, interoperability and processing methodologies. An attempt is also made to relate the content of this thesis to other work in the field.

In this Chapter the thesis is summarised (Section 8.1) and the main interoperability and architectural recommendations are stated (Section 8.2). Opportunities for future work are presented (Section 8.3) and in Section 8.4 a direction is set out for the future in this field. Objective 7 is addressed by both Chapter 7 and this Chapter.

8.2 Interface and Architectural Recommendations

8.2.1 Improvements to OGC standards

Recommendations for improvements to OGC standards resulting from this thesis are summarised as follows.

1. Extension of WPS specification to accommodate continuous processing operations.

The WPS specification does not currently enable continuous processing operations to be performed and it is assumed that every processing operation has a finite lifespan. In the context of sensor web geoprocessing, the extension of the WPS specification to allow for the management of open ended computational tasks is recommended. Specifically, the inclusion of a *StopExecuting* operation would enable continuous processing tasks to be managed through the WPS interface.

2. Integration of OGC push and pull interfaces for sensor data

The current distinction in the OGC specifications between the push interfaces to sensor data (SAS/SES) and the pull interface (SOS) is unnecessary. Integrating these two service interfaces into a single unified interface would significantly improve and simplify sensor based geoprocessing workflows that conform to these specifications.

3. OGC schema and web service tool compatibility

The lack of interoperability between OGC schema and standard web service tools presents a major barrier to the integration of geospatial web services with distributed computing. To make progress in this field, solving this compatibility problem is crucial.

8.2.2 Architectural Recommendations

The geoprocessing typology developed in this thesis was used to determine the appropriate architecture for each geoprocessing scenario. To process multiple independent streams of geospatial data on the grid (DSG), the one sensor per processor approach was found to be over simplistic. It is recommended that a tool be developed to balance the processing load produced by a sensor array, such that a single sensor data stream can be partitioned amongst multiple processors if necessary, and conversely so multiple sensor data streams can be managed by a single processor as required. In this regard, three distinct sub categories of DSG operations have been identified; atomic transformations, stream dependent transformations and event correlation.

To process snapshots of geospatial data that require little or no synchronisation between sub-processes (CGSG) there are already a plethora of frameworks available to process in parallel, common examples include Condor and Hadoop MapReduce. CGSG geoprocessing operations are well suited to the NOW processing architecture but require data partitioning to be performed which adds complexity to the workflow. It is suggested that a new language is required to express how spatial data be partitioned and reassembled. Inclusion of information on data partitioning and reassembly into WPS specification would enable existing geoprocessing infrastructure and services to be leveraged in a standardised fashion.

To process snapshots of geospatial data that require considerable synchronisation between sub-processes (FGSG) two architectural approaches are suggested; processing within a spatial database and processing using the message passing paradigm on an MPP cluster. In this thesis only the spatial database approach was evaluated. The approach was found to be effective but not universally applicable as the range of geoprocessing operations that can be performed at the database is limited, due to restrictions in the type of data that can be stored in the database, and in the expression of geoprocessing tasks using SQL. It can thus be concluded that the spatial database approach is suitable for the majority of simple fine-grained geoprocessing operations, but for complex modelling it is suggested that the message passing / MPP approach is more suitable.

A summary of the popular combinations of geoprocessing task, processing architecture, parallel strategy, partitioning schema and programming model is displayed in Table 8.1

Table 8.1: Geoprocessing Operations, Architectures and Parallel Strategies

Parent Category	Processing Architecture	Parallel Strategy	Partitioning Strategy	Programming Model /
DSG	MPP	Pipeline	Functional	Any
	NOW	Sensor per processor	Data (by sensor)	Any
	NOW / DSMS	Data stream partitioning	Data (by observation)	Any
CGSG	NOW	Task farm	Data (geometric)	Data parallel
	NOW	Divide & Conquer	Data (geometric)	Any
	NOW	MapReduce	Data + Functional	Data parallel
FGSG	MPP	MPI	Data or Functional	Message passing
	DBMS	De-clustering	Data	SQL

8.3 Future Work

This research has resulted in many questions and topics that are in need of further investigation. Firstly, the implementations in Chapters 4-6 served to highlight some important omissions in the geoprocessing typology developed in Chapter 3. It was found that two of the three identified geoprocessing categories were not finely divided enough to deduce the most appropriate parallel processing architecture. Limitations of the typology are highlighted in Section 7.4 and some refinements are proposed; namely that the DSG category could be further subdivided into atomic transformations, stream-dependent transformations and event correlation to better reflect the possible methods of parallelisation.

Implementing a system from each of the suggested sub-categories is proposed as future work to validate these refinements in the context of the geoprocessing typology.

A major limitation in this study was found to be job scheduling bottlenecks which prevented sensor data streams from being processed in near real-time. Further research into SLAs for near real-time processing is likely to be of considerable benefit to the sensor web geoprocessing community, and to several other communities that have an interest in on-demand near real-time processing.

The WPS-JSDL profiling work carried out by Woolf and Shaon (2009a) adds considerable flexibility to the provisioning of computational resources to geoprocessing tasks. A possible extension to this concept of flexible resource provisioning would be to enable end-users to specify their target grid infrastructure. In the map-matching system the WPS was tightly coupled to an NGS GridSAM endpoint. This tight coupling between the WPS service interface and the grid endpoint negates one of the benefits of low-level gridification in that the end-user is unable to choose the infrastructure on which their job runs. A complete separation between the front end service interface and the back end processing resource would require two significant changes to the system. Firstly, the interface would have to be changed to accommodate an endpoint parameter; for SOAP based WPS this could be achieved by incorporating WS-Addressing into the WPS request. The second issue is the staging of executables from the WPS to the target grid. In the map-matching system this issue was bypassed by storing the executables on a user's home directory from where they could be staged on to the compute node via GridFTP. Were the user allowed to specify their own grid endpoint then a mechanism would be required to stage the executables from the WPS to the endpoint. Currently this presents a problem for low-level gridification as it requires a secure GridFTP server to be running on the non-gridified WPS. An infrastructure agnostic WPS would be an important step towards OGC-OGF interoperability and this concept presents an interesting opportunity for further work.

With regards to OWS and SWE deployment, a further study investigating the deployment of such services onto the cloud is likely to be of value to the research community. Cloud technologies circumvent many of the obstacles presented by gridification as geospatial web services that are deployed on the cloud can be interfaced by the end user in the same way as normal web services so there is no need to set up a web based proxy. As a result, the benefits of high-level gridification (Krüger and Kolbe, 2008) can be realised without the negative aspect of poor interoperability with other geospatial web services. Furthermore, OGC data services backed by eventually consistent databases are likely to alleviate the I/O bottleneck experienced at strongly consistent databases in this systems implemented in thesis. Some work has been carried out on cloud based OWS (Baranski *et al.*, 2009, Liang *et al.*, 2010) but a full ecosystem of geospatial web services in the cloud has not been implemented.

8.3 Future Outlook

The current trends in Earth systems monitoring are being driven by technological improvements. Sensors monitoring the Earth are becoming cheaper, smaller and more plentiful. They are also capturing data at resolutions previously unattainable, resulting in huge amounts of spatial data that must be filtered and analysed. To a large extent these improvements are driven by growing concerns about environmental problems such as climate change. Increasingly complex scientific techniques and Earth systems models now require extremely detailed data from environmental sensors. If this trend continues then the role of distributed computing to share, manage and analyse data will become increasingly important.

At the global and national levels, research in this area is being carried out through Spatial Data Infrastructure (SDI) projects. An SDI is defined as an internet-based mechanism for the coordinated production, discovery, and use of geospatial information in a digital environment (Budhathoki *et al.*, 2008). European funded SDI projects such as GDI-GRID, ORCHESTRA, GEOSS, INSPIRE and GMES are attempting to put in place a set of core services to facilitate Earth systems monitoring. Higher level applications can be rapidly and dynamically composed from these core services for all kinds of purposes. As these projects operate at the global and national level, a large amount of data is involved as well as a potentially huge numbers of users. Consequently, distributed computing is an important element that is required to ensure the success of these projects.

As we have seen in the work carried out in this thesis, distributed computing is also necessary for Earth systems monitoring at the regional level. Traffic monitoring applications and remotely sensed image processing can benefit from distributed computing to improve processing time performance, and scalability in terms of the amount of data that can be processed by a given system.

To achieve large scale Earth systems monitoring it seems that there are few alternatives to using distributed computing for geoprocessing. Reducing the scale or depth of analysis, or running compute jobs for long time periods are not sustainable solutions. Supercomputing is a valid alternative to distributed computing for large-scale geoprocessing, but it cannot compete in terms of cost performance (Abbas, 2004). Additionally, supercomputing can in many cases be accessed via distributed systems. For example, grid infrastructures such as the NGS offer supercomputing services, and cloud platforms such as AWS offer a platform from which MPI jobs can be run. Consequently, improving the ease with which parallel geoprocessing can be interfaced is required, if our Earth systems monitoring programmes are to be sustained in the long term.

Barriers to integration of geospatial web services with distributed computing are beginning to diminish as research into standards harmonisation continues. The work presented in this thesis has made a contribution to the field by defining a categorisation of real-time geoprocessing, and showing how these different types of geoprocessing can be achieved in a distributed computing environment, through a set of real world examples. In the future it seems likely that the cloud architecture will see widespread adoption in the geospatial field. Unlike the grid, which has been driven by academia, the cloud model is driven by commercially driven IT companies. Consequently software providers are likely to turn to this infrastructure because it is well supported and relatively simple to use. Cloud computing is a combination of grid computing, utility computing and virtualisation technology. It therefore incorporates the advantages of grid computing, but it also provides a model that enables infrastructure providers and service providers to benefit from economies of scale, which is likely to prove successful in the commercial environment, as well as in academia.

Appendix A Sensor Observation Service (SOS) Build & Deployment Steps

Prerequisites

1) Java Development Kit (JDK) version 1.5 or higher
   http://java.sun.com/javase/downloads

2) Working installation of Apache Maven
   http://maven.apache.org/ (v2.2.1)

3) Working installation of PostgreSQL database management system
   http://www.postgresql.org/ (v8.3)

4) PostGIS spatial extension to PostgreSQL
   http://postgis.refractions.net/ (v1.5)

5) Working installation of Apache Tomcat (v6.0)
   http://tomcat.apache.org/

6) Subversion client, e.g. TortoiseSVN
http://tortoisesvn.tigris.org/

Procedure

The following steps were followed to build, configure and deploy the 52 North SOS. *Note: This procedure assumes a windows platform although this is not strictly required*

1) Create a Spatial Database
   a. Open the PostgreSQL PgAdmin console
   b. Click on the local database server in the right hand pane and connect.
   c. Right click on the server and select ‘new database’
   d. Name the database ‘SOS’ and select ‘template_postgis’ from the template dropdown box.

2) Configure Maven

a. Open the *conf* folder in the Maven install directory and edit the *settings.xml* file

b. Under the profile tag insert the following profile:

```
<profile>
    <id>52n-start</id>
        <repositories>
            <repository>
                <id>n52-releases</id>
                <name>52n Releases</name>
                <url>http://52north.org/maven/repo/releases</url>
                <releases>
                    <enabled>true</enabled>
                </releases>
                <snapshots>
                    <enabled>false</enabled>
                </snapshots>
            </repository>
            <repository>
                <id>geotools</id>
                <name>Geotools repository</name>
                <url>http://maven.geotools.fr/repository</url>
            </repository>
            <repository>
                <id>Refractions</id>
                <name>Refractions repository</name>
                <url>http://lists.refractions.net/m2</url>
            </repository>
            <repository>
                <id>Apache</id>
                <name>Apache repository</name>
                <url>http://repo1.maven.org/maven2</url>
            </repository>
    </repositories>
</profile>
```

c. After the profiles section insert the following active profile:

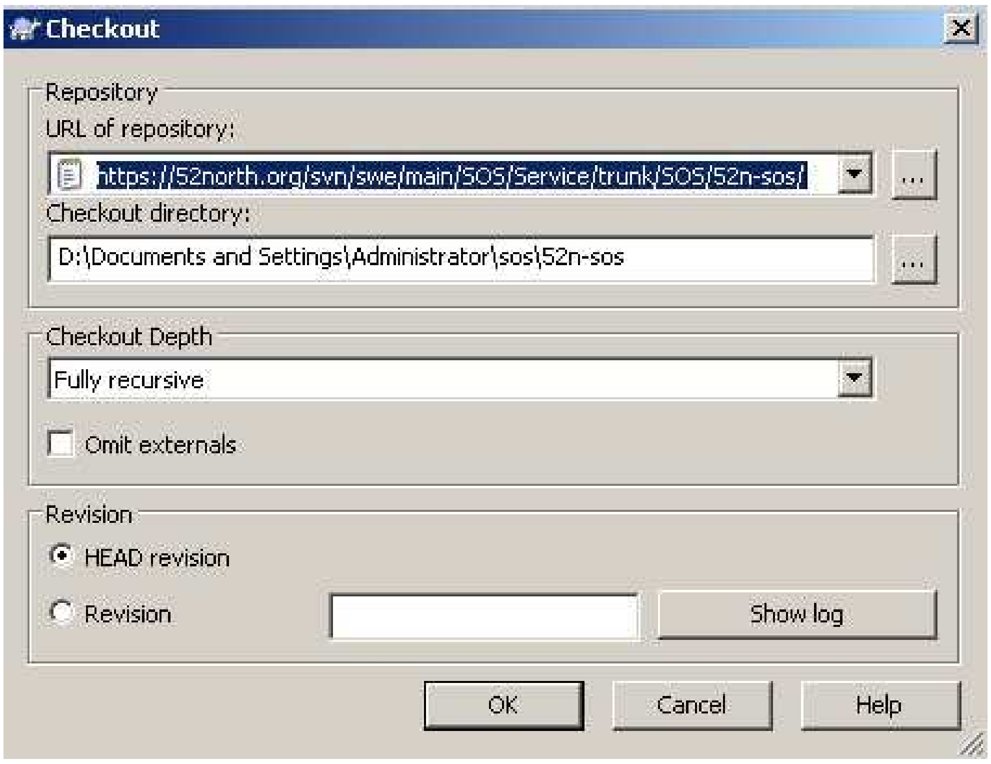
```
<activeProfiles>
        <activeProfile>52n-start</activeProfile>
  </activeProfiles>
```

3) Checkout the project from SVN:

a. Create a windows folder in which to install the sources

b. Right click in this folder and select SVN Checkout from the popup menu

c. Checkout the project by filling out the SVN URL *http://52north.org/svn/swe/main/SOS/service/trunk/SOS* and the destination directory for the source code.



4) Configure the Maven project object model: *pom.xml*

Once the sources have finished downloading open the pom.xml in a text editor and make the following changes:

a. The name of your SOS webapp

```
<conf.sos.name>52nSOSv3</conf.sos.name>
```

b. The public IP of your SOS webapp

```
<deploy.target.host>128.240.60.30</deploy.target.host>
```

c. The port

```
<deploy.target.port>9090</deploy.target.port>
```

d. Your tomcat manager connection settings

```
<!-- Tomcat Manager username **HAS TO BE CHANGED**-->
<deploy.tomcat.manager.username>admin</deploy.tomcat.manager.username>
<!-- Tomcat Manager password **HAS TO BE CHANGED**-->
<deploy.tomcat.manager.password>******</deploy.tomcat.manager.password>
<!--installation directory of the tomcat servlet engine **HAS TO BE CHANGED, IF NECESSARY**-->
<deploy.tomcat.home>C:/Programme/Apache Software Foundation/Tomcat 6.0</deploy.tomcat.home>
```

e. The connection settings for the database created in Step 1.

```
<!--connectionstring to the DB **HAS TO BE CHANGED** -->
```

```
          <conf.sos.ds.connectionstring>
                    jdbc:postgresql://localhost:5432/SOS
          </conf.sos.ds.connectionstring>
<!-- your DB-username **HAS TO BE CHANGED** -->
          <conf.sos.ds.user>aengus</conf.sos.ds.user>
<!-- your DB-password **HAS TO BE CHANGED** -->
          <conf.sos.ds.password>a4131673</conf.sos.ds.password>
```

f. The request decoders

```
<conf.sos.postRequestDecoder>
          org.n52.sos.decode.impl.HttpPostRequestDecoderMobile
</conf.sos.postRequestDecoder>
<conf.sos.getRequestDecoder>
          org.n52.sos.decode.impl.HttpGetRequestDecoderMobile
</conf.sos.getRequestDecoder>
```

g. Mobile enabled

```
<conf.sos.mobileEnabled>true</conf.sos.mobileEnabled>
```

h. Capabilities settings

```
<conf.sos.capabilities.provider.name>
          Newcastle University
</conf.sos.capabilities.provider.name>
<conf.sos.capabilities.provider.site>
          http://ceg.ncl.ac.uk
</conf.sos.capabilities.provider.site>
<conf.sos.capabilities.provider.individual.name>
     Mr Aengus McCullough </conf.sos.capabilities.provider.individual.name>
<conf.sos.capabilities.provider.position.name>
          PhD Student
</conf.sos.capabilities.provider.position.name>
          <conf.sos.capabilities.provider.phone>
+44(0)777777777
</conf.sos.capabilities.provider.phone>
<conf.sos.capabilities.provider.address>
Cassie Building, Claremont Rd
</conf.sos.capabilities.provider.address>
<conf.sos.capabilities.provider.city>
Newcastle upon Tyne
</conf.sos.capabilities.provider.city>
<conf.sos.capabilities.provider.zip>
NE17RU
</conf.sos.capabilities.provider.zip>
<conf.sos.capabilities.provider.state>
```

```
Tyne and Wear
</conf.sos.capabilities.provider.state>
<conf.sos.capabilities.provider.country>
        UK
</conf.sos.capabilities.provider.country>
<conf.sos.capabilities.provider.email>aengus.mccullough@ncl.ac.uk</conf.sos.capabiliti
es.provider.email>
```

i. URL of the web application

```
<conf.sos.service.url>
        http://128.240.60.30:9090/${conf.sos.name}
</conf.sos.service.url>
```

5) Populate the SOS database with the SOS data structure
    a. Open PgAdmin and the SOS database and select the toolbar button 'execute SQL query'.
    b. Open the SQL file located in the SOS
    sourcedir/db/datamodel_postgres83.sql
    c. Execute the SQL and exit PgAdmin

6) Adjust the capabilities skeleton mobile and the SensorML skeleton
    a. Browse to the */52n-sos-service/src/main/webapp/WEB-INF/conf/* folder and modify the */capabilities/capabilities_skeleton_mobile* file to describe the SOS capabilities.
    b. Next, for each vehicle in the fleet insert a SensorML description of the sensor into the */sensors* folder, named *sensor-name.xml* where sensor-name is the name given to the sensor in the SensorML file.

7) Start Tomcat
    a. If Tomcat is installed as a service then open the services manager as follows:
        i. Click on Start >> Run and type *services.msc*
        ii. Select the Tomcat Service and right click *start service*

    Otherwise browse to the Tomcat installation folder and select */bin/startup.bat*

8) Build and deploy the SOS
    a. Open a windows command prompt Start >>Run and type *cmd*
    b. Browseto the root source directory of the SOS project that was checked out and type *mvn –Pwith-deploy install*

9) Restart Tomcat, the SOS should now be available at the path specified as URL of web application (Step 4).
10) Consult the 52 North SOS documentation for troubleshooting advice.

Appendix B Loading GPS Observations into PostGIS

Prerequisites

1) A working installation of PostgreSQL database management system http://www.postgresql.org/ (v8.2) with the PostGIS spatial extension http://postgis.refractions.net/ (v1.4.2)
2) Microsoft Excel 2003

Procedure

Note: This procedure assumes a windows platform although this is not strictly required

11) Create a Spatial Database
    a. Open the PostgreSQL PgAdmin console
    b. Click on the local database server in the right hand pane and connect.
    c. Right click on the server and select ‘new database’
    d. Name the database ‘SOS’ and select ‘template_postgis’ from the template dropdown box.

12) Write SQL:

```
CREATE TABLE observations(id INTEGER, time TIMESTAMPTZ,
elevation FLOAT(5), geometry VARCHAR(50));
SET Datestyle = ‘DMY’;
```

13) Prepare GPS observation file

Open the text file containing the observations in Microsoft Excel and ensure that the order of columns returned by the database query: `SELECT * FROM observations` matches the column order in the text file. If there is a header row then remove it. Save as a tab delimited text file with a .txt extension.

14) Fix bug in Proj4 library

If using Proj4 library < 1.6.2 (postgis 1.4.2) then need to fix a bug in the ‘spatial_ref_sys’ table. If the query ‘`SELECT proj4text FROM spatial_ref_sys WHERE srid=27700`’ returns: ‘`+proj=tmerc +lat_0=49 +lon_0=-2 +k=0.999601 +x_0=400000 +y_0=-`

`100000  +ellps=airy +units=m +no_defs’` then it is necessary to append ‘`datum=OSGB36`’ to the string to ensure correct reprojections.

```
UPDATE  spatial_ref_sys  SET  proj4text=  ‘+proj=tmerc  +lat_0=49
+lon_0=-2  +k=0.999601  +x_0=400000  +y_0=-100000    +ellps=airy
+units=m +no_defs +datum=OSGB36’ WHERE srid=27700;
UPDATE   spatial_ref_sys   SET   proj4text   =   '+proj=longlat
+ellps=airy +datum=OSGB36 +no_defs' WHERE srid=4277;
```

5) It is now possible to correctly transform the coordinate system from wgs84(EPSG:4326) to osgb36 (EPSG:27700)

```
UPDATE observations SET osgb_geom = transform(wgs_geom,27700);
```

Appendix C Publishing Ordnance Survey MasterMap Integrated Transport Network (ITN) data with Geoserver

Prerequisites

1) A Java Development Kit (JDK) version 1.5 or higher

   http://java.sun.com/javase/downloads

2) A working installation of Geoserver

   http://geoserver.org/ (v2.0)

3) A working installation of PostgreSQL

http://www.postgresql.org/ (v8.2)

4) The PostGIS spatial extension

   http://postgis.refractions.net/ (v1.4.2)

5) An InterpOSe installation

   http://www.dottedeyes.com/spatial_data_loading/interpose/

Procedure

Note: This procedure assumes a windows platform although this is not strictly required

1) Download Master-Map ITN Data from *http://www.edina.ac.uk/digimap*
   - *a. Ordnance Survey Collection >> Data Download Service>> MasterMap Download>>ITN layer (road network)*
   - b. Select GZip GML download format

2) Load GML data into PostGIS database
   - a. Create a database to store the data. From the postgres shell type the command: *createdb –T template_postgis <yourDBname>* and enter your password. If using PostGIS version <1.5 then follow Step 4 in Appendix B to correct the proj4 library.
   - b. Open InterpOSe and follow the wizard based instructions to convert the downloaded Mastermap data in GML format, to ESRI Shapefile format. Note that although it is possible to load GML directly into PostGIS using OGR2OGR this method does not preserve the unique TOID identifier of

each feature, thus it is necessary to first convert it to shapefile using the interpose tool.

c. Use the PostGIS shp2pgsql tool to load the data in ESRI shapefile format into PostGIS. From the postgres shell type the following for each shapefile layer, i.e. roadlink_polyline and roadnode_point. Substitute in the parameters for the database created above:

```
d:\Program Files\postgresql\8.2\bin>shp2pgsql -s 27700 -d -g
the_geom  d:\data\ITN\Road_Link_polyline  public.roadLink  |
psql -h localhost -d itn_test -U Administrator
```

d. Ensure the data has loaded correctly. From the postgres shell, connect to the db: `psql` <myDBName> and check it has tables*:* `show columns from table \d`

3) Publish the data as a WFS using Geoserver

a. Start GeoServer and point browser at http://localhost:8080/geoserver

Click on *config* >> *Data* >>*Namespace*>> *New* and type *osgb* as the prefix and http://www.ordnancesurvey.co.uk/xml/namespaces/osgb as the URI.

b. Click on *Data* >> *DataStores* >>*New* and select *Postgis* from the Feature Data Set Description. Type *MasterMap_ITN* as the feature dataset ID and click *New*.

c. Select *osgb* as the namespace and fill out the connection details for <*yourDBname*>. Click on submit and in the left hand panel click *Apply* >> *Save*>> *Load.*

d. Click on *Data* >> *FeatureTypes* >> *New* and select from the Feature_Type_Name drop down <yourDBname>..roadlink and click *New.*

e. From the following page select a display style that is appropriate to the geometry, '*simple_roads'* for example. Click on '*Lookup SRS'* and 'Generate Bounding Box', check 'Enable caching' and click submit and *Apply* >> *Save*>> *Load.*

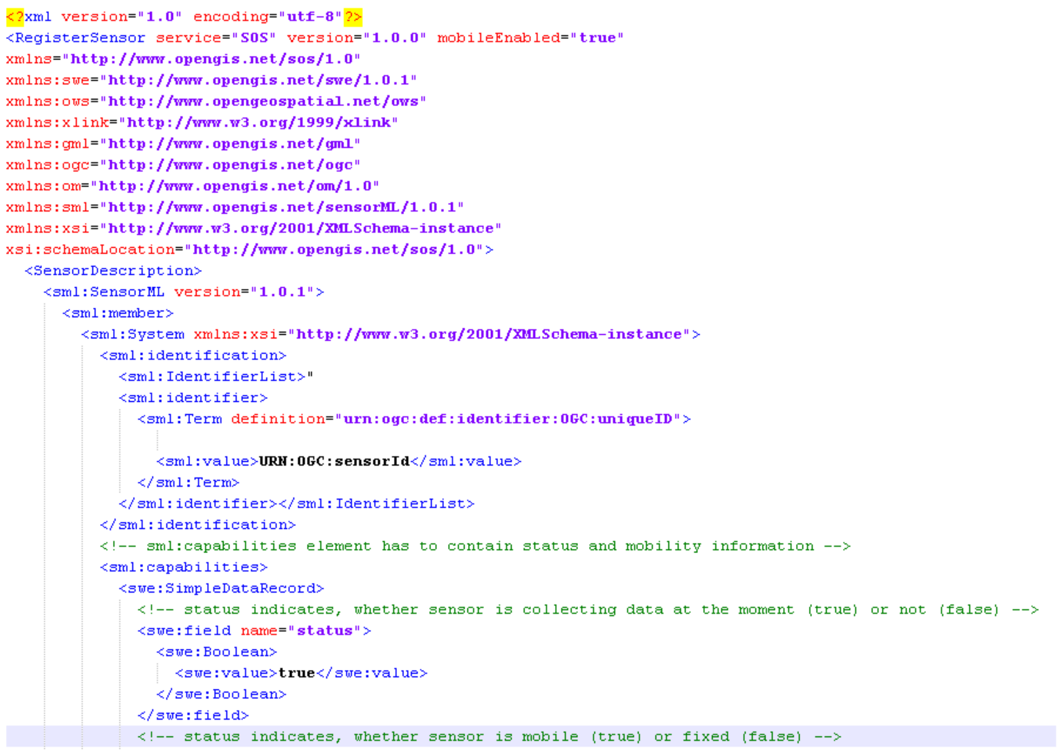
f. Once a feature type has been created it should be possible to view the data by clicking on Welcome >> Demo >> Map Preview >> <yourFeatureTypeName> The layer should be visible.

g. Send a test GetCapabilities request to the WFS: http://localhost:8080/geoserver/wfs?request=getCapabilities&service=WFS &version=1.0.0

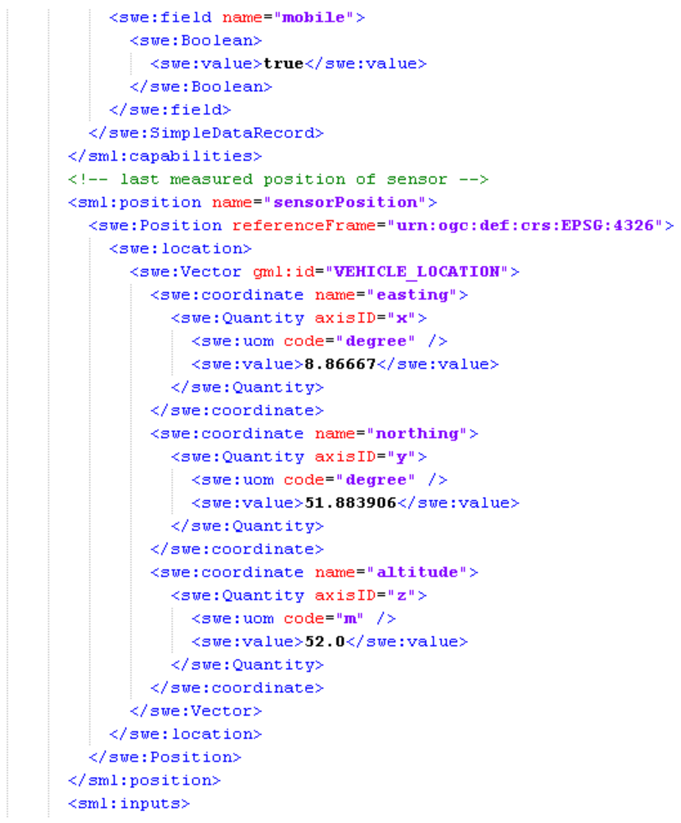
h. Finally, the WFS can be ported to a publicly accessible server once it is successfully running locally.

Appendix D Example Requests and Responses

SOS *RegisterSensor* Request



```
<?xml version="1.0" encoding="utf-8"?>
<RegisterSensor service="SOS" version="1.0.0" mobileEnabled="true"
xmlns="http://www.opengis.net/sos/1.0"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:om="http://www.opengis.net/om/1.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sos/1.0">
  <SensorDescription>
    <sml:SensorML version="1.0.1">
      <sml:member>
        <sml:System xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <sml:identification>
            <sml:IdentifierList>"
            <sml:identifier>
              <sml:Term definition="urn:ogc:def:identifier:OGC:uniqueID">

                <sml:value>URN:OGC:sensorId</sml:value>
              </sml:Term>
            </sml:identifier></sml:IdentifierList>
          </sml:identification>
          <!-- sml:capabilities element has to contain status and mobility information -->
          <sml:capabilities>
            <swe:SimpleDataRecord>
              <!-- status indicates, whether sensor is collecting data at the moment (true) or not (false) -->
              <swe:field name="status">
                <swe:Boolean>
                  <swe:value>true</swe:value>
                </swe:Boolean>
              </swe:field>
              <!-- status indicates, whether sensor is mobile (true) or fixed (false) -->
```



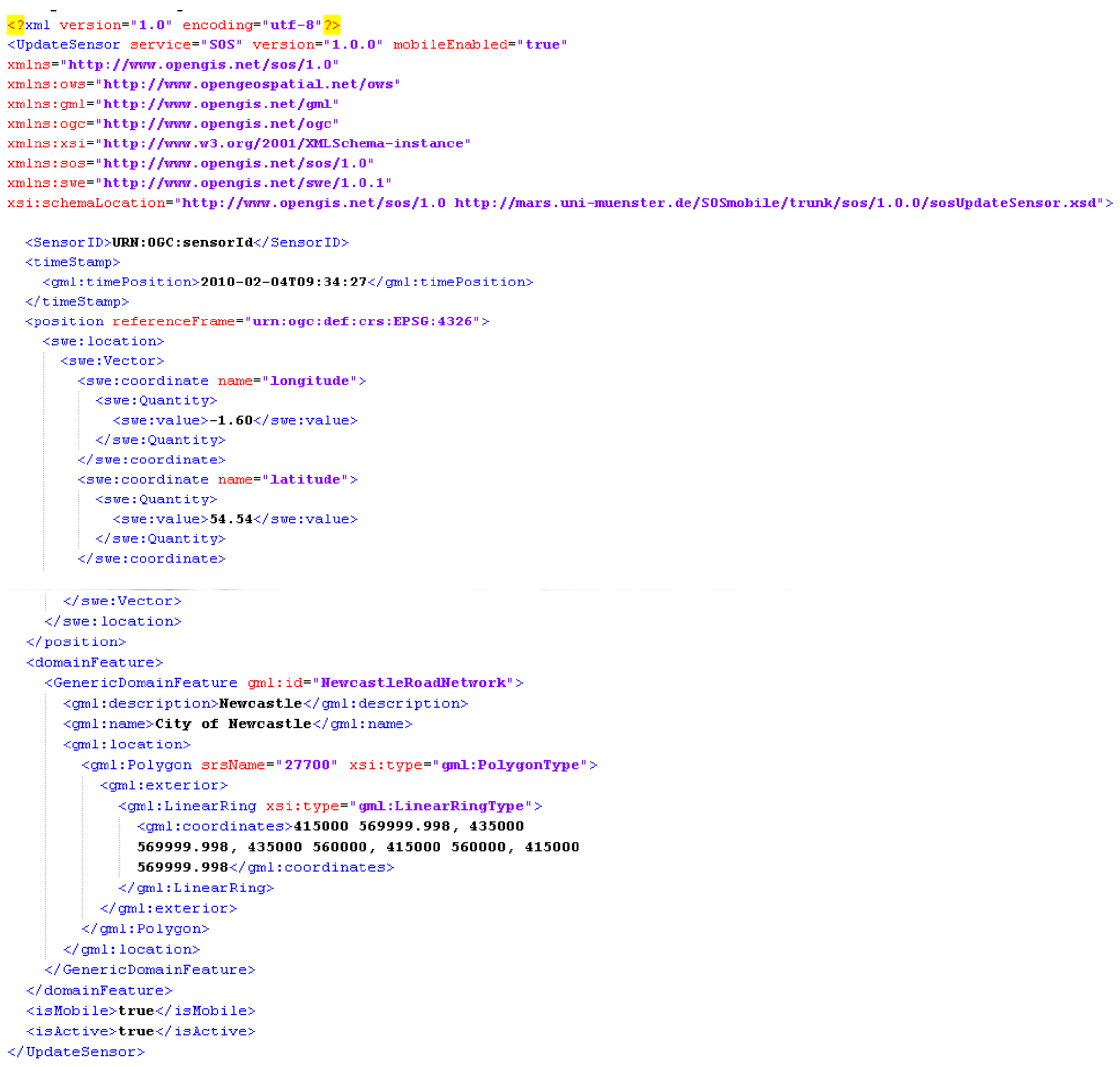
```
              <swe:field name="mobile">
                <swe:Boolean>
                  <swe:value>true</swe:value>
                </swe:Boolean>
              </swe:field>
            </swe:SimpleDataRecord>
          </sml:capabilities>
          <!-- last measured position of sensor -->
          <sml:position name="sensorPosition">
            <swe:Position referenceFrame="urn:ogc:def:crs:EPSG:4326">
              <swe:location>
                <swe:Vector gml:id="VEHICLE_LOCATION">
                  <swe:coordinate name="easting">
                    <swe:Quantity axisID="x">
                      <swe:uom code="degree" />
                      <swe:value>8.86667</swe:value>
                    </swe:Quantity>
                  </swe:coordinate>
                  <swe:coordinate name="northing">
                    <swe:Quantity axisID="y">
                      <swe:uom code="degree" />
                      <swe:value>51.883906</swe:value>
                    </swe:Quantity>
                  </swe:coordinate>
                  <swe:coordinate name="altitude">
                    <swe:Quantity axisID="z">
                      <swe:uom code="m" />
                      <swe:value>52.0</swe:value>
                    </swe:Quantity>
                  </swe:coordinate>
                </swe:Vector>
              </swe:location>
            </swe:Position>
          </sml:position>
          <sml:inputs>
```

```
            <sml:InputList>
              <sml:input name="pseudorange">
                <swe:ObservableProperty definition="urn:ogc:def:phenomenon:OGC:1.0.30:pseudorange" />
              </sml:input>
            </sml:InputList>
          </sml:inputs>
          <sml:outputs>
            <sml:OutputList>
              <sml:output name="bearing">
                <swe:Quantity definition="urn:ogc:def:phenomenon:OGC:1.0.30:bearing">
                  <gml:metaDataProperty>
                    <offering>
                      <id>BEARING</id>
                      <name>Whole Circle Bearing</name>
                    </offering>
                  </gml:metaDataProperty>
                  <swe:uom code="rad" />
                </swe:Quantity>
              </sml:output>
              <sml:output name="RoadID">
                <swe:Text definition="urn:ogc:def:phenomenon:OGC:1.0.30:roadID">
                  <gml:metaDataProperty>
                    <offering>
                      <id>roadID</id>
                      <name>TOID of road section</name>
                    </offering>
                  </gml:metaDataProperty>
                </swe:Text>
              </sml:output>
            </sml:OutputList>
          </sml:outputs>
        </sml:System>
      </sml:member>
    </sml:SensorML>
  </SensorDescription>
  <ObservationTemplate>
    <om:Measurement>
      <om:samplingTime />
      <om:procedure />
      <om:observedProperty />
      <om:featureOfInterest></om:featureOfInterest>
      <om:result uom=""></om:result>
    </om:Measurement>
  </ObservationTemplate>
  <ObservationTemplate>
    <om:CategoryObservation>
      <om:samplingTime />
      <om:procedure />
      <om:observedProperty />
      <om:featureOfInterest></om:featureOfInterest>
      <om:result></om:result>
    </om:CategoryObservation>
  </ObservationTemplate>
  <domainFeature>
    <GenericDomainFeature gml:id="NewcastleRoadNetwork">
      <gml:description>Newcastle</gml:description>
      <gml:name>City of Newcastle</gml:name>
      <gml:location>
        <gml:Polygon srsName="27700" xsi:type="gml:PolygonType">
          <gml:exterior>
            <gml:LinearRing xsi:type="gml:LinearRingType">
              <gml:coordinates>415000 569999.998, 435000
              569999.998, 435000 560000, 415000 560000, 415000
              569999.998</gml:coordinates>
            </gml:LinearRing>
          </gml:exterior>
        </gml:Polygon>
      </gml:location>
    </GenericDomainFeature>
  </domainFeature>

</RegisterSensor>
```

SOS *RegisterSensor* Response

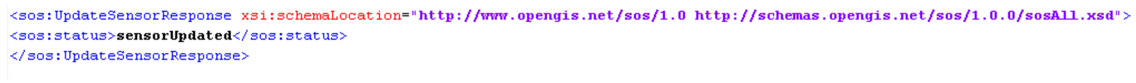
```
<sos:RegisterSensorResponse xsi:schemaLocation="http://www.opengis.net/sos/1.0 http://schemas.opengis.net/sos/1.0.0/sosAll.xsd">
<sos:AssignedSensorId>URN:OGC:sensorId</sos:AssignedSensorId>
</sos:RegisterSensorResponse>
```

SOS *UpdateSensor* Request



```
<?xml version="1.0" encoding="utf-8"?>
<UpdateSensor service="SOS" version="1.0.0" mobileEnabled="true"
xmlns="http://www.opengis.net/sos/1.0"
xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:gml="http://www.opengis.net/gml"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sos="http://www.opengis.net/sos/1.0"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xsi:schemaLocation="http://www.opengis.net/sos/1.0 http://mars.uni-muenster.de/SOSmobile/trunk/sos/1.0.0/sosUpdateSensor.xsd">

  <SensorID>URN:OGC:sensorId</SensorID>
  <timeStamp>
    <gml:timePosition>2010-02-04T09:34:27</gml:timePosition>
  </timeStamp>
  <position referenceFrame="urn:ogc:def:crs:EPSG:4326">
    <swe:location>
      <swe:Vector>
        <swe:coordinate name="longitude">
          <swe:Quantity>
            <swe:value>-1.60</swe:value>
          </swe:Quantity>
        </swe:coordinate>
        <swe:coordinate name="latitude">
          <swe:Quantity>
            <swe:value>54.54</swe:value>
          </swe:Quantity>
        </swe:coordinate>
      </swe:Vector>
    </swe:location>
  </position>
  <domainFeature>
    <GenericDomainFeature gml:id="NewcastleRoadNetwork">
      <gml:description>Newcastle</gml:description>
      <gml:name>City of Newcastle</gml:name>
      <gml:location>
        <gml:Polygon srsName="27700" xsi:type="gml:PolygonType">
          <gml:exterior>
            <gml:LinearRing xsi:type="gml:LinearRingType">
              <gml:coordinates>415000 569999.998, 435000
              569999.998, 435000 560000, 415000 560000, 415000
              569999.998</gml:coordinates>
            </gml:LinearRing>
          </gml:exterior>
        </gml:Polygon>
      </gml:location>
    </GenericDomainFeature>
  </domainFeature>
  <isMobile>true</isMobile>
  <isActive>true</isActive>
</UpdateSensor>
```

SOS *UpdateSensor* Response



```
<sos:UpdateSensorResponse xsi:schemaLocation="http://www.opengis.net/sos/1.0 http://schemas.opengis.net/sos/1.0.0/sosAll.xsd">
<sos:status>sensorUpdated</sos:status>
</sos:UpdateSensorResponse>
```

SOS *InsertCategoryObservation* Request

```xml
<?xml version="1.0" encoding="utf-8"?>
<InsertObservation xmlns="http://www.opengis.net/sos/1.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:om="http://www.opengis.net/om/1.0"
xmlns:sos="http://www.opengis.net/sos/1.0"
xmlns:sa="http://www.opengis.net/sampling/1.0"
xmlns:n52="http://www.52north.org/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sos/1.0 http://schemas.opengis.net/sos/1.0.0/sosInsert.xsd
http://www.opengis.net/sampling/1.0 http://schemas.opengis.net/sampling/1.0.0/sampling.xsd
http://www.opengis.net/om/1.0 http://schemas.opengis.net/om/1.0.0/extensions/observationSpecialization_override.xsd"
service="SOS" version="1.0.0">
  <AssignedSensorId>
  urn:ogc:object:feature:Sensor:NCL:ncl-sensor-1</AssignedSensorId>
  <om:CategoryObservation>
    <om:samplingTime>
      <gml:TimeInstant>2007-05-16T19:44:19+0000</gml:TimeInstant>
    </om:samplingTime>
    <om:procedure xlink:href="urn:ogc:object:feature:Sensor:NCL:ncl-sensor-1" />
    <om:observedProperty xlink:href="urn:ogc:def:phenomenon:OGC:1.0.30:roadID" />
    <om:featureOfInterest>
      <!-- a sampling feature is needed to insert CategoryObservations -->
      <sa:SamplingPoint gml:id="Position2">
        <gml:name>Position2</gml:name>
        <sa:sampledFeature xlink:href="" />
        <sa:position>
          <gml:Point>
            <gml:pos srsName="urn:ogc:def:crs:EPSG:4326">54.98709
            -1.59405</gml:pos>
          </gml:Point>
        </sa:position>
      </sa:SamplingPoint>
    </om:featureOfInterest>
    <om:result>itn_10_features.13472</om:result>
  </om:CategoryObservation>
</InsertObservation>
```

SOS *InsertObservation* Response

```xml
<sos:InsertObservationResponse xmlns:sos="http://www.opengis.net/sos/1.0">
  <sos:AssignedObservationId>o_-2147483648</sos:AssignedObservationId>
</sos:InsertObservationResponse>
```

SOS *InsertMeasurement* Request

```xml
<InsertObservation
service=\"SOS\"
version=\"1.0.0\"
mobileEnabled=\"true\"
xmlns=\"http://www.opengis.net/sos/1.0\"
xmlns:ows=\"http://www.opengis.net/ows/1.1\"
xmlns:ogc=\"http://www.opengis.net/ogc\"
xmlns:om=\"http://www.opengis.net/om/1.0\"
xmlns:sos=\"http://www.opengis.net/sos/1.0\"
xmlns:sa=\"http://www.opengis.net/sampling/1.0\"
xmlns:gml=\"http://www.opengis.net/gml\"
xmlns:swe=\"http://www.opengis.net/swe/1.0.1\"
xmlns:xlink=\"http://www.w3.org/1999/xlink\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:schemaLocation=\"http://www.opengis.net/sos/1.0
http://mars.uni-muenster.de/sosmobile/trunk/sos/1.0.0/sosInsert.xsd
http://www.opengis.net/sampling/1.0
http://schemas.opengis.net/sampling/1.0.0/sampling.xsd
http://www.opengis.net/om/1.0
http://schemas.opengis.net/om/1.0.0/extensions/observationSpecialization_override.xsd\" >
```

```
<AssignedSensorId>urn:ogc:object:feature:Sensor:NCL:ncl-sensor-7</AssignedSensorId>
<om:Measurement>
     <om:samplingTime>
          <gml:TimeInstant>
               <gml:timePosition>2008-04-01T17:44:15+00</gml:timePosition>
          </gml:TimeInstant>
     </om:samplingTime>
     <om:procedure xlink:href=\"urn:ogc:object:feature:Sensor:NCL:ncl-sensor-1\"/>
     <om:observedProperty xlink:href=\"urn:ogc:def:phenomenon:OGC:1.0.30:bearing\"/>
     <om:domainFeature>
       <GenericDomainFeature gml:id=\"NewcastleRoadNetwork\">
        <gml:description>Newcastle</gml:description>
        <gml:name>City of Newcastle</gml:name>
        <gml:location>
          <gml:Polygon srsName=\"4326\" xsi:type=\"gml:PolygonType\">
            <gml:exterior>
                <gml:LinearRing xsi:type=\"gml:LinearRingType\">
                    <gml:coordinates>8.76667 51.7167, 8.76667 52.7167, 9.76667 52.7167, 9.76667 51.7167, 8.76667 51.7167</gml:coordinates>
                </gml:LinearRing>
            </gml:exterior>
          </gml:Polygon>
        </gml:location>
       </GenericDomainFeature>
     </om:domainFeature>
     <om:featureOfInterest>
          <sa:SamplingPoint gml:id=\"Position1\">
              <gml:name>FOI_EINS</gml:name>
              <sa:sampledFeature xlink:href=\"\"/>
              <sa:position>
                <gml:Point>
                  <gml:pos srsName=\"urn:ogc:def:crs:EPSG:4326\">8.86667 51.883906</gml:pos>
                </gml:Point>
              </sa:position>
            </sa:SamplingPoint>
     </om:featureOfInterest>
     <om:result uom=\"rad\">2.0</om:result>
</om:Measurement>
</InsertObservation>
```

SOS *DescribeSensorTimePeriod* Request

```
<?xml version="1.0" encoding="UTF-8"?>
<DescribeSensor version="1.0.0" service="SOS" mobileEnabled="true" xmlns="http://www.opengis.net/sos/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/sos/1.0
http://schemas.opengis.net/sos/1.0.0/sosDescribeSensor.xsd" outputFormat="text/xml;subtype=&quot;sensorML/1.0.1&quot;">
    <procedure>testgpsobs3</procedure>
    <time>
        <ogc:TM_During>
            <ogc:PropertyName>om:samplingTime</ogc:PropertyName>
            <gml:TimePeriod>
                <gml:beginPosition>2010-02-11T14:03:35+00</gml:beginPosition>
                <gml:endPosition>2010-02-11T14:04:35+00</gml:endPosition>
            </gml:TimePeriod>
        </ogc:TM_During>
    </time>
</DescribeSensor>
```

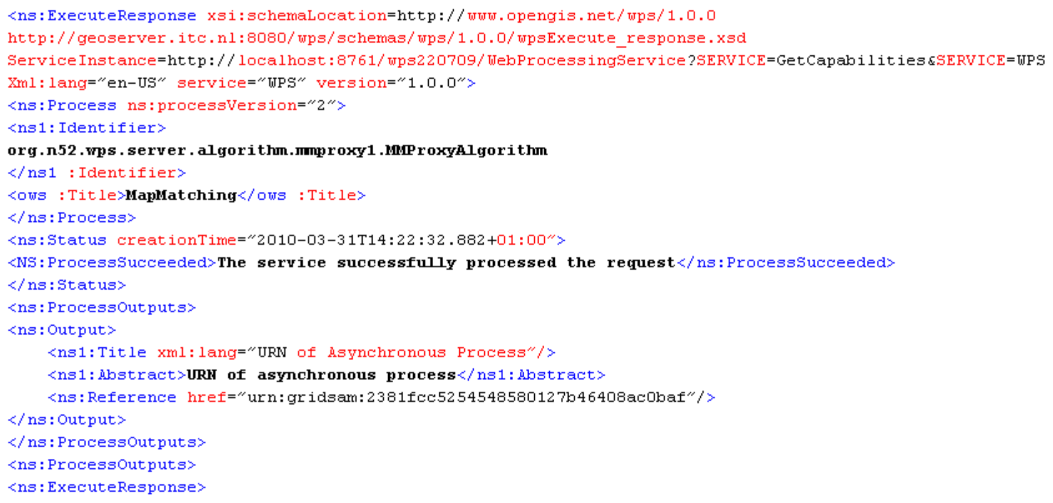
WFS GetFeature Request

```
<wfs:GetFeature service="WFS" version="1.1.0"
  xmlns:osgb="http://www.ordnancesurvey.co.uk/xml/namespaces"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wfs
                      http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
  <wfs:Query typeName="topp:states">
    <wfs:PropertyName>osgb:roadlink</wfs:PropertyName>
    <ogc:Filter>
      <ogc:BBOX>
        <ogc:PropertyName>the_geom</ogc:PropertyName>
        <gml:Envelope srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
           <gml:lowerCorner>-75.102613 40.212597</gml:lowerCorner>
           <gml:upperCorner>-72.361859 41.512517</gml:upperCorner>
        </gml:Envelope>
      </ogc:BBOX>
   </ogc:Filter>
  </wfs:Query>
</wfs:GetFeature>
```

WPS *Execute* (HTTP GET) Request

```
http://localhost:8080/wps/WebProcessingService?request=execute&DataInp
uts=SOS_URL=http://128.240.60.1:9091/52nSOSv3/sos;SENSOR_ID=gpsobs3;WF
S_URL=http://128.240.60.30:8762/geoserver/wfs;WFS_NS=osgb;WFS_TN=roadl
ink&version=1.0&Identifier=org.n52.wps.server.algorithm.mmproxy1.MMPro
xyAlgorithm
```

WPS *Execute* Response

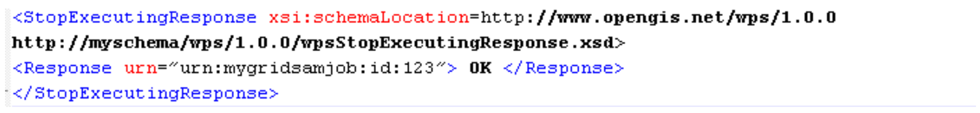


```
<ns:ExecuteResponse xsi:schemaLocation=http://www.opengis.net/wps/1.0.0
http://geoserver.itc.nl:8080/wps/schemas/wps/1.0.0/wpsExecute_response.xsd
ServiceInstance=http://localhost:8761/wps220709/WebProcessingService?SERVICE=GetCapabilities&SERVICE=WPS
Xml:lang="en-US" service="WPS" version="1.0.0">
<ns:Process ns:processVersion="2">
<ns1:Identifier>
org.n52.wps.server.algorithm.mmproxy1.MMProxyAlgorithm
</ns1 :Identifier>
<ows :Title>MapMatching</ows :Title>
</ns:Process>
<ns:Status creationTime="2010-03-31T14:22:32.882+01:00">
<NS:ProcessSucceeded>The service successfully processed the request</ns:ProcessSucceeded>
</ns:Status>
<ns:ProcessOutputs>
<ns:Output>
    <ns1:Title xml:lang="URN of Asynchronous Process"/>
    <ns1:Abstract>URN of asynchronous process</ns1:Abstract>
    <ns:Reference href="urn:gridsam:2381fcc5254548580127b46408ac0baf"/>
</ns:Output>
</ns:ProcessOutputs>
<ns:ProcessOutputs>
<ns:ExecuteResponse>
```

WPS *StopExecuting* (HTTP GET) Request

```
http://128.240.60.1/wps?request=stopExecuting&service=wps&version=1.0.
0& job_id=urn:mygridsamjob:id:123
```

WPS *StopExecuting* Response



```
<StopExecutingResponse xsi:schemaLocation=http://www.opengis.net/wps/1.0.0
http://myschema/wps/1.0.0/wpsStopExecutingResponse.xsd>
<Response urn="urn:mygridsamjob:id:123"> OK </Response>
</StopExecutingResponse>
```

Appendix E Loading Ordnance Survey MasterMap ITN data into Oracle Spatial 11g

Prerequisites

1) Working installation of Oracle Client (not the *Instant Client*)

2) Mastermap ITN dataset in ESRI shapefile format. *See Appendix C for details on converting MasterMap GML into shapefile format.*

3) Connection parameters and write access to an Oracle Spatial 11g database instance

4) Install of Oracle's *shp2sdo* tool

Procedure

The following steps were followed to load the Ordnance Survey Mastermap ITN network dataset into Oracle 11g spatial database.

1) Set up Connection to Oracle

   Create a text file named 'tnsnames.ora' containing the following text and save it into the *bin* subdirectory of Oracle Client's install directory:

```
sand =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST =
oracle.vidar.ngs.manchester.ac.uk)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = sand)
    )
  )
```

2) Set Environment

Set an environment variable named *TNS_ADMIN* to point to the aforementioned *bin* subdirectory of the Oracle Client install directory.

3) Convert shapefile to Oracle Input File

   Run the *shp2sdo* utility by browsing to the shp2sdo install directory and the subdirectory that reflects the operating system you are using; i.e.

shp2sdo_nt for windows users. Both the road_link and road_node shapefiles must be converted.

The following syntax is used to run shp2sdo which converts ESRI shapefiles into Oracle input files:
-g = the desired name of the geometry column in the resulting feature table
-x, -y = the maximum and minimum bounding box extents of the data. If not known this can be determined by opening the shapefile in a data browsing tool such as ArcCatalog (http://www.esri.com) or UDig (http://udig.refractions.net).
-t = tolerance
-s = SRID, i.e 27700 for OSGB36

```
shp2sdo      path-to-shapefile-shapefilename-noextension      –g
nameofgeomcolumn –x(-180,180) –y(-90,90) –t 0.0001 –s 27700
```

For example:

```
D:\documents and settings\administrator\my
documents\downloads\shp2sdo\shp2sdo_nt>
shp2sdo d:\data\itn\road_link_polyline –g geom. –x
(422785,434334) –y (556566,566761) –s 27700
```

This procedure should create three new files in your working directory with the same name as the shapefile, but with three different extensions, .sql, .dat and .ctl. Respectively these files are used to create the table structure in Oracle, store the data and define how the data is to be inserted into the tables.

4) Create Table Structure in Oracle

Run sqlplus from the Oracle Client/bin directory using the following command :

```
D:\Oracle_client\product\11.1.0\client_2\BIN>sqlplus
username/password @database-instancename
```

This command will refer to the *tnsnames.ora* file to connnect to the Oracle instance.

To create the table structure run the SQL file created in the previous step :

```
@ road_link_polyline.sql
```

Finally exit sqlplus by typing :

```
Exit
```

5) Load Data

   Edit the .ctl file INFILE value to point to path of .dat file (put in single quotes, eg `INFILE='D:/TEMP/MYFILE.DAT'`

   Set up the environment to run sqlldr:

```
set ORACLE_BASE=C:\oracle\product\11.2.0
set ORACLE_HOME=c:\oracle\product\11.2.0\dbhome_1
set PATH=$PATH:$ORACLE_HOME/bin
```

From the Oracle client/bin directory run sqlldr using the following command:

```
Sqlldr username/password@database-instancename
CONTROL= d:\data\itn\road_link_polyline.ctl
```

6) Validate Data

   To ensure the data has loaded successfully connect to the database through SQLPLUS once more and run the following commands.

```
create table validation(sdo_rowid rowid, status varchar2(2000));
execute
SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT('road_link_polyline','geom'
,'validation');
SELECT * FROM VALIDATION;
```

   If successful the value 'TRUE' should appear as result.

7) Create a spatial index on the table

```
create index roads_index on road_link_polyline(geom) indextype
is mdsys.SPATIAL_INDEX;
```

Appendix F Generation of a Spatial Road Network in Oracle

1) Create LINK_TABLE

```
CREATE TABLE link_table
(link_id NUMBER,
 toid NUMBER,
 toid_direction CHAR(1),
 start_node_id NUMBER,
 end_node_id NUMBER,
 geometry_column SDO_GEOMETRY,
 bidirected CHAR(1),
 link_name VARCHAR2(50),
 link_length NUMBER,
 cost NUMBER,
 hr_4 NUMBER, hr_5 NUMBER, hr_6 NUMBER, hr_7 NUMBER, hr_8 NUMBER, hr_9 NUMBER, hr_10 NUMBER, hr_11 NUMBER,
 hr_12 NUMBER, hr_13 NUMBER, hr_14 NUMBER, hr_15 NUMBER, hr_16 NUMBER, hr_17 NUMBER, hr_18 NUMBER, hr_19 NUMBER,
 hr_20 NUMBER, hr_21 NUMBER, hr_22 NUMBER, hr_23 NUMBER, hr_24 NUMBER);
```

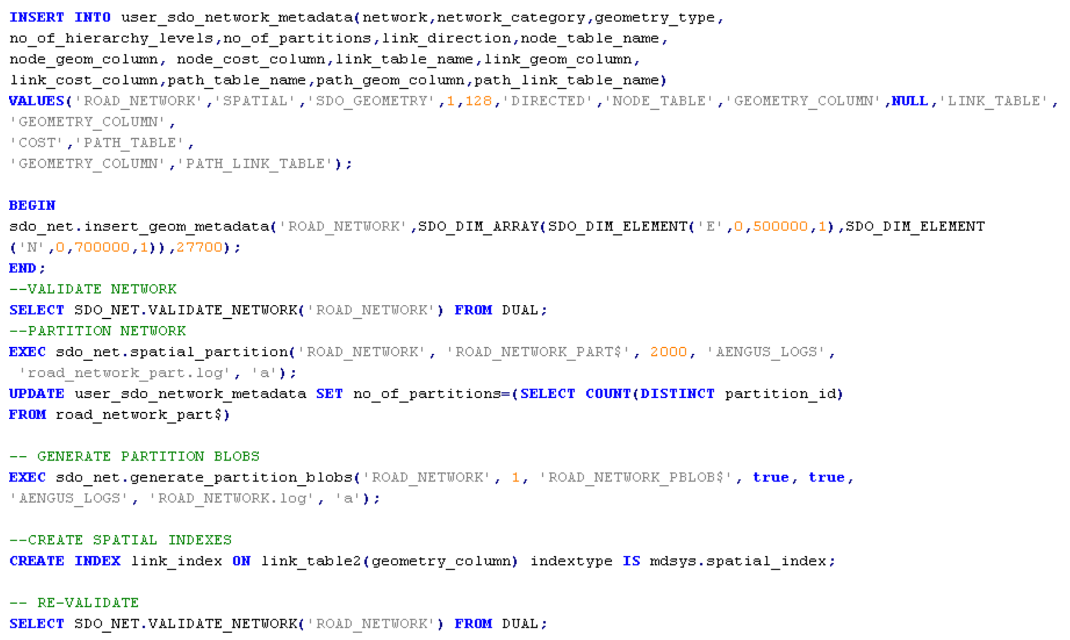
2) Create and Populate TOID_LINKID_LOOKUP

```
CREATE TABLE toid_linkid_lookup
(link_id NUMBER,
 toid NUMBER,
 toid_direction CHAR(1));

CREATE SEQUENCE  "AENGUS"."LINKID_SEQUENCE"  MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1
START WITH 1 CACHE 20 NOORDER  NOCYCLE ;

INSERT INTO toid_linkid_lookup (toid) SELECT TO_NUMBER(toid) FROM road_link_polyline;

UPDATE toid_linkid_lookup SET toid_direction='F';

UPDATE toid_linkid_lookup SET link_id = LINKID_SEQUENCE.nextval;

INSERT INTO toid_linkid_lookup (toid) SELECT TO_NUMBER(toid) FROM road_link_polyline;
UPDATE toid_linkid_lookup SET toid_direction='B' where toid_direction IS NULL;
UPDATE toid_linkid_lookup SET link_id = LINKID_SEQUENCE.nextval WHERE link_id IS NULL;
```

3) Join and Populate LINK_TABLE

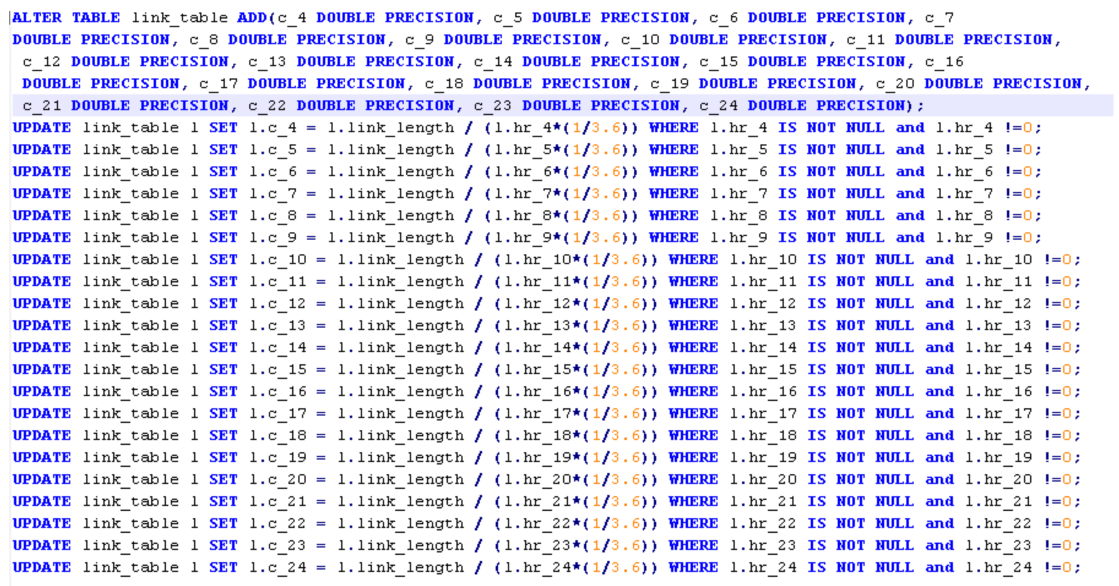
```
INSERT INTO link_table
(link_id, toid,toid_direction,start_node_id,end_node_id, geometry_column,bidirected,link_name,link_length,hr_4,hr_5,hr_6,
hr_7,hr_8,hr_9,hr_10,hr_11,hr_12,hr_13,hr_14,hr_15,hr_16,hr_17,hr_18,hr_19,hr_20,hr_21,hr_22,hr_23,hr_24)

SELECT
l.link_id,r.toid,'F',r.start_node, r.end_node,r.geometry,'N', r.road_name,a.linklength,a.hr_4,a.hr_5,a.hr_6,a.hr_7,a.hr_8,
a.hr_9,a.hr_10,a.hr_11,a.hr_12,a.hr_13,a.hr_14,a.hr_15,a.hr_16,a.hr_17,a.hr_18,a.hr_19,a.hr_20,a.hr_21,a.hr_22,a.hr_23,a.hr_24
FROM    road_link_polyline r LEFT OUTER JOIN ncc_travel_time_data a ON r.toid=a.toid and a.direction='F'
 JOIN toid_linkid_lookup l on r.toid=l.toid AND 'F' = l.toid_direction;

INSERT INTO link_table
(link_id,toid,toid_direction,start_node_id,end_node_id,geometry_column,bidirected,link_name,link_length,hr_4,hr_5,hr_6,
hr_7,hr_8, hr_9,hr_10,hr_11,hr_12,hr_13,hr_14,hr_15,hr_16,hr_17,hr_18,hr_19,hr_20,hr_21,hr_22,hr_23,hr_24)
SELECT l.link_id, r.toid,'B',r.end_node,r.start_node,r.geometry,'N' ,r.road_name,a.linklength,a.hr_4,a.hr_5,a.hr_6,a.hr_7,
a.hr_8,a.hr_9,a.hr_10,a.hr_11,a.hr_12,a.hr_13,a.hr_14,a.hr_15,a.hr_16,a.hr_17,a.hr_18,a.hr_19,a.hr_20,a.hr_21,a.hr_22,
a.hr_23,a.hr_24
FROM road_link_polyline r LEFT OUTER JOIN apriori_speed a ON r.toid=a.toid and a.direction='B' JOIN toid_linkid_lookup l
 ON r.toid = l.toid AND 'B'=l.toid_direction;

CREATE TABLE temp_table (link_id number, toid number, toid_direction char(1), start_node_id number, end_node_id number,
geometry_column sdo_geometry, bidirected char(1), link_name VARCHAR2(50), link_length NUMBER, cost NUMBER, hr_4 NUMBER,
hr_5 NUMBER, hr_6 NUMBER, hr_7 NUMBER, hr_8 NUMBER, hr_9 NUMBER, hr_10 NUMBER, hr_11 NUMBER, hr_12 NUMBER, hr_13 NUMBER,
hr_14 NUMBER, hr_15 NUMBER, hr_16 NUMBER, hr_17 NUMBER, hr_18 NUMBER, hr_19 NUMBER, hr_20 NUMBER, hr_21 NUMBER, hr_22 NUMBER,
hr_23 NUMBER, hr_24 NUMBER, start_n NUMBER, end_n NUMBER);

INSERT INTO temp_table(link_id,toid,toid_direction,start_node_id,
end_node_id, geometry_column,bidirected,link_name,link_length,cost,hr_4, hr_5, hr_6,hr_7,hr_8, hr_9,hr_10,hr_11,hr_12,hr_13,
hr_14,hr_15,hr_16,hr_17, hr_18, hr_19, hr_20,hr_21,hr_22,hr_23,hr_24)
SELECT l.link_id,l.toid,l.toid_direction,t.nodeid,p.nodeid, l.geometry_column, l.bidirected,l.link_name,l.link_length,l.cost,
l.hr_4, l.hr_5,l.hr_6,l.hr_7, l.hr_8,l.hr_9, l.hr_10, l.hr_11,l.hr_12,l.hr_13, l.hr_14,l.hr_15,l.hr_16, l.hr_17,l.hr_18,l.hr_19,
l.hr_20,l.hr_21,l.hr_22, l.hr_23,l.hr_24
FROM link_table l JOIN toid_nodeid_lookup t ON l.start_node_id = t.node_toid JOIN toid_nodeid_lookup p ON p.node_toid =
l.end_node_id;

DROP TABLE link_table;
RENAME temp_table TO link_table;
ALTER TABLE link_table add(link_level number);
UPDATE link_table SET link_level=1;
```

4) Create, Validate and Partition the Network



```
INSERT INTO user_sdo_network_metadata(network,network_category,geometry_type,
no_of_hierarchy_levels,no_of_partitions,link_direction,node_table_name,
node_geom_column, node_cost_column,link_table_name,link_geom_column,
link_cost_column,path_table_name,path_geom_column,path_link_table_name)
VALUES('ROAD_NETWORK','SPATIAL','SDO_GEOMETRY',1,128,'DIRECTED','NODE_TABLE','GEOMETRY_COLUMN',NULL,'LINK_TABLE',
'GEOMETRY_COLUMN',
'COST','PATH_TABLE',
'GEOMETRY_COLUMN','PATH_LINK_TABLE');

BEGIN
sdo_net.insert_geom_metadata('ROAD_NETWORK',SDO_DIM_ARRAY(SDO_DIM_ELEMENT('E',0,500000,1),SDO_DIM_ELEMENT
('N',0,700000,1)),27700);
END;
--VALIDATE NETWORK
SELECT SDO_NET.VALIDATE_NETWORK('ROAD_NETWORK') FROM DUAL;
--PARTITION NETWORK
EXEC sdo_net.spatial_partition('ROAD_NETWORK', 'ROAD_NETWORK_PART$', 2000, 'AENGUS_LOGS',
 'road_network_part.log', 'a');
UPDATE user_sdo_network_metadata SET no_of_partitions=(SELECT COUNT(DISTINCT partition_id)
FROM road_network_part$)

-- GENERATE PARTITION BLOBS
EXEC sdo_net.generate_partition_blobs('ROAD_NETWORK', 1, 'ROAD_NETWORK_PBLOB$', true, true,
'AENGUS_LOGS', 'ROAD_NETWORK.log', 'a');

--CREATE SPATIAL INDEXES
CREATE INDEX link_index ON link_table2(geometry_column) indextype IS mdsys.spatial_index;

-- RE-VALIDATE
SELECT SDO_NET.VALIDATE_NETWORK('ROAD_NETWORK') FROM DUAL;
```

5) Add Travel Time Column

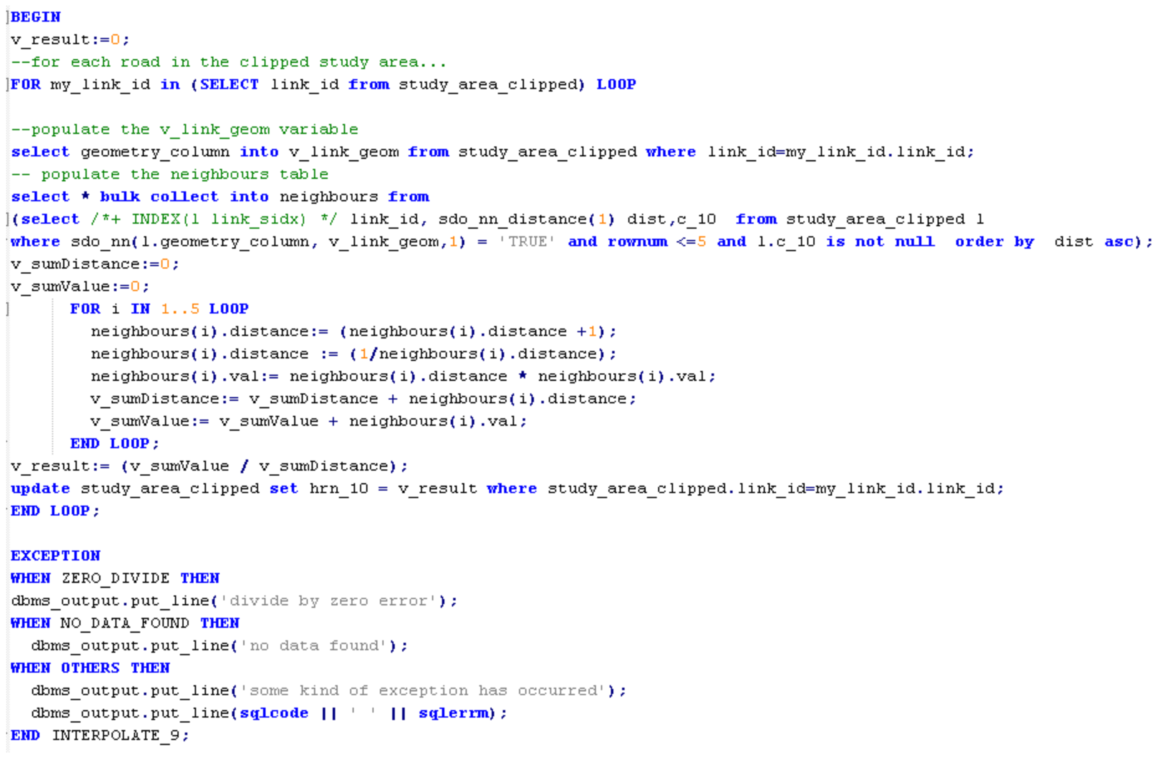


```
ALTER TABLE link_table ADD(c_4 DOUBLE PRECISION, c_5 DOUBLE PRECISION, c_6 DOUBLE PRECISION, c_7
DOUBLE PRECISION, c_8 DOUBLE PRECISION, c_9 DOUBLE PRECISION, c_10 DOUBLE PRECISION, c_11 DOUBLE PRECISION,
 c_12 DOUBLE PRECISION, c_13 DOUBLE PRECISION, c_14 DOUBLE PRECISION, c_15 DOUBLE PRECISION, c_16
 DOUBLE PRECISION, c_17 DOUBLE PRECISION, c_18 DOUBLE PRECISION, c_19 DOUBLE PRECISION, c_20 DOUBLE PRECISION,
 c_21 DOUBLE PRECISION, c_22 DOUBLE PRECISION, c_23 DOUBLE PRECISION, c_24 DOUBLE PRECISION);
UPDATE link_table l SET l.c_4 = l.link_length / (l.hr_4*(1/3.6)) WHERE l.hr_4 IS NOT NULL and l.hr_4 !=0;
UPDATE link_table l SET l.c_5 = l.link_length / (l.hr_5*(1/3.6)) WHERE l.hr_5 IS NOT NULL and l.hr_5 !=0;
UPDATE link_table l SET l.c_6 = l.link_length / (l.hr_6*(1/3.6)) WHERE l.hr_6 IS NOT NULL and l.hr_6 !=0;
UPDATE link_table l SET l.c_7 = l.link_length / (l.hr_7*(1/3.6)) WHERE l.hr_7 IS NOT NULL and l.hr_7 !=0;
UPDATE link_table l SET l.c_8 = l.link_length / (l.hr_8*(1/3.6)) WHERE l.hr_8 IS NOT NULL and l.hr_8 !=0;
UPDATE link_table l SET l.c_9 = l.link_length / (l.hr_9*(1/3.6)) WHERE l.hr_9 IS NOT NULL and l.hr_9 !=0;
UPDATE link_table l SET l.c_10 = l.link_length / (l.hr_10*(1/3.6)) WHERE l.hr_10 IS NOT NULL and l.hr_10 !=0;
UPDATE link_table l SET l.c_11 = l.link_length / (l.hr_11*(1/3.6)) WHERE l.hr_11 IS NOT NULL and l.hr_11 !=0;
UPDATE link_table l SET l.c_12 = l.link_length / (l.hr_12*(1/3.6)) WHERE l.hr_12 IS NOT NULL and l.hr_12 !=0;
UPDATE link_table l SET l.c_13 = l.link_length / (l.hr_13*(1/3.6)) WHERE l.hr_13 IS NOT NULL and l.hr_13 !=0;
UPDATE link_table l SET l.c_14 = l.link_length / (l.hr_14*(1/3.6)) WHERE l.hr_14 IS NOT NULL and l.hr_14 !=0;
UPDATE link_table l SET l.c_15 = l.link_length / (l.hr_15*(1/3.6)) WHERE l.hr_15 IS NOT NULL and l.hr_15 !=0;
UPDATE link_table l SET l.c_16 = l.link_length / (l.hr_16*(1/3.6)) WHERE l.hr_16 IS NOT NULL and l.hr_16 !=0;
UPDATE link_table l SET l.c_17 = l.link_length / (l.hr_17*(1/3.6)) WHERE l.hr_17 IS NOT NULL and l.hr_17 !=0;
UPDATE link_table l SET l.c_18 = l.link_length / (l.hr_18*(1/3.6)) WHERE l.hr_18 IS NOT NULL and l.hr_18 !=0;
UPDATE link_table l SET l.c_19 = l.link_length / (l.hr_19*(1/3.6)) WHERE l.hr_19 IS NOT NULL and l.hr_19 !=0;
UPDATE link_table l SET l.c_20 = l.link_length / (l.hr_20*(1/3.6)) WHERE l.hr_20 IS NOT NULL and l.hr_20 !=0;
UPDATE link_table l SET l.c_21 = l.link_length / (l.hr_21*(1/3.6)) WHERE l.hr_21 IS NOT NULL and l.hr_21 !=0;
UPDATE link_table l SET l.c_22 = l.link_length / (l.hr_22*(1/3.6)) WHERE l.hr_22 IS NOT NULL and l.hr_22 !=0;
UPDATE link_table l SET l.c_23 = l.link_length / (l.hr_23*(1/3.6)) WHERE l.hr_23 IS NOT NULL and l.hr_23 !=0;
UPDATE link_table l SET l.c_24 = l.link_length / (l.hr_24*(1/3.6)) WHERE l.hr_24 IS NOT NULL and l.hr_24 !=0;
```

6) Create and Populate STUDY_AREA_CLIPPED

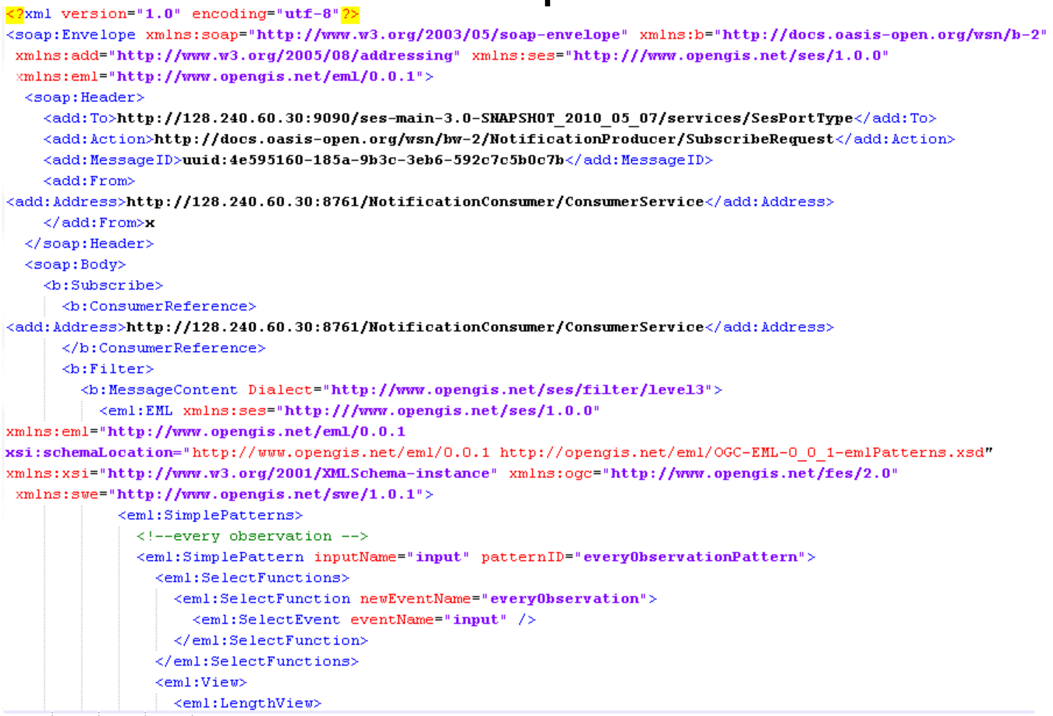
```sql
CREATE TABLE study_area_clipped as (select * from LINK_TABLE WHERE sdo_relate(geometry_column,(select
sdo_aggr_mbr(geometry_column) from LINK_TABLE WHERE hr_4 is not null),'mask=anyinteract')='TRUE');
insert into user_sdo_geom_metadata values( 'study_area_clipped','GEOMETRY_COLUMN',
SDO_DIM_ARRAY(SDO_DIM_ELEMENT('E',407130.000,439906.000,0.0001),SDO_DIM_ELEMENT
('N',544362.000,585752.000,0.0001)),27700);
CREATE index study_areac_index on study_area_clipped(geometry_column) indextype is mdsys.spatial_index;
alter TABLE study_area_clipped add constraint studyareac_pk primary key(link_id, toid_direction);
CALL SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT('study_area_clipped','geometry_column','VAL_RESULTS');
select * from val_results;

ALTER TABLE link_TABLE add(c_4 DOUBLE PRECISION, c_5 DOUBLE PRECISION, c_6 DOUBLE PRECISION, c_7
DOUBLE PRECISION, c_8 DOUBLE PRECISION, c_9 DOUBLE PRECISION, c_10 DOUBLE PRECISION, c_11 DOUBLE PRECISION,
c_12 DOUBLE PRECISION, c_13 DOUBLE PRECISION, c_14 DOUBLE PRECISION, c_15 DOUBLE PRECISION, c_16
 DOUBLE PRECISION, c_17 DOUBLE PRECISION, c_18 DOUBLE PRECISION, c_19 DOUBLE PRECISION, c_20 DOUBLE PRECISION,
 c_21 DOUBLE PRECISION, c_22 DOUBLE PRECISION, c_23 DOUBLE PRECISION,
c_24 DOUBLE PRECISION);
ALTER TABLE link_TABLE add(HRN_4 DOUBLE PRECISION, HRN_5 DOUBLE PRECISION, HRN_6 DOUBLE PRECISION, HRN_7
 DOUBLE PRECISION, HRN_8 DOUBLE PRECISION, HRN_9 DOUBLE PRECISION, HRN_10 DOUBLE PRECISION, HRN_11
 DOUBLE PRECISION, HRN_12 DOUBLE PRECISION, HRN_13 DOUBLE PRECISION, HRN_14 DOUBLE PRECISION, HRN_15
 DOUBLE PRECISION, HRN_16 DOUBLE PRECISION, HRN_17 DOUBLE PRECISION, HRN_18 DOUBLE PRECISION, HRN_19
 DOUBLE PRECISION, HRN_20 DOUBLE PRECISION, HRN_21 DOUBLE PRECISION, HRN_22 DOUBLE PRECISION, HRN_23 DOUBLE PRECISION,
HRN_24 DOUBLE PRECISION);
UPDATE link_TABLE l SET l.c_4 = l.link_length / (l.hr_4*(1/3.6)) WHERE l.hr_4 is not null and l.hr_4 !=0;
UPDATE link_TABLE l SET l.c_5 = l.link_length / (l.hr_5*(1/3.6)) WHERE l.hr_5 is not null and l.hr_5 !=0;
UPDATE link_TABLE l SET l.c_6 = l.link_length / (l.hr_6*(1/3.6)) WHERE l.hr_6 is not null and l.hr_6 !=0;
UPDATE link_TABLE l SET l.c_7 = l.link_length / (l.hr_7*(1/3.6)) WHERE l.hr_7 is not null and l.hr_7 !=0;
UPDATE link_TABLE l SET l.c_8 = l._length / (l.hr_8*(1/3.6)) WHERE l.hr_8 is not null and l.hr_8 !=0;
UPDATE link_TABLE l SET l.c_9 = l.link_length / (l.hr_9*(1/3.6)) WHERE l.hr_9 is not null and l.hr_9 !=0;
UPDATE link_TABLE l SET l.c_10 = l.link_length / (l.hr_10*(1/3.6)) WHERE l.hr_10 is not null and l.hr_10 !=0;
UPDATE link_TABLE l SET l.c_11 = l.link_length / (l.hr_11*(1/3.6)) WHERE l.hr_11 is not null and l.hr_11 !=0;
UPDATE link_TABLE l SET l.c_12 = l.link_length / (l.hr_12*(1/3.6)) WHERE l.hr_12 is not null and l.hr_12 !=0;
 UPDATE link_TABLE l SET l.c_13 = l.link_length / (l.hr_13*(1/3.6)) WHERE l.hr_13 is not null and l.hr_13 !=0;
UPDATE link_TABLE l SET l.c_14 = l.link_length / (l.hr_14*(1/3.6)) WHERE l.hr_14 is not null and l.hr_14 !=0;
UPDATE link_TABLE l SET l.c_15 = l.link_length / (l.hr_15*(1/3.6)) WHERE l.hr_15 is not null and l.hr_15 !=0;
UPDATE link_TABLE l SET l.c_16 = l.link_length / (l.hr_16*(1/3.6)) WHERE l.hr_16 is not null and l.hr_16 !=0;
UPDATE link_TABLE l SET l.c_17 = l.link_length / (l.hr_17*(1/3.6)) WHERE l.hr_17 is not null and l.hr_17 !=0;
UPDATE link_TABLE l SET l.c_18 = l.link_length / (l.hr_18*(1/3.6)) WHERE l.hr_18 is not null and l.hr_18 !=0;
UPDATE link_TABLE l SET l.c_19 = l.link_length / (l.hr_19*(1/3.6)) WHERE l.hr_19 is not null and l.hr_19 !=0;
UPDATE link_TABLE l SET l.c_20 = l.link_length / (l.hr_20*(1/3.6)) WHERE l.hr_20 is not null and l.hr_20 !=0;
UPDATE link_TABLE l SET l.c_21 = l.link_length / (l.hr_21*(1/3.6)) WHERE l.hr_21 is not null and l.hr_21 !=0;
UPDATE link_TABLE l SET l.c_22 = l.link_length / (l.hr_22*(1/3.6)) WHERE l.hr_22 is not null and l.hr_22 !=0;
UPDATE link_TABLE l SET l.c_23 = l.link_length / (l.hr_23*(1/3.6)) WHERE l.hr_23 is not null and l.hr_23 !=0;
UPDATE link_TABLE l SET l.c_24 = l.link_length / (l.hr_24*(1/3.6)) WHERE l.hr_24 is not null and l.hr_24 !=0;
```

Appendix G PL/SQL Interpolation Procedure

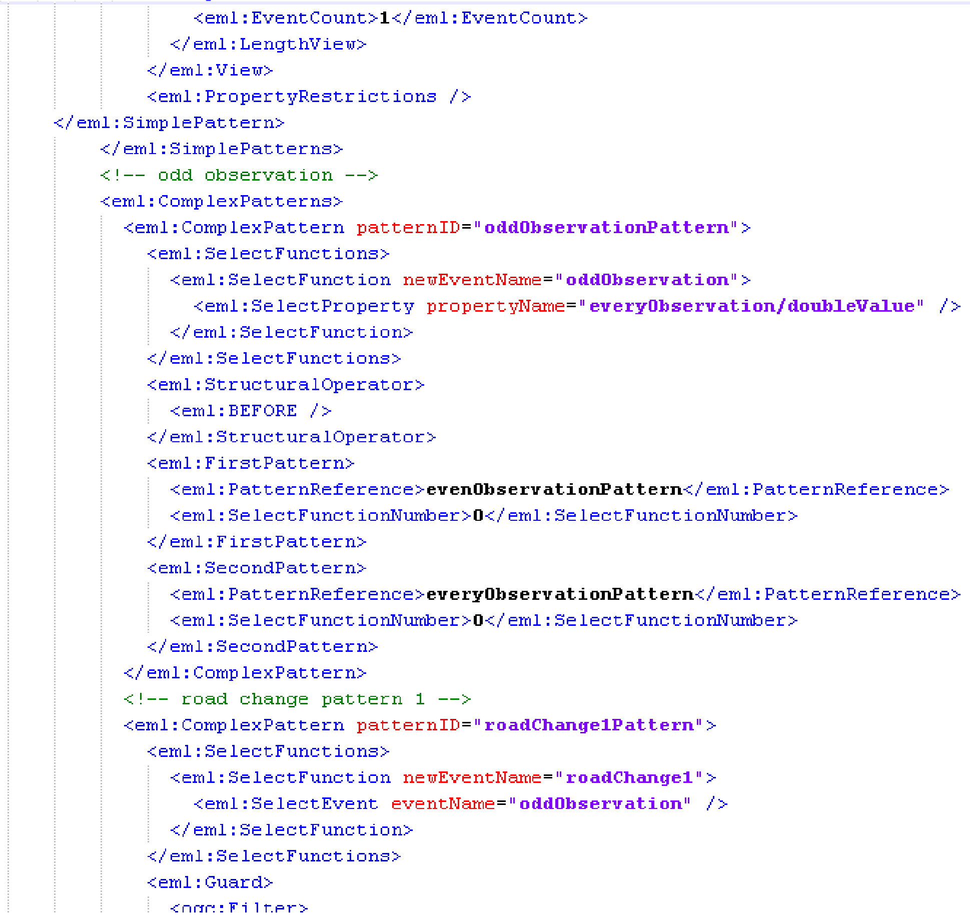


```
BEGIN
v_result:=0;
--for each road in the clipped study area...
FOR my_link_id in (SELECT link_id from study_area_clipped) LOOP

--populate the v_link_geom variable
select geometry_column into v_link_geom from study_area_clipped where link_id=my_link_id.link_id;
-- populate the neighbours table
select * bulk collect into neighbours from
(select /*+ INDEX(l link_sidx) */ link_id, sdo_nn_distance(1) dist,c_10  from study_area_clipped l
where sdo_nn(l.geometry_column, v_link_geom,1) = 'TRUE' and rownum <=5 and l.c_10 is not null  order by  dist asc);
v_sumDistance:=0;
v_sumValue:=0;
      FOR i IN 1..5 LOOP
        neighbours(i).distance:= (neighbours(i).distance +1);
        neighbours(i).distance := (1/neighbours(i).distance);
        neighbours(i).val:= neighbours(i).distance * neighbours(i).val;
        v_sumDistance:= v_sumDistance + neighbours(i).distance;
        v_sumValue:= v_sumValue + neighbours(i).val;
      END LOOP;
v_result:= (v_sumValue / v_sumDistance);
update study_area_clipped set hrn_10 = v_result where study_area_clipped.link_id=my_link_id.link_id;
END LOOP;

EXCEPTION
WHEN ZERO_DIVIDE THEN
dbms_output.put_line('divide by zero error');
WHEN NO_DATA_FOUND THEN
  dbms_output.put_line('no data found');
WHEN OTHERS THEN
  dbms_output.put_line('some kind of exception has occurred');
  dbms_output.put_line(sqlcode || ' ' || sqlerrm);
END INTERPOLATE_9;
```

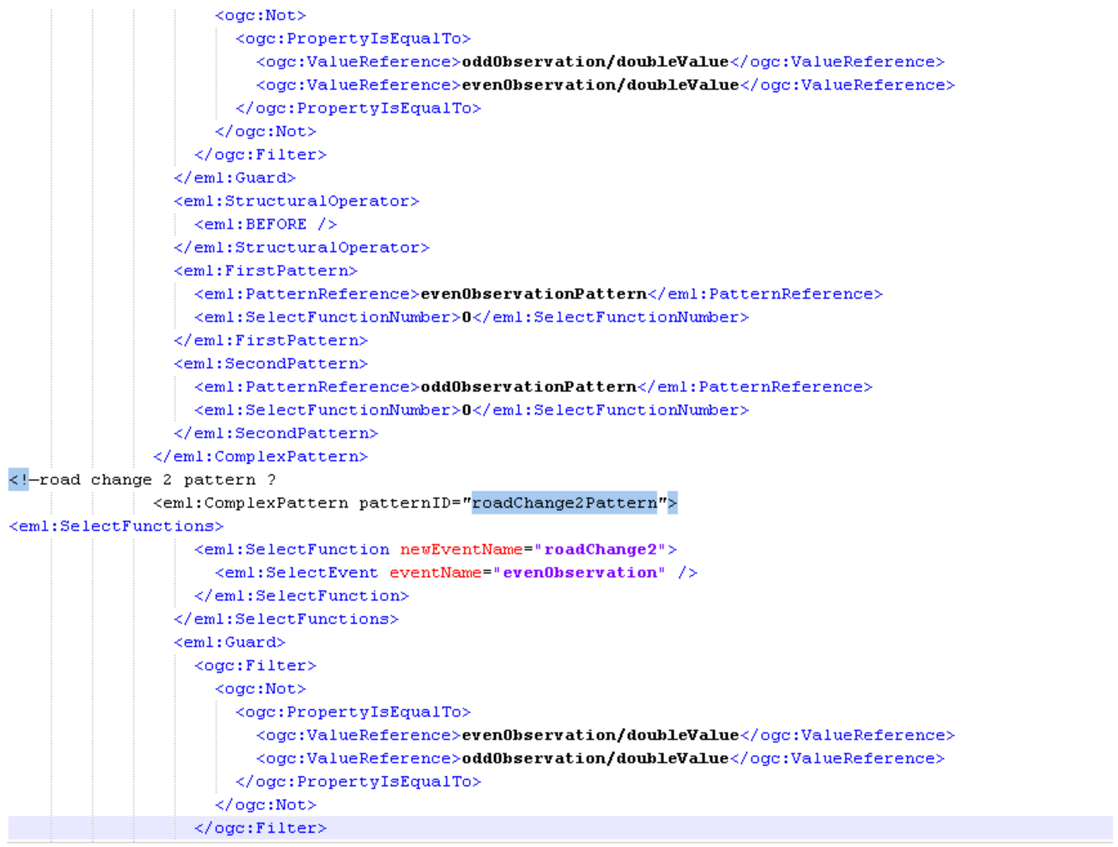
Appendix H Event Pattern Markup Language (EML) Filter Subscribe Request



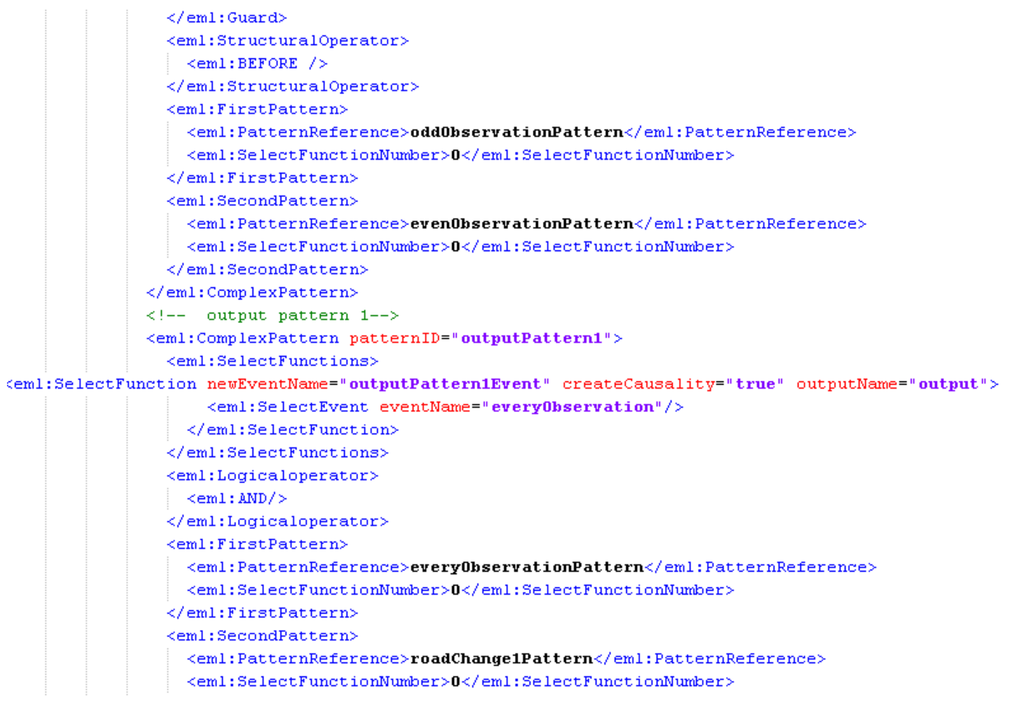
```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:b="http://docs.oasis-open.org/wsn/b-2"
 xmlns:add="http://www.w3.org/2005/08/addressing" xmlns:ses="http:///www.opengis.net/ses/1.0.0"
 xmlns:eml="http://www.opengis.net/eml/0.0.1">
  <soap:Header>
    <add:To>http://128.240.60.30:9090/ses-main-3.0-SNAPSHOT_2010_05_07/services/SesPortType</add:To>
    <add:Action>http://docs.oasis-open.org/wsn/bw-2/NotificationProducer/SubscribeRequest</add:Action>
    <add:MessageID>uuid:4e595160-185a-9b3c-3eb6-592c7c5b0c7b</add:MessageID>
    <add:From>
<add:Address>http://128.240.60.30:8761/NotificationConsumer/ConsumerService</add:Address>
    </add:From>x
  </soap:Header>
  <soap:Body>
    <b:Subscribe>
      <b:ConsumerReference>
<add:Address>http://128.240.60.30:8761/NotificationConsumer/ConsumerService</add:Address>
      </b:ConsumerReference>
      <b:Filter>
        <b:MessageContent Dialect="http://www.opengis.net/ses/filter/level3">
          <eml:EML xmlns:ses="http:///www.opengis.net/ses/1.0.0"
xmlns:eml="http://www.opengis.net/eml/0.0.1
xsi:schemaLocation="http://www.opengis.net/eml/0.0.1 http://opengis.net/eml/OGC-EML-0_0_1-emlPatterns.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ogc="http://www.opengis.net/fes/2.0"
 xmlns:swe="http://www.opengis.net/swe/1.0.1">
            <eml:SimplePatterns>
              <!--every observation -->
              <eml:SimplePattern inputName="input" patternID="everyObservationPattern">
                <eml:SelectFunctions>
                  <eml:SelectFunction newEventName="everyObservation">
                    <eml:SelectEvent eventName="input" />
                  </eml:SelectFunction>
                </eml:SelectFunctions>
                <eml:View>
                  <eml:LengthView>
```



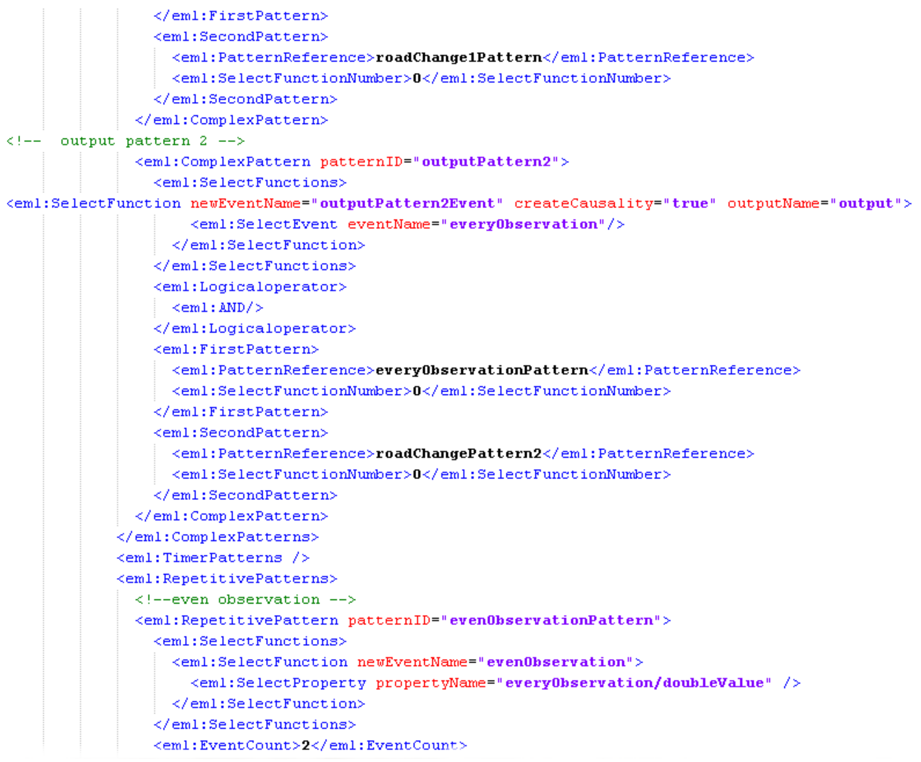
```
                    <eml:EventCount>1</eml:EventCount>
                  </eml:LengthView>
                </eml:View>
                <eml:PropertyRestrictions />
            </eml:SimplePattern>
              </eml:SimplePatterns>
              <!-- odd observation -->
              <eml:ComplexPatterns>
                <eml:ComplexPattern patternID="oddObservationPattern">
                  <eml:SelectFunctions>
                    <eml:SelectFunction newEventName="oddObservation">
                      <eml:SelectProperty propertyName="everyObservation/doubleValue" />
                    </eml:SelectFunction>
                  </eml:SelectFunctions>
                  <eml:StructuralOperator>
                    <eml:BEFORE />
                  </eml:StructuralOperator>
                  <eml:FirstPattern>
                    <eml:PatternReference>evenObservationPattern</eml:PatternReference>
                    <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                  </eml:FirstPattern>
                  <eml:SecondPattern>
                    <eml:PatternReference>everyObservationPattern</eml:PatternReference>
                    <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                  </eml:SecondPattern>
                </eml:ComplexPattern>
                <!-- road change pattern 1 -->
                <eml:ComplexPattern patternID="roadChange1Pattern">
                  <eml:SelectFunctions>
                    <eml:SelectFunction newEventName="roadChange1">
                      <eml:SelectEvent eventName="oddObservation" />
                    </eml:SelectFunction>
                  </eml:SelectFunctions>
                  <eml:Guard>
                    <ogc:Filter>
```



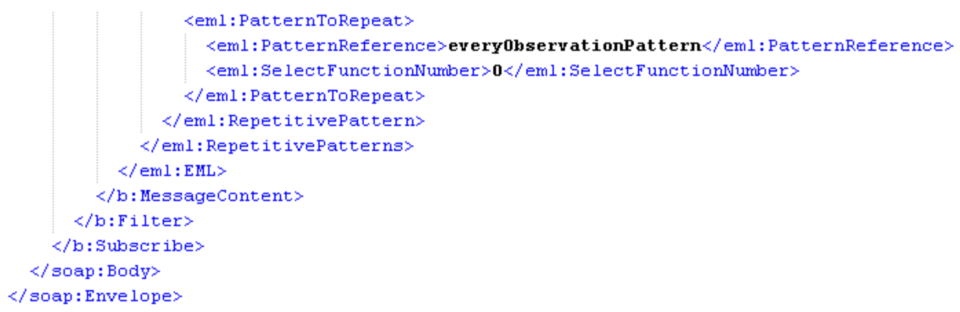
```
                    <ogc:Not>
                      <ogc:PropertyIsEqualTo>
                        <ogc:ValueReference>oddObservation/doubleValue</ogc:ValueReference>
                        <ogc:ValueReference>evenObservation/doubleValue</ogc:ValueReference>
                      </ogc:PropertyIsEqualTo>
                    </ogc:Not>
                  </ogc:Filter>
                </eml:Guard>
                <eml:StructuralOperator>
                  <eml:BEFORE />
                </eml:StructuralOperator>
                <eml:FirstPattern>
                  <eml:PatternReference>evenObservationPattern</eml:PatternReference>
                  <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                </eml:FirstPattern>
                <eml:SecondPattern>
                  <eml:PatternReference>oddObservationPattern</eml:PatternReference>
                  <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                </eml:SecondPattern>
              </eml:ComplexPattern>
<!—road change 2 pattern ?
              <eml:ComplexPattern patternID="roadChange2Pattern">
<eml:SelectFunctions>
                  <eml:SelectFunction newEventName="roadChange2">
                    <eml:SelectEvent eventName="evenObservation" />
                  </eml:SelectFunction>
                </eml:SelectFunctions>
                <eml:Guard>
                  <ogc:Filter>
                    <ogc:Not>
                      <ogc:PropertyIsEqualTo>
                        <ogc:ValueReference>evenObservation/doubleValue</ogc:ValueReference>
                        <ogc:ValueReference>oddObservation/doubleValue</ogc:ValueReference>
                      </ogc:PropertyIsEqualTo>
                    </ogc:Not>
                  </ogc:Filter>
```



```
                </eml:Guard>
                <eml:StructuralOperator>
                  <eml:BEFORE />
                </eml:StructuralOperator>
                <eml:FirstPattern>
                  <eml:PatternReference>oddObservationPattern</eml:PatternReference>
                  <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                </eml:FirstPattern>
                <eml:SecondPattern>
                  <eml:PatternReference>evenObservationPattern</eml:PatternReference>
                  <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                </eml:SecondPattern>
              </eml:ComplexPattern>
              <!--  output pattern 1-->
              <eml:ComplexPattern patternID="outputPattern1">
                <eml:SelectFunctions>
<eml:SelectFunction newEventName="outputPattern1Event" createCausality="true" outputName="output">
                    <eml:SelectEvent eventName="everyObservation"/>
                  </eml:SelectFunction>
                </eml:SelectFunctions>
                <eml:Logicaloperator>
                  <eml:AND/>
                </eml:Logicaloperator>
                <eml:FirstPattern>
                  <eml:PatternReference>everyObservationPattern</eml:PatternReference>
                  <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                </eml:FirstPattern>
                <eml:SecondPattern>
                  <eml:PatternReference>roadChange1Pattern</eml:PatternReference>
                  <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
```

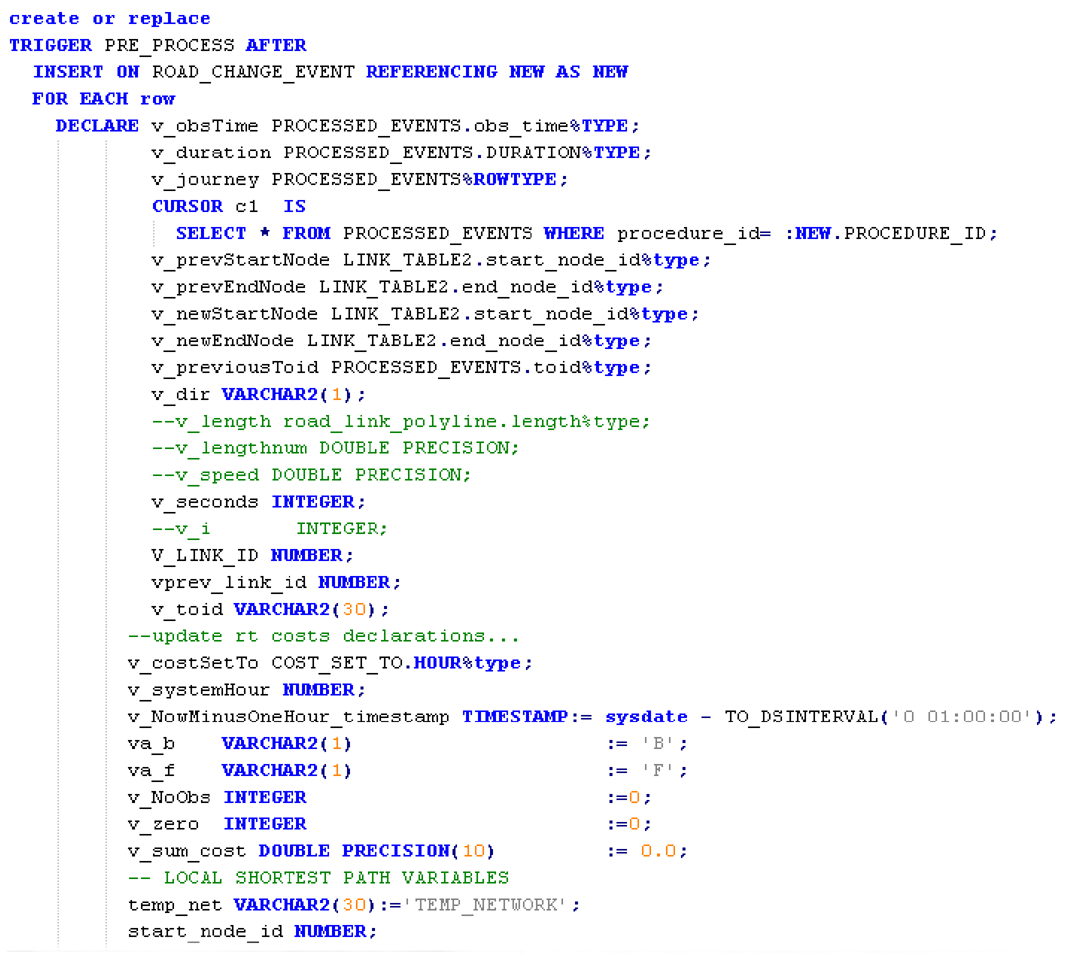


```
                <eml:FirstPattern>
                <eml:SecondPattern>
                  <eml:PatternReference>roadChange1Pattern</eml:PatternReference>
                  <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                </eml:SecondPattern>
              </eml:ComplexPattern>
<!--  output pattern 2 -->
              <eml:ComplexPattern patternID="outputPattern2">
                <eml:SelectFunctions>
<eml:SelectFunction newEventName="outputPattern2Event" createCausality="true" outputName="output">
                    <eml:SelectEvent eventName="everyObservation"/>
                  </eml:SelectFunction>
                </eml:SelectFunctions>
                <eml:Logicaloperator>
                  <eml:AND/>
                </eml:Logicaloperator>
                <eml:FirstPattern>
                  <eml:PatternReference>everyObservationPattern</eml:PatternReference>
                  <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                </eml:FirstPattern>
                <eml:SecondPattern>
                  <eml:PatternReference>roadChangePattern2</eml:PatternReference>
                  <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                </eml:SecondPattern>
              </eml:ComplexPattern>
            </eml:ComplexPatterns>
            <eml:TimerPatterns />
            <eml:RepetitivePatterns>
              <!--even observation -->
              <eml:RepetitivePattern patternID="evenObservationPattern">
                <eml:SelectFunctions>
                  <eml:SelectFunction newEventName="evenObservation">
                    <eml:SelectProperty propertyName="everyObservation/doubleValue" />
                  </eml:SelectFunction>
                </eml:SelectFunctions>
                <eml:EventCount>2</eml:EventCount>
```

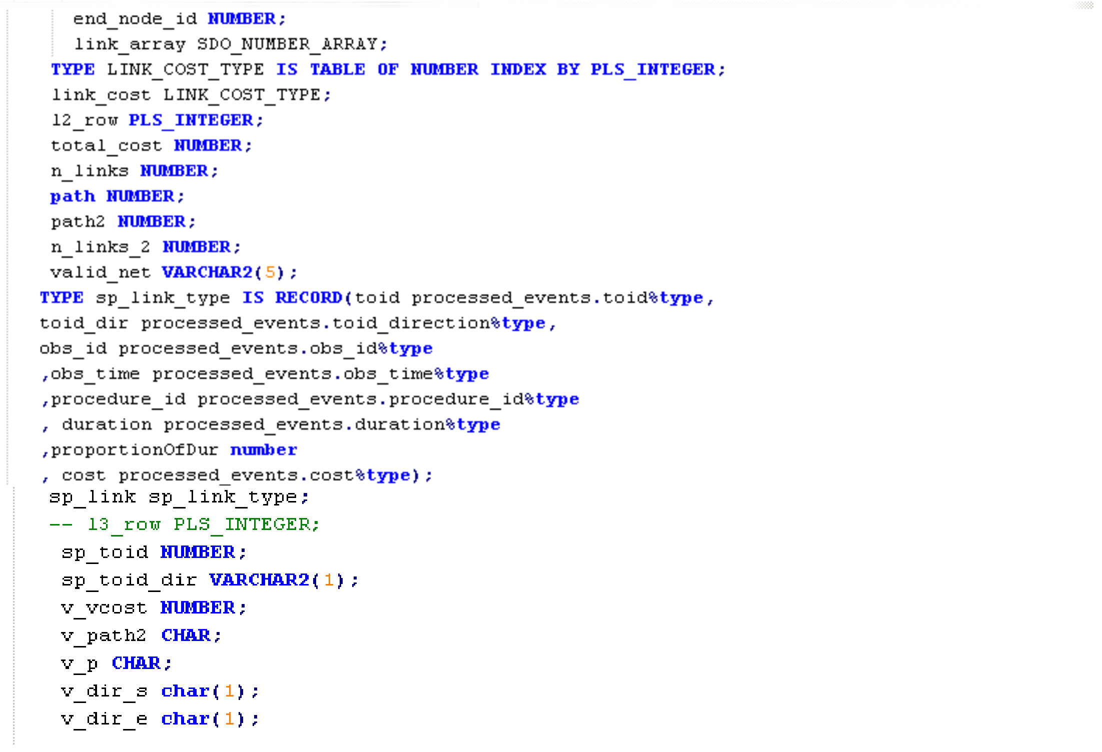


```
                <eml:PatternToRepeat>
                  <eml:PatternReference>everyObservationPattern</eml:PatternReference>
                  <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
                </eml:PatternToRepeat>
              </eml:RepetitivePattern>
            </eml:RepetitivePatterns>
          </eml:EML>
        </b:MessageContent>
      </b:Filter>
    </b:Subscribe>
  </soap:Body>
</soap:Envelope>
```

Appendix I PL/SQL Derived Attribute and COST Column Calculation Trigger

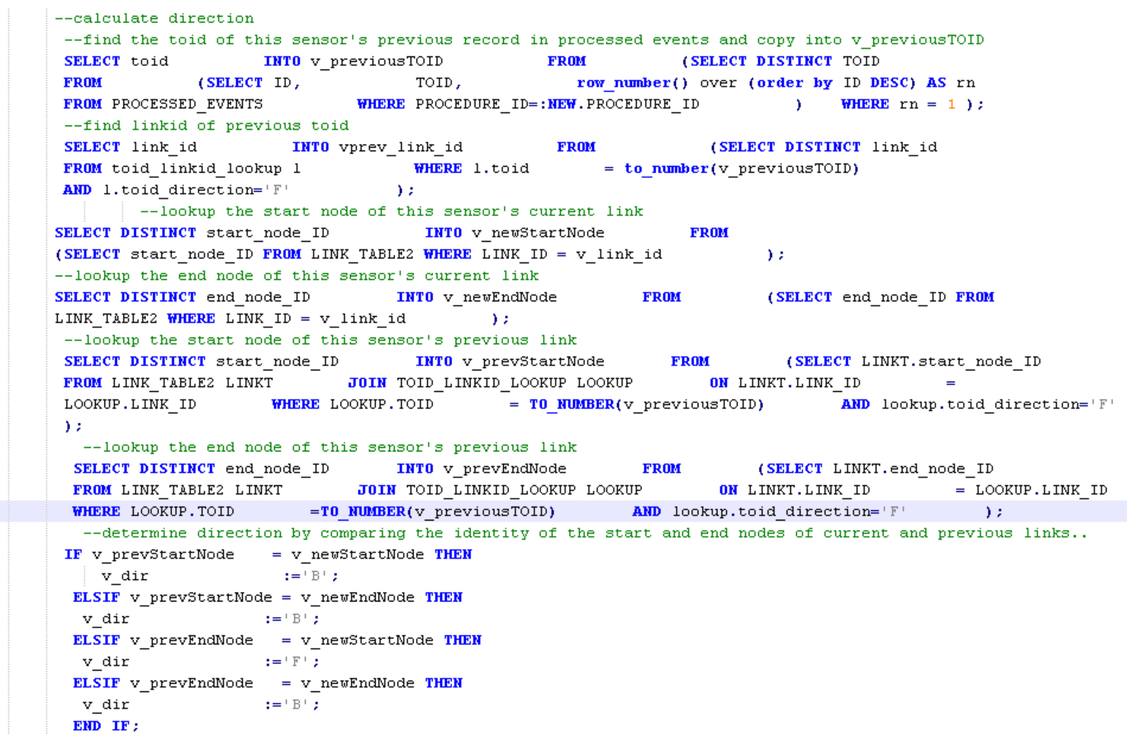


```
create or replace
TRIGGER PRE_PROCESS AFTER
  INSERT ON ROAD_CHANGE_EVENT REFERENCING NEW AS NEW
  FOR EACH row
    DECLARE v_obsTime PROCESSED_EVENTS.obs_time%TYPE;
            v_duration PROCESSED_EVENTS.DURATION%TYPE;
            v_journey PROCESSED_EVENTS%ROWTYPE;
            CURSOR c1  IS
              SELECT * FROM PROCESSED_EVENTS WHERE procedure_id= :NEW.PROCEDURE_ID;
            v_prevStartNode LINK_TABLE2.start_node_id%type;
            v_prevEndNode LINK_TABLE2.end_node_id%type;
            v_newStartNode LINK_TABLE2.start_node_id%type;
            v_newEndNode LINK_TABLE2.end_node_id%type;
            v_previousToid PROCESSED_EVENTS.toid%type;
            v_dir VARCHAR2(1);
            --v_length road_link_polyline.length%type;
            --v_lengthnum DOUBLE PRECISION;
            --v_speed DOUBLE PRECISION;
            v_seconds INTEGER;
            --v_i         INTEGER;
            V_LINK_ID NUMBER;
            vprev_link_id NUMBER;
            v_toid VARCHAR2(30);
          --update rt costs declarations...
          v_costSetTo COST_SET_TO.HOUR%type;
          v_systemHour NUMBER;
          v_NowMinusOneHour_timestamp TIMESTAMP:= sysdate - TO_DSINTERVAL('0 01:00:00');
          va_b     VARCHAR2(1)                      := 'B';
          va_f     VARCHAR2(1)                      := 'F';
          v_NoObs INTEGER                           :=0;
          v_zero  INTEGER                           :=0;
          v_sum_cost DOUBLE PRECISION(10)           := 0.0;
          -- LOCAL SHORTEST PATH VARIABLES
          temp_net VARCHAR2(30):='TEMP_NETWORK';
          start_node_id NUMBER;
```



```
          end_node_id NUMBER;
          link_array SDO_NUMBER_ARRAY;
        TYPE LINK_COST_TYPE IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
        link_cost LINK_COST_TYPE;
        l2_row PLS_INTEGER;
        total_cost NUMBER;
        n_links NUMBER;
        path NUMBER;
        path2 NUMBER;
        n_links_2 NUMBER;
        valid_net VARCHAR2(5);
       TYPE sp_link_type IS RECORD(toid processed_events.toid%type,
       toid_dir processed_events.toid_direction%type,
       obs_id processed_events.obs_id%type
       ,obs_time processed_events.obs_time%type
       ,procedure_id processed_events.procedure_id%type
       , duration processed_events.duration%type
       ,proportionOfDur number
       , cost processed_events.cost%type);
        sp_link sp_link_type;
        -- l3_row PLS_INTEGER;
         sp_toid NUMBER;
         sp_toid_dir VARCHAR2(1);
         v_vcost NUMBER;
         v_path2 CHAR;
         v_p CHAR;
         v_dir_s char(1);
         v_dir_e char(1);
```

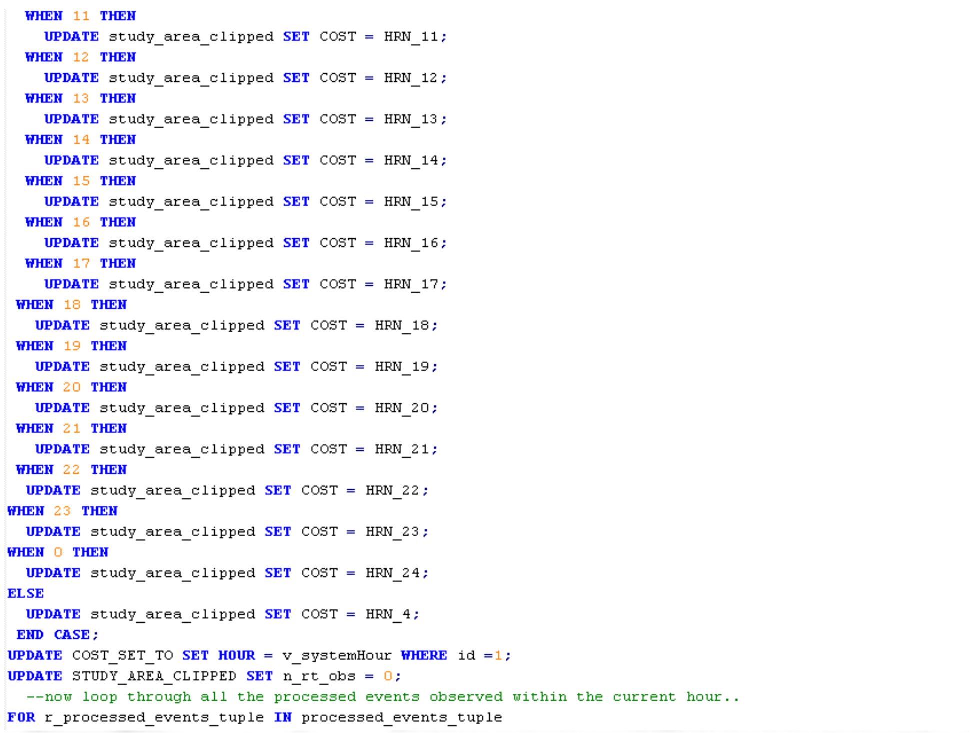
```
    --ALL PROCESSED EVENTS WITHIN CURRENT HOURS TIMESTAMP
    CURSOR processed_events_tuple IS
      SELECT DISTINCT toid, toid_direction   FROM processed_events WHERE OBS_TIME >
      v_NowMinusOneHour_timestamp   ORDER BY toid, toid_direction;
    TYPE toid_and_direction_type IS  RECORD  (    toid processed_events.toid%type,
    toid_dir processed_events.toid_direction%type);
    TYPE obs_table_type IS  TABLE OF processed_events.cost%type INDEX BY PLS_INTEGER;
    v_proc_cost processed_events.cost%type;
    l_row PLS_INTEGER;
    toid_record toid_and_direction_type;
    obs_table obs_table_type;
    v_prevLength NUMBER;
    v_costoflast NUMBER;
    v_endcheck NUMBER;
    v_endcheck2 NUMBER;
    loop_length NUMBER;
    v_reqtime timestamp;
BEGIN
v_reqtime:= systimestamp;
--CHECK THAT THE NEW OBSERVATION IS VALID (TOID MUST NOT BE NULL)
IF :NEW.TOID IS  NULL OR :NEW.TOID = 'null' THEN
  dbms_output.put_line('toid not null');
  RETURN;
END IF;

  --IF NO PREVIOUS OBSERVATIONS FOR THIS VEHICLE THEN DON'T CALCULATE DIRECTION OR DURATION.
  --populate with the first record...
  OPEN c1;
  FETCH c1 INTO v_journey;
  CLOSE c1;
  --IF this is a new rather than an existing procedure (sensor) then simply copy known values across
  as impossible to calc. duration and direction
  IF v_journey.id IS NULL THEN
     dbms_output.put_line('first obs from this sensor');
     --a. ensure that referential integrity is maintained by deleting any records with the same id
     DELETE     FROM PROCESSED_EVENTS     WHERE PROCESSED_EVENTS.OBS_ID = :NEW.OBS_ID;
     --b. copy the known values of the new record into processed_events
     INSERT     INTO PROCESSED_EVENTS      (        obs_id,        procedure_id,        toid,
     obs_time,insertion_time,request_time)
     VALUES        (         :NEW.OBS_ID,         :NEW.PROCEDURE_id,         :NEW.TOID,
     :NEW.OBS_TIME,systimestamp,v_reqtime       );
     RETURN;
    --ELSE if this is an existing procedure(sensor) then start calculating direction and duration
  ELSE
     dbms_output.put_line('previous observations found for this sensor');
     --find the timestamp of the most recent observation for this sensor and copy it into v_ObsTime variable
     SELECT obs_time      INTO v_ObsTime      FROM       (SELECT DISTINCT ID,        OBS_TIME,
     row_number() over (order by obs_time DESC) AS rn
     FROM PROCESSED_EVENTS       WHERE PROCEDURE_ID=:NEW.PROCEDURE_ID )    WHERE rn=1;
     --if no observation time then zero duration and speed values...
     --similarly if old obs time > new obs time - can happen due to SES DELAYS..
      IF v_obstime IS NULL OR v_obstime > :NEW.obs_time THEN
         dbms_output.put_line('timestamp is null');
          v_duration := to_dsinterval('0 00:00:00');
        INSERT  INTO PROCESSED_EVENTS(obs_id,procedure_id,toid,obs_time, duration,insertion_time,request_time)
        VALUES( :NEW.OBS_ID,:NEW.PROCEDURE_ID,:NEW.TOID,:NEW.OBS_TIME,v_duration,systimestamp,v_reqtime);
        RETURN;
      ELSE
         --otherwise calculate duration as the timestamp of this observation minus the timestamp of the previous
         observation
         v_duration:= :NEW.obs_time - v_obstime;
         --convert duration to integer seconds
         v_seconds:= intervaltoseconds(v_duration);

         -- lookup the link_id corresponding to the TOID present in this observation and copy it into v_link_id
         --this is potentially troublesome because unknown toids will throw a no_data_found exception...
         SELECT link_id    INTO v_link_id  FROM (SELECT DISTINCT link_id  FROM toid_linkid_lookup l  WHERE l.toid
         = to_number(:NEW.TOID) AND l.toid_direction='F'                );
         -- lookup the length of the aforementioned link and copy it into v_length
        --this is potentially troublesome because unknown toids will throw a no_data_found exception...
        SELECT link_id    INTO v_link_id  FROM (SELECT DISTINCT link_id  FROM toid_linkid_lookup l  WHERE l.toid
        = to_number(:NEW.TOID) AND l.toid_direction='F'               );
```



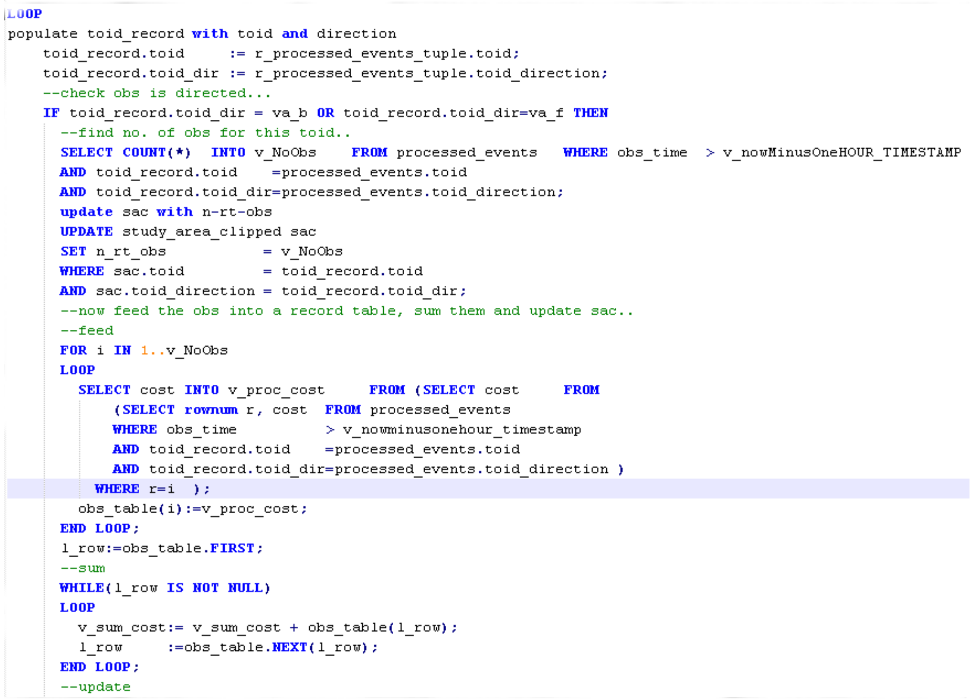
```
--calculate direction
 --find the toid of this sensor's previous record in processed events and copy into v_previousTOID
 SELECT toid          INTO v_previousTOID          FROM          (SELECT DISTINCT TOID
 FROM          (SELECT ID,            TOID,            row_number() over (order by ID DESC) AS rn
 FROM PROCESSED_EVENTS          WHERE PROCEDURE_ID=:NEW.PROCEDURE_ID          )     WHERE rn = 1 );
 --find linkid of previous toid
 SELECT link_id           INTO vprev_link_id          FROM              (SELECT DISTINCT link_id
 FROM toid_linkid_lookup l            WHERE l.toid        = to_number(v_previousTOID)
 AND l.toid_direction='F'           );
        --lookup the start node of this sensor's current link
SELECT DISTINCT start_node_ID          INTO v_newStartNode          FROM
(SELECT start_node_ID FROM LINK_TABLE2 WHERE LINK_ID = v_link_id           );
--lookup the end node of this sensor's current link
SELECT DISTINCT end_node_ID         INTO v_newEndNode          FROM          (SELECT end_node_ID FROM
LINK_TABLE2 WHERE LINK_ID = v_link_id         );
 --lookup the start node of this sensor's previous link
 SELECT DISTINCT start_node_ID        INTO v_prevStartNode       FROM        (SELECT LINKT.start_node_ID
 FROM LINK_TABLE2 LINKT        JOIN TOID_LINKID_LOOKUP LOOKUP         ON LINKT.LINK_ID          =
 LOOKUP.LINK_ID        WHERE LOOKUP.TOID         = TO_NUMBER(v_previousTOID)        AND lookup.toid_direction='F'
 );
   --lookup the end node of this sensor's previous link
  SELECT DISTINCT end_node_ID        INTO v_prevEndNode        FROM        (SELECT LINKT.end_node_ID
  FROM LINK_TABLE2 LINKT        JOIN TOID_LINKID_LOOKUP LOOKUP        ON LINKT.LINK_ID         = LOOKUP.LINK_ID
  WHERE LOOKUP.TOID        =TO_NUMBER(v_previousTOID)        AND lookup.toid_direction='F'        );
   --determine direction by comparing the identity of the start and end nodes of current and previous links..
 IF v_prevStartNode    = v_newStartNode THEN
    v_dir                :='B';
  ELSIF v_prevStartNode = v_newEndNode THEN
   v_dir                :='B';
  ELSIF v_prevEndNode    = v_newStartNode THEN
   v_dir                :='F';
  ELSIF v_prevEndNode    = v_newEndNode THEN
   v_dir                :='B';
  END IF;
```

```
--if direction could not be determined (i.e. if this and previous links are not adjacent) then guess the path
--taken between the two links,
--and insert the resulting cost directly into study_area_clipped
IF v_dir IS NULL THEN
    dbms_output.put_line('direction could not be determined, links may not be adjacent');
   --1. CREATE NETWORK
    IF v_seconds IS NOT NULL and v_seconds > 0 THEN
        delete from temp_network_links;
        delete from temp_network_nodes;
        dbms_output.put_line('creating network');
        insert into temp_network_links(SELECT * FROM LINK_TABLE2 LT WHERE SDO_WITHIN_DISTANCE(LT.GEOMETRY_COLUMN,
        (SELECT LT.GEOMETRY_COLUMN FROM LINK_TABLE2 LT WHERE LT.LINK_ID=v_link_id),'DISTANCE=2000 UNIT=M') =
        'TRUE');
        insert into temp_network_nodes(select geometry_column, node_id from node_table nt where nt.node_id in
        (select distinct nt.node_id  from node_table nt join temp_network_links tl on nt.node_id =
        tl.start_node_id or nt.node_id = tl.end_node_id));
        INSERT INTO user_sdo_network_metadata(network,network_category,geometry_type,
        no_of_hierarchy_levels,no_of_partitions,link_direction,node_table_name,
        node_geom_column, node_cost_column,link_table_name,link_geom_column,
        link_cost_column,path_table_name,path_geom_column,path_link_table_name)
        VALUES('TEMP_NETWORK','SPATIAL','SDO_GEOMETRY',1,1,'DIRECTED','temp_network_nodes','GEOMETRY_COLUMN',
        NULL,'temp_network_links','GEOMETRY_COLUMN','link_length','PATH_TABLE',
        'GEOMETRY_COLUMN','PATH_LINK_TABLE');
        dbms_output.put_line('created network');
        sdo_net.insert_geom_metadata('TEMP_NETWORK',SDO_DIM_ARRAY(SDO_DIM_ELEMENT('E',0,500000,1),
        SDO_DIM_ELEMENT('N',0,700000,1)),27700);
        dbms_output.put_line('validating network....');
        SELECT sdo_net.VALIDATE_NETWORK('TEMP_NETWORK') INTO valid_net FROM DUAL;

        IF valid_net='FALSE' THEN
            dbms_output.put_line('network invalid...');
        END IF;
        dbms_output.put_line('ok');
     --2. PERFORM SHORTEST PATH QUERY
     sdo_net_mem.network_manager.read_network('TEMP_NETWORK','TRUE');
     path:=  SDO_NET_MEM.NETWORK_MANAGER.SHORTEST_PATH(temp_net,to_number(v_prevStartNode),to_number(v_newStartNode),NULL);
     n_links:=sdo_net_mem.path.get_no_of_links(temp_net,path);
     path2:= SDO_NET_MEM.NETWORK_MANAGER.SHORTEST_PATH(temp_net,to_number(v_prevEndNode),to_number(v_newStartNode),NULL);
     n_links_2:=sdo_net_mem.path.get_no_of_links(temp_net,path2);
     if n_links < n_links_2 then
       start_node_id:= to_number(v_prevStartNode);
       v_dir_s:='F';
      else
       start_node_id:=to_number(v_prevEndNode);
       v_dir_s:='B';
     end if;

     path:=SDO_NET_MEM.NETWORK_MANAGER.SHORTEST_PATH(temp_net,start_node_id,to_number(v_NewStartNode),NULL);
      n_links:=sdo_net_mem.path.get_no_of_links(temp_net,path);
      path2:=SDO_NET_MEM.NETWORK_MANAGER.SHORTEST_PATH(temp_net,start_node_id,to_number(v_newEndNode),NULL);
     n_links_2:=sdo_net_mem.path.get_no_of_links(temp_net,path2);
     if n_links < n_links_2 then
       end_node_id:=to_number(v_NewStartNode);
       v_dir_e:='B';
     else
       end_node_id:=to_number(v_NewEndNode);
       v_dir_e:='F';
     end if;

     path:=SDO_NET_MEM.NETWORK_MANAGER.SHORTEST_PATH(temp_net,start_node_id,end_node_id,NULL);
     n_links:=sdo_net_mem.path.get_no_of_links(temp_net,path);
     link_array:= SDO_NET_MEM.PATH.GET_LINK_IDS(temp_net,path);
     v_costoflast:=sdo_net_mem.link.get_cost(temp_net,link_array(n_links));
      SELECT distinct link_length into v_prevLength from study_area_clipped where toid=v_previousTOID;
      total_cost:=  sdo_net_mem.path.get_cost(temp_net,path);
      total_cost:= total_cost + v_prevLength;

     FOR i IN link_array.FIRST..link_array.LAST LOOP
```

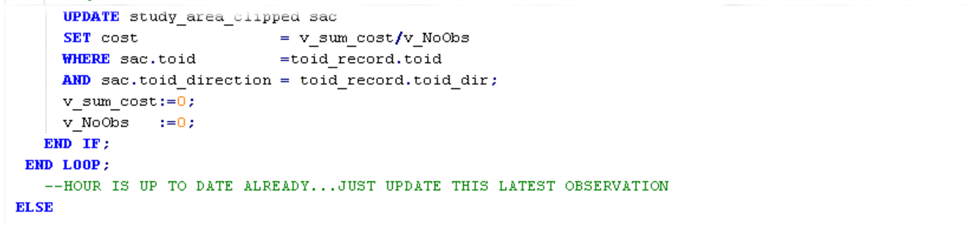
```
            SELECT TOID INTO sp_link.toid FROM TOID_LINKID_LOOKUP WHERE link_id = link_array(i);
            SELECT TOID_DIRECTION INTO sp_link.toid_dir FROM TOID_LINKID_LOOKUP tll WHERE tll.link_id=link_array(i);
            sp_link.proportionOfDur:= ((sdo_net_mem.link.get_cost(temp_net,link_array(i)))/total_cost);
            SELECT NUMTODSINTERVAL((v_seconds*sp_link.proportionOfDur),'SECOND') INTO sp_link.duration FROM dual;
            sp_link.obs_id:= :NEW.obs_id;
            sp_link.obs_time:= :NEW.obs_time;
            sp_link.cost:= sp_link.proportionOfDur*v_seconds;
            UPDATE STUDY_AREA_CLIPPED SAC SET COST = (((sac.cost * sac.n_rt_obs) + sp_link.cost) / (sac.n_rt_obs + 1))
            where toid = sp_link.toid and toid_direction = sp_link.toid_dir;
            update study_area_clipped set n_rt_obs = n_rt_obs+1 where study_area_clipped.toid=sp_link.toid and
            study_area_clipped.toid_direction = sp_link.toid_dir;
            INSERT INTO PROCESSED_EVENTS(obs_id,procedure_id,toid,obs_time,duration,cost,toid_direction,insertion_time,
            request_time)
            VALUES(sp_link.obs_id,:NEW.PROCEDURE_ID,sp_link.toid,sp_link.obs_time,sp_link.duration,sp_link.cost,
            sp_link.toid_dir,systimestamp,v_reqtime);
          END LOOP;
            dbms_output.put_line('total cost' || total_cost);

            UPDATE STUDY_AREA_CLIPPED SAC SET COST= (((sac.cost * sac.n_rt_obs) + ((v_prevLength/total_cost)*v_seconds))
            /(sac.n_rt_obs+1)) where toid = v_previousTOID and toid_direction = v_dir_s;
            UPDATE study_area_clipped set n_rt_obs = (n_rt_obs+1) where study_area_clipped.toid=v_previousTOID and
            study_area_clipped.toid_direction = v_dir_s;
            INSERT INTO PROCESSED_EVENTS(obs_id,procedure_id,toid,obs_time,insertion_time,request_time) VALUES(:NEW.OBS_ID,
            :NEW.PROCEDURE_ID,:NEW.TOID,:NEW.OBS_TIME,systimestamp,v_reqtime);
          --3. CLEAN UP
          SDO_NET_MEM.NETWORK_MANAGER.DROP_NETWORK('TEMP_NETWORK');
          delete from user_SDO_NETWORK_METADATA WHERE NETWORK='TEMP_NETWORK';
          RETURN;
      ELSE
          dbms_output.put_line('doing nothing - vduration is <0 or null');
          INSERT       INTO PROCESSED_EVENTS        (          obs_id,          procedure_id,          toid,
          obs_time,         duration,          cost,insertion_time,request_time          )          VALUES
          (     :NEW.OBS_ID,  :NEW.PROCEDURE_ID,  :NEW.TOID,   :NEW.OBS_TIME, v_duration, v_seconds , systimestamp,v_reqtime);
      END IF;
    ELSE
     dbms_output.put_line('direction calcd');
     dbms_output.put_line(v_previousTOID);
     --otherwise insert direction,duration and cost...
     INSERT       INTO PROCESSED_EVENTS  (     obs_id,     procedure_id,   toid, obs_time, duration,
     TOID_DIRECTION,cost,insertion_time,request_time  )    VALUES   (     :NEW.OBS_ID,:NEW.PROCEDURE_ID,:NEW.TOID,
     :NEW.OBS_TIME,            v_duration,           v_dir,           v_seconds,systimestamp,v_reqtime          );
    END IF;
  END IF;
END IF;
  --begin update rt cost trigger
  SELECT HOUR
  INTO v_costSetTo
  FROM COST_SET_TO
  WHERE ID=1;
  SELECT EXTRACT(hour FROM systimestamp)+1 INTO v_systemHour FROM dual;
--IF(v_costSetTo != v_systemHour) THEN

  --dbms_output.put_line('costset to != system hour');
  CASE v_systemHour
  WHEN 5 THEN
    UPDATE study_area_clipped SET COST = HRN_5;
  WHEN 6 THEN
    UPDATE study_area_clipped SET COST = HRN_6;
  WHEN 7 THEN
    UPDATE study_area_clipped SET COST = HRN_7;
  WHEN 8 THEN
    UPDATE study_area_clipped SET COST = HRN_8;
  WHEN 9 THEN
    UPDATE study_area_clipped SET COST = HRN_9;
  WHEN 10 THEN
    UPDATE study_area_clipped SET COST = HRN_10;
```



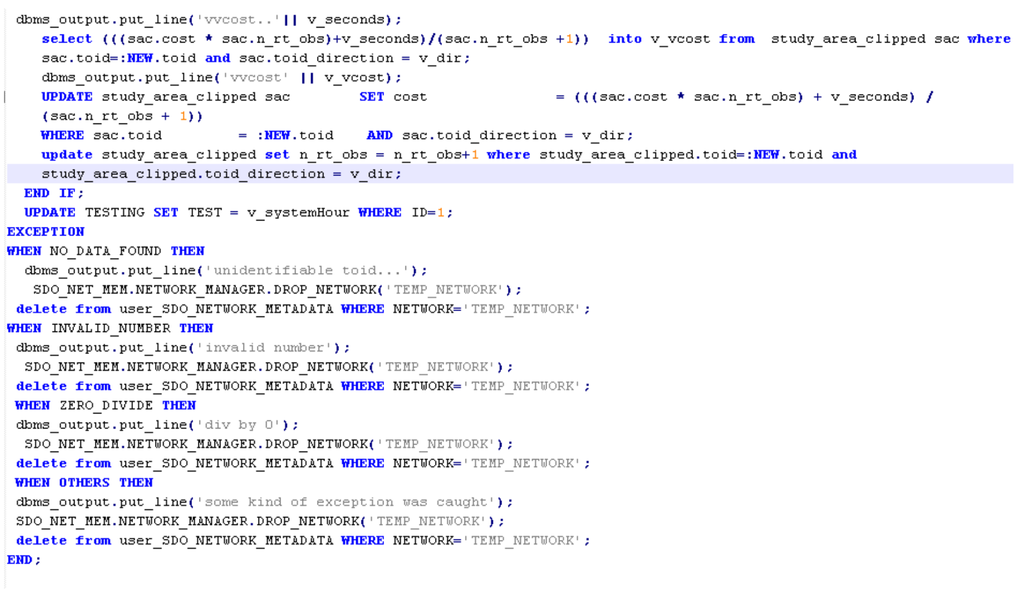
```
  WHEN 11 THEN
    UPDATE study_area_clipped SET COST = HRN_11;
  WHEN 12 THEN
    UPDATE study_area_clipped SET COST = HRN_12;
  WHEN 13 THEN
    UPDATE study_area_clipped SET COST = HRN_13;
  WHEN 14 THEN
    UPDATE study_area_clipped SET COST = HRN_14;
  WHEN 15 THEN
    UPDATE study_area_clipped SET COST = HRN_15;
  WHEN 16 THEN
    UPDATE study_area_clipped SET COST = HRN_16;
  WHEN 17 THEN
    UPDATE study_area_clipped SET COST = HRN_17;
 WHEN 18 THEN
   UPDATE study_area_clipped SET COST = HRN_18;
 WHEN 19 THEN
   UPDATE study_area_clipped SET COST = HRN_19;
 WHEN 20 THEN
   UPDATE study_area_clipped SET COST = HRN_20;
 WHEN 21 THEN
   UPDATE study_area_clipped SET COST = HRN_21;
 WHEN 22 THEN
   UPDATE study_area_clipped SET COST = HRN_22;
WHEN 23 THEN
  UPDATE study_area_clipped SET COST = HRN_23;
WHEN 0 THEN
  UPDATE study_area_clipped SET COST = HRN_24;
ELSE
  UPDATE study_area_clipped SET COST = HRN_4;
 END CASE;
UPDATE COST_SET_TO SET HOUR = v_systemHour WHERE id =1;
UPDATE STUDY_AREA_CLIPPED SET n_rt_obs = 0;
  --now loop through all the processed events observed within the current hour..
FOR r_processed_events_tuple IN processed_events_tuple
```



```
LOOP
populate toid_record with toid and direction
    toid_record.toid     := r_processed_events_tuple.toid;
    toid_record.toid_dir := r_processed_events_tuple.toid_direction;
    --check obs is directed...
    IF toid_record.toid_dir = va_b OR toid_record.toid_dir=va_f THEN
      --find no. of obs for this toid..
      SELECT COUNT(*)  INTO v_NoObs    FROM processed_events   WHERE obs_time  > v_nowMinusOneHOUR_TIMESTAMP
      AND toid_record.toid    =processed_events.toid
      AND toid_record.toid_dir=processed_events.toid_direction;
      update sac with n-rt-obs
      UPDATE study_area_clipped sac
      SET n_rt_obs            = v_NoObs
      WHERE sac.toid          = toid_record.toid
      AND sac.toid_direction = toid_record.toid_dir;
      --now feed the obs into a record table, sum them and update sac..
      --feed
      FOR i IN 1..v_NoObs
      LOOP
        SELECT cost INTO v_proc_cost     FROM (SELECT cost     FROM
            (SELECT rownum r, cost  FROM processed_events
            WHERE obs_time          > v_nowminusonehour_timestamp
            AND toid_record.toid    =processed_events.toid
            AND toid_record.toid_dir=processed_events.toid_direction )
          WHERE r=i  );
        obs_table(i):=v_proc_cost;
      END LOOP;
      l_row:=obs_table.FIRST;
      --sum
      WHILE(l_row IS NOT NULL)
      LOOP
        v_sum_cost:= v_sum_cost + obs_table(l_row);
        l_row     :=obs_table.NEXT(l_row);
      END LOOP;
      --update
```



```
      UPDATE study_area_clipped sac
      SET cost                = v_sum_cost/v_NoObs
      WHERE sac.toid          =toid_record.toid
      AND sac.toid_direction = toid_record.toid_dir;
      v_sum_cost:=0;
      v_NoObs    :=0;
    END IF;
  END LOOP;
    --HOUR IS UP TO DATE ALREADY...JUST UPDATE THIS LATEST OBSERVATION
 ELSE
```



```
 dbms_output.put_line('vvcost..'|| v_seconds);
    select (((sac.cost * sac.n_rt_obs)+v_seconds)/(sac.n_rt_obs +1))  into v_vcost from  study_area_clipped sac where
    sac.toid=:NEW.toid and sac.toid_direction = v_dir;
    dbms_output.put_line('vvcost' || v_vcost);
    UPDATE study_area_clipped sac        SET cost               = (((sac.cost * sac.n_rt_obs) + v_seconds) /
    (sac.n_rt_obs + 1))
    WHERE sac.toid          = :NEW.toid    AND sac.toid_direction = v_dir;
    update study_area_clipped set n_rt_obs = n_rt_obs+1 where study_area_clipped.toid=:NEW.toid and
    study_area_clipped.toid_direction = v_dir;
  END IF;
  UPDATE TESTING SET TEST = v_systemHour WHERE ID=1;
EXCEPTION
WHEN NO_DATA_FOUND THEN
  dbms_output.put_line('unidentifiable toid...');
   SDO_NET_MEM.NETWORK_MANAGER.DROP_NETWORK('TEMP_NETWORK');
 delete from user_SDO_NETWORK_METADATA WHERE NETWORK='TEMP_NETWORK';
WHEN INVALID_NUMBER THEN
 dbms_output.put_line('invalid number');
  SDO_NET_MEM.NETWORK_MANAGER.DROP_NETWORK('TEMP_NETWORK');
 delete from user_SDO_NETWORK_METADATA WHERE NETWORK='TEMP_NETWORK';
 WHEN ZERO_DIVIDE THEN
 dbms_output.put_line('div by 0');
  SDO_NET_MEM.NETWORK_MANAGER.DROP_NETWORK('TEMP_NETWORK');
 delete from user_SDO_NETWORK_METADATA WHERE NETWORK='TEMP_NETWORK';
 WHEN OTHERS THEN
 dbms_output.put_line('some kind of exception was caught');
 SDO_NET_MEM.NETWORK_MANAGER.DROP_NETWORK('TEMP_NETWORK');
 delete from user_SDO_NETWORK_METADATA WHERE NETWORK='TEMP_NETWORK';
END;
```

Appendix J PL/pgSQL GPS Vehicle Track Data Loading Procedure

This procedure details the process of transforming a comma separated text file containing GPS observations from several vehicles, into a number of PostGIS database tables, one for each vehicle. The input format of the .CSV file is detailed as follows:

Input File Format

```
DateTime,vehicle,vehicletype,eventid,fix,latitude,longitude,bearing,sp
eed,inputs,geofence,status
21/09/2010   06:56:46,CS3675,COMPACT   SWEEPER,240,0,54.9728883333333,-
1.57299666666667,0,0,8,0,2
21/09/2010   06:57:02,CS3675,COMPACT   SWEEPER,16,0,54.9728883333333,-
1.57299666666667,0,0,8,128,2
```

Prerequisites

1) Microsoft Excel (2003)
2) PostGIS database (PostgreSQL 8.3 / PostGIS 1.5.1)

Procedure

1) Open the .CSV file in Microsoft Excel and remove the header row. Save it as a tab delimited text file: ‘observations.txt’. Note that due to file size limits in Microsoft Excel, this will only save the first 64536 records. Thus it is necessary to open the original file again, delete the first 64536 records and the header row, and save the remaining records as another tab delimited text file: ‘*observations_part2.txt’*

2) Create a PostGIS database by typing the following command from the PostgreSQL shell:

```
C:\postgresql\8.3\bin> createdb –T template_postgis all_gps
```

3) Login to the database from he PostgreSQL shell and create an observation table:

```
C:\postgresql\8.3\bin> psql all_gps
```

```
all_gps=# create table observations(dateTime timestamptz, vehicle varchar(30), vehicleType varchar(50), eventid integer, fix integer, latitude varchar(50), longitude varchar(50), bearing integer, speed integer, inputs integer, geofence integer, status integer);
```

4) Load observations into the table:

```
all_gps=# \copy observations from ‘<path to observations.txt’
all_gps=# \copy observations from ‘<path to observations_part2.txt’
```

5) Create a Primary Key:

```
all_gps=# ALTER TABLE observations ADD fid serial primary key;
```

6) Add columns for geometry

```
all_gps=# ALTER TABLE observation ADD numlat double precision;
all_gps=# ALTER TABLE observation ADD numlong double precision;
all_gps=# ALTER TABLE observation ADD etrs_geom double precision;
all_gps=# ALTER TABLE observation ADD wgs_geom double precision;
```

7) Cast Latitude / Longitude to numeric type

```
UPDATE observations SET numlat = cast(latitude as double precision);
UPDATE observations SET numlong = cast(longitude as double precision);
```

8) Convert to Geometry and transform

```
UPDATE OBSERVATIONS SET etrs_geom = st_setsrid(st_point(numlong,numlat),4258);
UPDATE observations SET wgs_geom = transform(etrs_geom,4326);
```

9) Remove erroneous observations

```
DELETE FROM observations WHERE numlat < 53 or numlat > 56;
DELETE FROM observations WHERE numlong < -3 or numlong > 0;
```

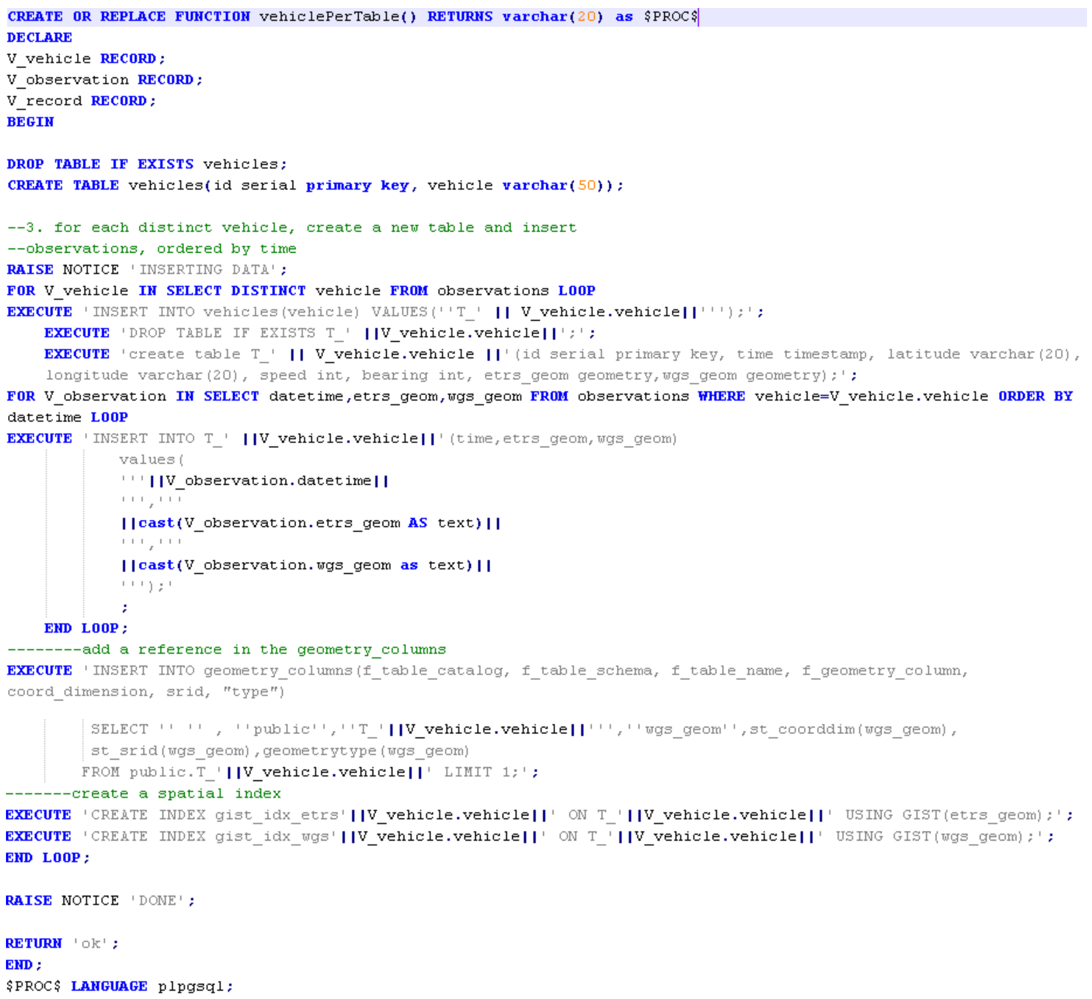
10) Register the geometry column

```
INSERT INTO geometry_columns(f_table_catalog, f_table_schema, f_table_name, f_geometry_column, coord_dimension, srid, "type")
SELECT '', 'public', 'observations', 'wgs_geom', ST_CoordDim(wgs_geom), ST_SRID(wgs_geom), GeometryType(wgs_geom)
FROM public.observations LIMIT 1;
```

11) Rename tubples that have a whitespace in vehicle name

```
SELECT DISTINCT VEHICLE FROM OBSERVATIONS WHERE VEHICLE LIKE
'% %';
UPDATE OBSERVATIONS SET VEHICLE = <name without space> WHERE
VEHICLE=<result of previous query>;
```

12) From PgAdmin console open and run the following script to create a function



```
CREATE OR REPLACE FUNCTION vehiclePerTable() RETURNS varchar(20) as $PROC$
DECLARE
V_vehicle RECORD;
V_observation RECORD;
V_record RECORD;
BEGIN

DROP TABLE IF EXISTS vehicles;
CREATE TABLE vehicles(id serial primary key, vehicle varchar(50));

--3. for each distinct vehicle, create a new table and insert
--observations, ordered by time
RAISE NOTICE 'INSERTING DATA';
FOR V_vehicle IN SELECT DISTINCT vehicle FROM observations LOOP
EXECUTE 'INSERT INTO vehicles(vehicle) VALUES(''T_' || V_vehicle.vehicle||''');';
    EXECUTE 'DROP TABLE IF EXISTS T_' ||V_vehicle.vehicle||';';
    EXECUTE 'create table T_' || V_vehicle.vehicle ||'(id serial primary key, time timestamp, latitude varchar(20),
    longitude varchar(20), speed int, bearing int, etrs_geom geometry,wgs_geom geometry);';
FOR V_observation IN SELECT datetime,etrs_geom,wgs_geom FROM observations WHERE vehicle=V_vehicle.vehicle ORDER BY
datetime LOOP
EXECUTE 'INSERT INTO T_' ||V_vehicle.vehicle||'(time,etrs_geom,wgs_geom)
            values(
            '''||V_observation.datetime||
            ''','''
            ||cast(V_observation.etrs_geom AS text)||
            ''','''
            ||cast(V_observation.wgs_geom as text)||
            ''');'
            ;
    END LOOP;
--------add a reference in the geometry_columns
EXECUTE 'INSERT INTO geometry_columns(f_table_catalog, f_table_schema, f_table_name, f_geometry_column,
coord_dimension, srid, "type")

          SELECT '' '' , ''public'',''T_'||V_vehicle.vehicle||''',''wgs_geom'',st_coorddim(wgs_geom),
          st_srid(wgs_geom),geometrytype(wgs_geom)
         FROM public.T_'||V_vehicle.vehicle||' LIMIT 1;';
-------create a spatial index
EXECUTE 'CREATE INDEX gist_idx_etrs'||V_vehicle.vehicle||' ON T_'||V_vehicle.vehicle||' USING GIST(etrs_geom);';
EXECUTE 'CREATE INDEX gist_idx_wgs'||V_vehicle.vehicle||' ON T_'||V_vehicle.vehicle||' USING GIST(wgs_geom);';
END LOOP;

RAISE NOTICE 'DONE';

RETURN 'ok';
END;
$PROC$ LANGUAGE plpgsql;
```

13) Run the function

```
all_gps=# SELECT vehiclePerTable();
```

14) Finished

The PostGIS database 'all_gps' should now contain all the GPS observations in separate tables, one per vehicle. The observations are all time ordered. Each vehicle is also listed in table 'vehicles'.

Appendix K Supervised Classification of a multispectral (XS) SPOT-1 HRV image of South East London

Prerequisites

1) ERDAS Imagine v9.3
   http://www.erdas.com

2) Multispectral image to be classified

3) Background mapping of the area represented in the image *(http://edina.ac.uk/digimap)*

4) GDAL library (FWTools)
   http://fwtools.maptools.org/

Procedure

1) Geometric Correction
   a. Download Ordnance Survey 1:25000 raster mapping in GeoTIFF format from www.edina.ac.uk/digimap for the entire area covered by the image.
   b. Load the image to be processed in Erdas Imagine in one viewer, and load each of the Ordnance Survey tiles in another viewer
   c. From the Imagine toolbar select Dataprep >> Geometric Image Correction
   d. Follow the prompts to select the viewer containing the classified image
   e. Select *POLYNOMIAL* from the model properties dialogue box and then close the box.
   f. Select *Existing viewer* when prompted for where to select reference control points from and click in the viewer containing the Ordnance Survey Data.
   g. Select at least 12 ground control points from the map data and identify the corresponding location on the image. Ideally the points should be spread both around the edges of the image and in the centre.
   h. Click on Display Model Properties and change the polynomial order to *2*.
   i. Click *Resample Image* and select *Nearest Neighbour* from the following dialogue box

j. Open the corrected image in the same viewer as the map data and use the Utility >> Blend tool to ensure that it is a good fit. If necessary repeat this process until a good fit has been achieved.

2) Classification

a. Define Land-cover Classes

It was opted to use six land-cover classes in this study that represent the major spectral classes in the image. These are listed as follows:

a. water
b. crops
c. forest
d. grass
e. small buildings
f. large buildings

b. Select Training Areas

Open the geometrically corrected image in Erdas Imagine and use the Area of Interest (AOI) tool and the Signature Editor to manually select training areas corresponding to each of the land-cover classes defined in step 1. This is achieved by cross referencing features on the image with features on an Ordnance Survey base map of the area. The size of each training area should be at least 30p pixels per class where p is the number of spectral bands. It should be noted that for this image the blue band was stripped out as it was of a poor quality, so only the green, red and near-infrared bands were used.

c. Perform Classification

Select the Supervised Classification tool from the Classification menu in Erdas and specify the image to be classified, the signature file containing the training area samples and an output location for the classified image. Ensure 'Maximum Likelihood' is selected in the Parametric Rule box and, 'None' is selected in the Non Parametric Rule box and click OK to start the classification.

d. Accuracy Assessment

An assessment of classification accuracy must be performed to ensure that the resulting classification is valid. The following procedure details how to use the Imagine Accuracy Assessment tool:

i) From the Imagine toolbar click on Classifier and then Accuracy Assessment.
ii) Click on File >> Open and select the classified and geometrically corrected image.
iii) Click on Edit >> Add/generate random points
iv) Open the Ordnance Survey map data in a new viewer and from the accuracy assessment window select View >> Select Viewer and click on the viewer containing the map data.
v) Click View >> Show All to display all of the random points on the map data viewer. Now click Edit >> Show class values to show the class each random point is assigned to.
vi) Now from the map data viewer select Utility >> Enquire Box and for each random point copy the coordinates into the enquire box to find the random test point on the map. Enter the code corresponding to the land-cover class actually found at the location of each test point.
vii) When complete click on Report >> Accuracy Report to generate a confusion matrix and calculate the Kappa co-efficient.

3) Format Conversion

The GDAL open-source raster translation library was used to convert the image from ERDAS Imagine proprietary format (.img) to NetCDF format. During this conversion some of the empty grid surrounding the image was also stripped out to reduce the amount of redundant processing required. It was opted to convert into a CF convention NetCDF file. Although GMT (http://gmt.soest.hawaii.edu/) compatible NetCDF is better supported by GDAL than NetCDF, reading and writing such files using the Java API proved to be more problematic. Thus the following translation command was used:

```
gdal_translate –ot Byte –of NetCDF –srcwin 0 0 4200 3000
c:\temp\geo_corrected.img c:\temp\geo_corrected.nc
```

Appendix L Land-use Templates

Key:	L	large building	S	small building
	G	grass	C	crops
	W	water	F	forest

L	L	S	S	S	L	L	L	S
L	L	L	S	L	L	L	S	S
L	L	S	S	L	L	L	S	S
S	S	S	S	S	S	L	S	L
L	S	S	S	L	L	L	L	L
S	S	L	S	L	L	L	L	L
S	S	S	S	L	S	L	L	L
S	S	S	L	S	S	S	S	S
S	S	S	S	S	L	S	S	S

1. Commercial / Industrial

F	F	F	S	S	S	S	S	S
S	F	F	S	S	S	S	S	G
S	S	F	S	S	S	S	S	G
S	S	S	S	S	S	S	S	S
S	S	S	S	S	S	S	S	S
S	S	S	S	S	S	S	S	L
S	S	S	G	S	S	S	S	S
S	S	G	G	S	L	S	S	S
S	G	G	S	L	L	S	S	S

2. Low Density Residential

S	S	S	S	S	S	S	S	S
S	S	S	S	S	S	S	S	S
S	S	S	S	S	G	L	L	S
S	S	S	S	S	S	S	S	S
S	S	S	S	L	L	L	L	L
S	S	S	L	L	L	S	S	L
S	S	L	S	S	S	L	L	L
S	S	G	S	L	L	L	S	L
S	L	S	S	S	S	S	S	L

3. Medium Density Residential

C	C	C	C	C	C	C	C	C
C	F	C	C	C	C	C	C	C
F	F	F	C	C	C	C	C	C
F	F	C	C	C	C	C	F	C
C	C	C	C	C	C	C	C	G
C	C	C	C	C	C	C	C	F
F	C	C	C	C	C	C	C	C
C	C	C	C	C	C	C	C	C
C	C	C	C	C	C	C	C	C

4. Arable

F	F	F	F	C	G	C	F	F
C	C	G	G	G	G	G	G	G
G	G	G	G	G	G	G	G	G
G	G	G	G	G	G	G	G	G
G	G	G	G	G	G	G	G	S
G	G	G	G	G	G	G	G	F
G	G	G	F	G	G	C	C	C
G	G	G	G	G	G	F	F	G
G	G	G	G	G	G	F	F	G

5. Pasture

W	W	W	W	W	W	W	W	W
W	W	W	W	W	W	W	W	W
W	W	W	W	W	W	W	W	W
W	W	W	W	W	W	W	W	W
W	W	W	W	W	W	W	W	W
W	W	W	W	W	W	W	W	W
W	W	W	W	W	W	W	W	W
W	W	W	W	W	W	W	W	W
W	W	W	W	W	W	W	W	W

6. Water

F	F	F	F	F	F	F	G	F

F	F	F	F	F	F	F	G	F
F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F
F	F	G	F	F	F	F	C	C
S	G	F	F	F	F	F	F	F
S	G	C	F	F	F	F	F	F
S	G	S	F	F	F	F	F	F
F	G	G	G	F	F	F	F	F

7. Woodland

G	G	G	G	G	G	F	G	G
G	G	G	G	G	G	F	G	G
G	G	G	G	G	G	F	G	G
G	G	G	G	G	G	F	G	G
G	G	G	G	G	G	C	G	G
G	G	S	S	G	G	G	G	G
G	G	S	S	G	G	G	G	G
C	G	G	G	C	C	G	G	G
G	G	G	F	C	C	C	G	G

8. Wasteland

Chapter 9 References

Abadi, D. J. (2009) 'Data Management in the Cloud: Limitations and Opportunities', *Data Engineering,* 32, (1), pp. 3-12.

Abadi, D. J., Ahmed, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S. (2005) 'The Design of the Borealis Stream Processing Engine'.*2nd Conference on Innovative Data Systems Research (CIDR'05).* Asilomar, California, USA:ACM.

Abbas, A. (2004) *Grid Computing: A Practical Guide to Technology and Applications.* Charles River Media: London.

Abel, D. J., Volker, J. G., Taylor, K. L. and Xiaofang, Z. (1999) 'SMART: Towards Spatial Internet Marketplaces', *Geoinformatica,* 3, (2), pp. 141-164.

Acache, J. (2007) *The Full Picture*. Group on Earth Observations [Online]. Available at: http://www.earthobservations.org/documents /the_full_picture.pdf (04/05/2009).

Adler, D. W. (2001) 'IBM DB2 Spatial Extender - Spatial data within the RDBMS'.*27th VLDB Conference.* Roma, Italy,

Aggarwal, A., Chazelle, B., Guibas, L., O'Dunlaing, C. and Yap, C. (1988) 'Parallel computational geometry', *Algorithmica,* 3, pp. 293-327.

Akinci, B., Hendrickson, C. and Karaesmen, I. (2003) 'Exploiting Motor Vehicle Information and Communications Technology for Transportation Engineering', *Journal of Transportation Engineering* 129, (5).

Alexander, J., Box, D. and Cabrerra, L. F. (2006) *Web Services Transfer (WS-Transfer).* W3C (http://www.w3.org/TR/2011/CR-ws-transfer-2011042).

Allcock, W., Bester, J., Bresnahan, J., Meder, S., Plaszczak, P. and Tuecke, S. (2003) *GridFTP: Protocol Extensions to FTP for the Grid.* Open Grid Forum (GFD.20).

Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raica, I. and Foster, I. (2005) 'The globus striped GridFTP framework and server'.*Supercomputing, SC05.* Seattle, WA, USA:ACM.

Almasi, G. S. and Gottlieb, A. (1990) 'Review of Highly parallel computing', *IBM Systems Journal,* 29, (1), pp. 165-166.

Aloisio, G., Cafaro, M., Epicoco, I., Fiore, R., Lezzi, D., Mirto, M. and Mocavero, S. (2005) 'iGrid, A Novel Grid Information Service '.*First European Grid Conference.*

Aloisio, G., Conte, D., Elefante, C., Epicoco, I., Marra, P. G., Mastrantonio, G. and Quarta, G. (2006) 'SensorML for Grid Sensor Networks'*, 2006 International Conference on Grid Computing Applications.* Las Vegas, Nevada, USA, SensorML for Grid Sensor Networks: CSREA Press, pp. 141-146.

Amdahl, G. (1967) 'The validity of the single processor approach to achieving large scale computing capabilities'*, AFIPS / Spring Joint Computer Conference.* The validity of the single processor approach to achieving large scale computing capabilities: pp. 483-485.

Anderson, J., Ahmed, R., Bourn, H. and McGraffin, R. (2008) *Tyne & Wear Air Quality Delivery Plan.* Newcastle upon Tyne: Planning and Transportation Dept, Newcastle City Council

Andrews, C. J. (2007) *Emerging Technology: AJAX and GeoJSON*. Available at: http://www.directionsmag.com/article.php?article_id=2550&trv=1 (Accessed: 29/10/2007).

Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S. and Xu, M. (2007) *Web Services Agreement Specification (WS-Agreement).* Open Grid Forum (GFD-R-P.107).

Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, A., Pulsipher, D. and Savva, A. (2005) *Job Submission Description Language (JSDL) Specification, Version 1.0.* Open Grid Forum (GFD-R.056).

Antonioletti, M., Collins, B., Krause, A., Laws, S., Magowan, J., Malaika, S. and Paton, N. (2006) *Web Services Data Access and Integration.* Open Grid Forum (GFD 75, GFD 76).

Arasu, A., Babu, S. and Widom, J. (2006) 'The CQL continuous query language: semantic foundations and query execution', *The VLDB Journal,* 15, (2), pp. 121-142.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. (2009) *Above the*

Clouds: A Berkeley View of Cloud Computing. Berkeley, USA: University of California at Berkeley (Technical Report No. UCB/EECS-2009-28).

Armstrong, M. P. and Densham, P. J. (1992) 'Domain Decomposition for Parallel Processing of Spatial Problems', *Computers, Environment and Urban Systems,* 16, pp. 497-513.

Armstrong, M. P. and Marciano, R. (1993) 'Parallel Spatial Interpolation*', Auto-Carto 11.* Bethesda, MD, USA, Parallel Spatial Interpolation: ASPRS and ACSM, pp. 414-423.

Atkinson, M., DeRoure, D., Dunlop, A., Fox, G., Henderson, P., Hey, T., Paton, N., Newhouse, S., Parastatidis, S., Trefethen, A., Watson, P. and Webber, J. (2004) 'Web Service Grids: An Evolutionary Approach', *Concurrency & Computation: Practice and Experience,* 17, pp. 377-389.

Babcock, B., Babu, S., Datar, M., Motwani, R. and Widom, J. (2002) 'Models and Issues in Data Stream Systems*', 21st Symposium on Principles of Database Systems.* Madison, Wisconsin, USA, Models and Issues in Data Stream Systems: ACM SIGACT-SIGMOD-SIGART, pp. 1-16.

Babu, S. and Widom, J. (2001) 'Continuous Queries over Data Streams', *ACM SIGMOD Record,* 30, (3), pp. 109-120.

Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederiskson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., Weeratunga, S. and (1994) *The NAS Parallel Benchmarks, .* NASA (RNR-94-007).

Ballinger, K., Ehnebuske, D., Ferris, C., Gudgin, M., Liu, C., Nottingham, M. and Yendluri, P. (2006) *Basic Profile Version 1.1.* Web Services Interoperability Organization (http://www.ws-i.org/Profiles/BasicProfile-1.1-2006-04-10.html).

Baranski, B. (2008) 'Grid Computing Enabled Web Processing Service*', 6th Geographic Information Days (GI-Days 2008).* Institut fur Geoinformatik, Munster, Grid Computing Enabled Web Processing Service: IfGIprints

Baranski, B. and Schäffer, B. (2010) 'Towards Service Level Agreements in Spatial Data Infrastructures'.*GSDI 12 World Conference.* Singapore,

Baranski, B., Schäffer, B. and Redweik, R. (2009) 'Geoprocessing in the Clouds*', Free and Open Source Software for Geospatial.* Sydney, Australia, Geoprocessing in the Clouds

Barcelló, J., Ferrer, J., Garcia, D., Florian, M. and Le Saux, E. (1998) 'Parallelization of microscopic traffic simulation for ATT systems', in Marcotte, P. and Nguyen, S.(eds) *Equilibrium and Advanced Transportation Modelling.* Kluwer Academic Publishers: Dordrecht, pp. 1-26.

Barcelló, J. and Grau, R. (1993) 'PACKSIM: An experience in using traffic simulation in a demand responsive traffic control system*', XIII World Conference on Operations Research.* Lisbon, Portugal, PACKSIM: An experience in using traffic simulation in a demand responsive traffic control system

Barnsley, M. J. and Barr, S. L. (1996) 'Inferring Urban Land Use from Satellite Sensor Images Using Kernel-Based Spatial Reclassification', *Photogrammetric Engineering & Remote Sensing,* 62, (8), pp. 949-958.

Baumann, P. (2010) *OGC WCS 2.0 Interface Standard - Core.* Open Geospatial Consortium Inc. (09-110r3).

Beard, K. (2007) 'Modelling Change in Space & Time: An event based approach', in Drummond, J.(ed), *Dynamic and Mobile GIS: Investigating Changes in Space and Time.*

Bermudez, L., Bogden, P., Bridger, E., Cook, T., Galvarino, C., Creager, G., Forrest, D. and Graybeal, J. (2009) 'Web Feature Service (WFS) and Sensor Observation Service (SOS) comparison to publish time series data*', International Symposium on Collaborative Technologies and Systems.* Web Feature Service (WFS) and Sensor Observation Service (SOS) comparison to publish time series data: Baltimore, Maryland, USA

Bethel, E. W., Humphreys, G., Paul, B. and Brederson, J. D. (2003) 'Sort-first, distributed memory parallel visualization and rendering', *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pp. 41-50.

Black, M. and Smith, R. G. (2003) 'Electronic monitoring in the criminal justice system', *Trends and Issues in Crime and Justice,* 254, pp. 241-260.

Blower, J. D. (2010) 'GIS in the cloud: implementing a Web Map Service on Google App Engine*', COM.Geo.* Washington D. C, USA, GIS in the cloud: implementing a Web Map Service on Google App Engine: ACM

Blunck, H., Godsk, T., Gronbaek, K., Kjaergaard, M. B., Jensen, J. L., Scharling, T., Toftkjaer, T. and Schougaard, K. R. (2010) 'PerPos: A Platgorm Providing Cloud Services for Pervasive Positioning'*, COM. Geo.* Washington D. C., USA, PerPos: A Platgorm Providing Cloud Services for Pervasive Positioning: ACM

Blythe, P. T., Bell, M. C., Sharif, B. and Watson, P. (2006) 'Pervasive Environmental Monitoring using Smartdust: The MESSAGE Project'. *The Institute of Engineering & Technology Seminar on RFID and Electronic Vehicle Identification.* Newcastle, UK:IEEE.

Bonnet, P., Gehrke, J. E. and Seshadri, P. (2000) 'Querying the Physical World', *IEEE Personal Communication,* 7, (5), pp. 10-15.

Bonnet, P. and Seshadri, P. (2000) 'Device Database Systems'.*16th International Conference on Data Engineering.* San Diego, California, USA,

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D. (2003) *Web Services Architecture: working draft.* W3C

Böse, M., Erdik, M. and Wenzel, F. (2007) 'A New Approach to Earthquake Early Warning', in Gasparini, P., Manfredi, G. and Zschau, J.(eds) *Earthquake Early Warning Systems.* Springer Berlin Heidelburg.

Bose, P., Krizanc, D., Langerman, S. and Morin, P. (2003) 'Asymmetric Communication Protocols via Hotlink Assignments', *Theory of Computing Systems,* 36, pp. 655-661.

Botts, M., Percivall, G., Reed, C. and Davidson, J. (2006) *OGC Sensor Web Enablement: Overview and High Level Architecture.* Open Geospatial Consortium (OGC 06-050R2).

Botts, M. and Robin, A. (2007) *OpenGIS Sensor Model Language (SensorML) Implementation Specification.* Open Geospatial Consortium (OGC-05-086).

Bowler, K. C., Bruce, A. D., Kenway, R. D., Pawley, G. S. and Wallace, D. J. (1987) 'Exploiting Highly Concurrent Computers for Physics', *Physics Today,* 40, (10), pp. 40-48.

Box, D., Cabrerra, L. F., Critchley, C., Curbera, F., Ferguson, D., Graham, S., Hull, D., Kakivaya, G., Lewis, A., Lovering, B., Niblett, P., Orchard, D., Samdarshi, S., Schlimmer, J., Sedukhin, I., Shewchuk, J., Weerawarana,

S. and Wortendyke, D. (2006) *Web Services Eventing (WS-Eventing).* W3C (http://www.w3.org/Submission/2006/SUBM-WS-Eventing-20060315/).

Box, D., Christensen, E., Curbera, F., Ferguson, D., Frey, J., Hadley, M., Kaler, C., Langworthy, D., Leymann, F., Lovering, B., Lucco, S., Millet, S., Mukhi, N., Nottingham, M., Orchard, D., Shewchuk, J., Sindambiwe, E., Storey, T., Weerawarana, S. and Winkler, S. (2004) *Web Services Addressing (WS-Addressing).* W3C (http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/).

Brackstone, M., Fisher, G. and McDonald, M. (2001) 'The use of probe vehicles on motorways, some emperical observations'.*World Congress on Intelligent Transport Systems.* Sydney, Australia,

Brakatsoulas, S., Pfoser, D., Salas, R. and Wenk, C. (2005) 'On Map-Matching Vehicle Tracking Data'.*31st Very Large Data Bases (VLDB) Conference.* Trondheim, Norway,

Brauner, J., Foerster, T., Schaeffer, B. and Baranski, B. (2009) 'Towards a Research Agenda for Geoprocessing Services'.*12th AGILE Conference on Geographic Information Science.* Leibniz Universitat, Hannover, Germany,

Braunl, T., Feyrer, S., Reinhardt, M. and Wolfgang, R. (2001) *Parallel Image Processing.* Springer: Berlin.

Brito, A. (2008) 'Optimistic Parallelization Support for Event Stream Processing Systems*', Middleware Doctoral Symposium 2008* Leuven, Belgium, Optimistic Parallelization Support for Event Stream Processing Systems: ACM, pp. 7-12.

Broering, A., Stasch, C. and Echterhoff, J. (2010) *OGC SOS 2.0 Interface Standard.* Open Geospatial Consortium Inc. (OGC 10-037).

Bruce, R. A. A., Chapple, S. R., MacDonald, N. B. and Trew, A. S. (1993) 'CHIMP and PUL: Support for Portable Parallel Computing'.*4th Annual Conference of the Meiko User Society.* Southampton,UK,

Budhathoki, N. R., Bruce, B. and Nedovic-Budie, Z. (2008) 'Reconceptualizing the role of the user of spatial data infrastructure', *GeoJournal,* 72, pp. 149-160.

Burkhart, H., Korn, C. F., Gutzwiller, S., Ohnacker, P. and Waser, S. (1993) *BACS: Basel Algorithm Classification Scheme.* Basel, Switzerland: University of Basel, Switzerland (Technical Report 93-03).

Burrough, P. A. and McDonnell, R. (1998) *Principles of Geographic Information Systems.* Oxford University Press, UK.

Burton, A. M., Miller, P., Bruce, V., Hancock, P. J. B. and Henderson, Z. (2001) 'Human and automatic face recognition: a comparison across image formats', *Vision Research,* 41, pp. 3185-3195.

Buyya, R., Yeo, C. S. and Venugopal, S. (2008) 'Market-oriented cloud computing: Vision, hype and reality for delivering IT services as computing utilities*', 10th IEEE International Conference on High Performance Computing and Communications.* Dalian, China, Market-oriented cloud computing: Vision, hype and reality for delivering IT services as computing utilities: IEEE, pp. 5-16.

Cameron, G. D. B. and Duncan, C. I. D. (1996) 'PARAMICS - parallel microscopic simulation of road traffic', *Journal of Supercomputing,* 10, (1), pp. 25.

Cardellini, V., Colajanni, M. and Yu, P. S. (2002) 'Dynamic load balancing on web-server systems', *IEEE Internet Computing,* 3, (3), pp. 28-39.

Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N. and Zdonik, S. (2002) 'Monitoring Streams - A New Class of Data Management Applications'.*28th International Conference on Very Large Databases (VLDB'02).* Hong Kong, China,

Carrara, A., Guzzetti, F., Cardinala, M. and Reichenbach, P. (2000) 'Use of GIS Technology in the Prediction and Monitoring of Landslide Hazard', *Natural Hazards*, (20), pp. 117-135.

Cary, A., Sun, Z., Hristidis, V. and Rishe, N. (2009) 'Experiences on Processing Spatial Data with MapReduce', *Lecture Notes in Computer Science,* 5566, pp. 302-319.

Catlett, C. (2002) 'The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility*', 2nd IEEE International Symposium on Cluster Computing and the Grid.* Berlin, Germany, The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility: pp. 8.

Chapman, D., Joshi, K. P., Yesha, Y., Halem, M., Yesha, Y. and Nguyen, P. (2010) 'Scientific Services on the Cloud', in Furht, B. and Escalante, A.(eds) *Handbook of Cloud Computing.* Springer: New York, pp. 379-406.

Chappell, D. and Liu, L. (2006) *Web Services Brokered Notification 1.3 (WS-BrokeredNotification).* OASIS (wsn-ws_brokered_notification-1.3-spec-os).

Chatterjee, S. and Webber, J. (2004) *Developing Enterprise Web Services: An architects guide.* Prentice Hall PTR.

Chen, A., Di, L., Wei, Y., Bai, Y. and Liu, Y. (2006) 'An Optimised Grid Based, OGC Standards Compliant Collaborative Software System for Serving NASA Geospatial Data'.*30th Annual IEEE/NASA Software Engineering Workshop.*

Chen, N., Di, L., Yu, G. and Gong, J. (2010) 'Geo-processing workflow driven wildfire hot pixel detection under sensor web environment', *Computers & Geosciences,* 36, pp. 362-372.

Chen, Q., Wang, L. and Shang, Z. (2008) 'MRGIS: A MapReduce-Enabled High Performance Workflow System for GIS'.*Fourth IEEE Conference on eScience.* Indiana, Indianapolis, USA,

Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Cetintemel, U., Xing, Y. and Zdonik, S. (2003) 'Scalable Distributed Stream Processing'.*First Biennial Conference on Innovative Data Systems Research.* Pacific Grove, California, USA,

Cheu, R. L., Xie, C. and Lee, D. H. (2002) 'Freeway traffic prediction using neural networks', *Computer-Aided Civil and Infrastructure Engineering,* 17, (1), pp. 53-60.

Chien, S., Tran, D., Davies, A., Johnston, M., Doubleday, J., Castano, R., Scharenbroich, L., Rabideau, G., Cichy, B., Kedar, S., Mandl, D., Frye, S., Song, W., Kyle, P., LaHusen, R. and Cappaelare, P. (2007) 'Lights Out Autonomous Operation of an Earth Observing Sensorweb'.*International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations (RCSGSO).* Moscow, Russia, Chiu, K., Govindaraju, M. and Bramley, R. (2002) 'Investigating the Limits of SOAP Performance for Scientific Computing'*, 11th International Symposium on High Performance Distributed Computing* Edinburgh, UK, Investigating the Limits of SOAP Performance for Scientific Computing: pp. 246.

Chu, K., Brewer, R. and Joseph, S. (2008) *Traffic and navigation support through an automobile head up display.* Manoa, Hawaii: University of Hawaii, USA (ICS-2008-05-02).

Chu, X. and Buyya, R. (2007) 'Service Oriented Sensor Web', in Mahalik, N. P.(ed), *Sensor Network and Configuration: Fundamentals, Standards, Platforms, and Applications.* Springer-Verlag,: Germany, pp. 51-74.

Cignoni, P., Montani, C., Perego, R. and Scopigno, R. (1993) 'Parallel 3D Delauney triangulation'.*Computer Graphics Forum.*Blackwell Publishers.

Clematis, A. and Puppo, E. (1993) 'Effective parallel processing of irregular geometric structures - an experience with the Delaunay triangulation'*, AICA - International Section: Parallel and Distributed Architectures and Algorithms.* Effective parallel processing of irregular geometric structures - an experience with the Delaunay triangulation: pp. 235-251.

Codd, E. F. (1970) 'A Relational Model of Data for Large Shared Data Banks', *Communications of the ACM,* 13, (6), pp. 377-387.

Codd, E. F. (1972) 'Further normalization of the data base relational model', in Rustin, R.(ed), *Data Base Systems.* Prentice-Hall: Englewood Cliffs, NJ, USA.

Cole, R., Goodrich, M. T. and O'Dunlaing, C. (1990) 'Merging free trees in parallel for efficient Voronoi diagram construction', in *Lecture notes in computer science.* Vol. 443 Springer-Verlag: Berlin, pp. 432-445.

Connolly, T. and Begg, C. (2005) *Database Systems: A practical approach to design, implementation, and management.* Pearson Education Ltd.

Cook, W. R. and Barfield, J. (2007) 'Web Service versus Distributed Objects: A Case Study of Performance and Interface Design', *International Journal of Web Services Research,* 4, (3), pp. 49-64.

Cooper, M., Dzambasow, Y., Hesse, P., Joseph, S. and Nicholas, R. (2005) *Internet X.509 Public Key Infrastructure: Certification Path Building*. Available at: http://tools.ietf.org/html/rfc4158.html (Accessed:

Couillard, J. (1993) 'A decision support system for vehicle fleet planning', *Decision Support Systems,* 9, (2), pp. 149-159.

Cox, S. (2007) *Observations and Measurements* Open Geospatial Consortium, Inc (07-022r1, 07-002r3).

Cox, S., Botts, M., Robin, A., Davidson, J. and Falke, S. (2006) *Observations & Measurements.* Open Geospatial Consortium Inc. (OGC® 05-087r4).

Crainic, T. G., Gendreau, M. and Potvin, J.-Y. (2009) 'Intelligent freight-transport systems: Assessment and the contribution of operations research', *Transportation Research Part C: Emerging Technologies,* 17, (6), pp. 541-557.

Cruanes, T., Dageville, B. and Ghosh, B. (2004) 'Parallel SQL execution in Oracle 10g'.*2004 ACM SIGMOD international conference on Management of data.*

Culler, D., Estrin, D. and Srivastava, M. (2004) 'Guest Editors' Introduction: Overview of Sensor Networks', *Computer,* 37, (8), pp. 41-49.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N. and Weerawarana, S. (2002) 'Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI', *Internet Computing, IEEE,* 6, (2), pp. 86-93.

Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S. and Vambenepe, W. (2004) *The WS-Resource Framework*. Available at: http://www-106.ibm.com/developerworks/ library/ws-resource/ws-wsrf.pdf (Accessed:

Danelutto, M., Di Meglio, R., Orlando, S., Pelagatti, S. and Vanneschi, M. (1992) 'A methodology for the development and the support of massively parallel programs', *Future Generation Computer Systems,* 8, (1-3), pp. 205-220.

Dashitenezhad, S., Nadeem, T., Dorohonceanu, B., Borcea, C., Kang, P. and Iftode, L. (2004) 'TrafficView: a driver assistant device for traffic monitoring based on car-to-car communication'*, IEEE 59th Vehicular Technology Conference 2004.* 19th May 2004. TrafficView: a driver assistant device for traffic monitoring based on car-to-car communication: IEEE, pp. 2946-2950.

Dattilo, G. and Spezzano, G. (2003) 'Simulation of a cellular landslide model with CAMELOT on high performance computers', *Parallel Computing,* 29, (10), pp. 1403-1418.

Davis, D., Malhotra, A., Warr, K. and Chou, W. (2009) *Web Services Resource Transfer (WS-RT).* W3C (http://www.w3.org/TR/2009/WD-ws-resource-transfer-20090317).

Davy, J. R. and Dew, P. M. (1989) 'A note on improving the performance of Delaunay triangulation', in Patrikalakis, N. M.(ed), *Scientific Visualization of Physical Phenomena.* Springer-Verlag: Hong Kong, pp. 209-226.

de Groot, W., Goldammer, J. G., Keenan, T., Brady, M., Lynham, T., Csiszar, I. A., Justice, C. O. and O'Loughlin, K. (2006) 'Developing a global early warning system for wildland fire'.*V International Conference on Forest Fire Research.*

Dean, J. and Ghemawat, S. (2008) 'MapReduce: Simplified Data Processing on Large Clusters', *Communications of the ACM,* 51, (1), pp. 107-113.

Dean, J. and Ghemawat, S. Google Inc. (2010) *System and method for efficient large-scale data processing*. 7650331.

DeMers, M. N. (2002) *GIS modelling in raster.* John Wiley & Sons: Chichester, UK.

DeWitt, D. and Gray, J. (1992) 'Parallel database systems: the future of high performance database systems', *Communications of the ACM,* 35, (6), pp. 85-98.

DeWitt, D. and Stonebraker, M. (2008a) 'MapReduce II', *Database Column*, [Online]. Available at: http://databasecolumn.vertica.com/database-innovation/mapreduce-ii/ (Accessed: 15/09/2010).

Dewitt, D. and Stonebraker, M. (2008b) 'MapReduce: A major step backwards', *Database Column*, [Online]. Available at: http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/ (Accessed: 15th September 2010).

Di, L., Chen, A., Yang, W., Liu, Y., Wei, Y., Mehrotra, P., Hu, C. and Williams, D. (2008) 'The development of a geospatial data Grid by integrating OGC Web services with Globus-based Grid technology', *Concurrency & Computation: Practice and Experience,* 20, (14), pp. 1617 - 1635.

Di, L., Chen, A., Yang, W. and Zhao, P. (2003) 'The Integration of Grid Technology with OGC Web Services in NWGISS for NASA EOS Data'.*Global Grid Forum 8 / HPDC12.* Seattle, USA,

Dijkstra, E. W. (1959) 'A Note on Two Problems in Connexion with Graphs', *Numerische Mathematik,* 1, (269-271).

Dikaiakos, M. D., Katsaros, D., Mehra, P., Pallis, G. and Vakali, A. (2009) 'Distributed internet computing for IT and scientific research', *IEEE Internet Computing,* 13, (5), pp. 10-13.

Dillaway, B., Humphrey, M., Smith, C., Theimer, M. and Wasson, G. (2007) *HPC Basic Profile Version 1.0.* Open Grid Forum (GFD-R-P.114).

Ding, Y. and Densham, P. J. (1994) 'A dynamic and recursive parallel algorithm for constructing Delaunay triangulations.*', 6th International Symposium on Spatial Data Handling.* Edinburgh, A dynamic and recursive parallel algorithm for constructing Delaunay triangulations.: pp. 682-696.

Echterhoff, J. and Everding, T. (2008) *OpenGIS Sensor Event Service Interface Specification (proposed).* Open Geospatial Consortium, Inc. (08-133).

Emmi, P. C. and Horton, C. A. (1995) 'A Monte Carlo simulation of error propogation in a GIS based assessment of seismic risk', *International Journal of Geographical Information Science (IJGIS),* 9, (4), pp. 447-461.

Etzion, O. (2005) 'Towards an Event-Driven Architecture: An Infrastructure for Event Processing Position Paper', in *Rules and Rule Markup Languages for the Semantic Web.* Vol. 3791/2005 Springer Berlin / Heidelberg, pp. 1-7.

Everding, T. and Echterhoff, J. (2008) *Event Pattern Markup Language (EML).* Open Geospatial Consortium, Inc. (08-132).

Everding, T. and Echterhoff, J. (2009) *OGC OWS-6 SWE Event Architecture Engineering Report.* Open Geospatial Consortium, Inc. (09-032).

Fairbairn, D., James, P., Hobona, G. and Watson, P. (2008) *SAW-GEO*. Available at: http://edina.ac.uk/projects/seesaw/index.html (Accessed: 23/11/2008).

Fairgrieve, S. (2010) *OWS-7 CCSI-SWE Best Practices Engineering Report.* Open Geospatial Consortium Inc. (10-073r1).

Fang, Y., Lee, B., Chou, T., Lin, Y. and Lien, J. (2009) 'The implementation of SOA within grid structure for disaster monitoring', *Expert Systems with Applications,* 36, pp. 5784-5792.

Farnhill, J. and McAllister, A. (2006) *Grid and Open Geospatial Consortium Collision.* JISC (JISC Circular 02/2006 Full Text).

Farooqui, K., Logrippo, L. and Meer, J. d. (1995) 'The ISO Reference Model for Open Distributed Processing: an introduction', *Computer Networks and ISDN Systems* 27, pp. 1215-1229.

Fielding, R. (2000) *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis. University of California.

Flynn, M. (1966) 'Very High Speed Computing Systems', *Proceedings of IEEE,* 54, pp. 1901-1909.

Follino, G., Forestiero A., Papuzzo, G. and Spezzano, G. (2010) 'A Grid Portal for Solving Geoscience Problems using Distributed Knowledge Discovery Service', *Future Generation Computer Systems,* 26, (1), pp. 87-96.

Forney, G. D. (1973) 'The Viterbi Algorithm', *Proceedings of the IEEE,* 61, (3), pp. 268-278.

Foster, I. (1995) *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Addison Wesley.

Foster, I. (2002) 'What is the Grid? A Three Point Checklist', Grid Today, 1, 6,

Foster, I. (2005) 'Service-Oriented Science', *Science,* 308, (5723), pp. 814-817.

Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Sedukhin, I., Snelling, D., Storey, T., Vambenepe, W. and Weerawarana, S. (2004) *Modeling Stateful Resources with Web Services: Version 1.1.*

Foster, I., Grimshaw, A., Lane, P., Lee, W., Morgan, M., Newhouse, S., Pickles, S., Pulsipher, D., Smith, C. and Theimer, M. (2008) *OGSA Basic Execution Service Version 1.0.* Open Grid Forum

Foster, I. and Kesselman, C. (1998) 'Computational Grids', *CERN European Organization for Nuclear Research* 8, pp. 87-114.

Foster, I. and Kesselman, C. (1999) 'Computational Grids', in Foster, I.(ed), *The GRID: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers Inc.

Foster, I., Kesselman, C., Nick, J. M. and Tuecke, S. (2002) 'The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration'.

Foster, I., Kesselman, C. and Tuecke, S. (2001) 'The Anatomy of the Grid: Enabling Scalable Virtual Organisations', *International Journal of High Performance Computing Applications,* 15, (3), pp. 200-222.

Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J. and Von Reich, J. (2005) *The Open Grid Services Architecture, Version 1.0.* GGF

Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J. and Von Reich, J. (2006) *The Open Grid Services Architecture Version 1.5.* Open Grid Forum

Foster, I., Parastatidis, S., Watson, P. and McKeown, M. (2009) 'How Do I Model State? Let Me Count the Ways', *Queue,* 7, (2), pp. 54-55.

Fox, G. (1989) 'Parallel Computing Comes of Age: Supercomputer Level Parallel Computations at Caltech', *Concurrency & Computation: Practice and Experience,* 1, (1), pp. 63-103.

Fox, G. (2004) 'Software Development Around a Millisecond', Computers in Science and Engineering (CISE Magazine), March/April, p.93-96.

Fox, G., Aktas, M., Aydin, G., H, G., Pallickara, S., Pierce, M. and Sayar, A. (2008) 'Algorithms and the Grid', *Computing and Visualization in Science,* 12, (3), pp. 115-124.

Fox, G., Williams, D. and Messina, P. (1994) *Parallel Computing Works.* Morgan Kaufmann Publishers Inc.

Fraternali, P., Rossi, G. and Sanchez-Figueroa. (2010) 'Rich Internet Applications', IEEE Internet Computing, 14, 3, p.9-12.

Freeman, P. K., Keen, M. and Mani, M. (2003) *Dealing with Increased Risk of Natural Disasters: Challenges and Options.* International Monetary Fund (WP/03/197).

Friis-Christensen, A., Lutz, M., Ostlander, N. and Bernard, L. (2007) 'Designing Service Architectures for Distributed Geoprocessing: Challenges and Future Directions', *Transactions in GIS,* 11, (6), pp. 799-816.

Frizziero, E., Gulmini, M., Lelli, F., Maron, G., Oh, A., Orlando, S., Petrucci, A., Squizzato, S. and Traldi, S. (2006) 'Instrument Element: A New Grid component that enables the control of remote instrumentation*', International Conference on Cluster Computing and Grid.* Singapore, Instrument Element: A New Grid component that enables the control of remote instrumentation

Gagliardi, F., Jones, B., Grey, F., Begin, M. and Heikkurinen, M. (2005) 'Building an infrastructure for scientific Grid computing: status and goals of the EGEE project', *Philosophical transactions of the Royal Society,* 363, (1833), pp. 1729-1742.

Gaynor, M., Moulton, S. L., Welsh, M., LaCombe, E., Rowan, A. and Wynne, J. (2004) 'Integrating Wireless Sensor Networks with the GRID', IEEE Internet Computing, 8, 4, July-August 2004, p.32-39.

Ghiani, G., Guerrriero, F., Laporte, G. and Musmanno, R. (2003) 'Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies', *European Journal of Operational Research,* 151, pp. 1-11.

Ghimire, D. R., Simonis, I. and Wytzisk, A. (2005) 'Integration of GRID Approaches into the Geographic Web Service Domain*', FIG Working Week and GSDI-8.* Cairo, Egypt, Integration of GRID Approaches into the Geographic Web Service Domain

Gittings, B. M., Sloan, T. M., Healey, R. G., Dowers, S. and Waugh, T. C. (1994) 'Meeting expectations: a review of GIS performance issues', in Mether, P. M.(ed), *Geographical Information Handling - Research and Applications.* Wiley, pp. 33-45.

Glatard, T. (2008) 'A Service Oriented Architecture enabling dynamic service grouping for optimizing distributed workflow execution', *Future Generation Computer Systems (In Press)*.

Glatard, T., Montagnat, J. and Pennec, X. (2006) 'Efficient services composition for grid-enabled data-intensive applications'.*International Symposium on High Performance Distributed computing (HPDC'06).* Paris, France IEEE.

Glimsdal, S., Pedersen, G. K. and Langtangen, H. P. (2004) 'An investigation of overlapping domain decomposition methods for one-dimensional dispersive long wave equations', *Advances in Water Resources,* 27, pp. 1111-1133.

Goldammer, J. G. (2006) 'Global Early Warning System for Wildland Fire*', 3rd International Conference on Early Warning.* Bonn, Germany, Global Early Warning System for Wildland Fire

Goldman, O. and Lenkov, D. (2005) *XML Binary Characterization.* W3C

Gong, J. and Xie, J. (2009) 'Extraction of drainage networks from large terrain datasets using high throughput computing', *Computers & Geosciences,* 35, pp. 337-346.

González Cortéz, F. and Leduc, T. (2010) 'GGL: A geo-processing definition language that enhance spatial SQL with paramaterization'.*13th AGILE International Conference on Geographic Information Science.* Guimarães, Portugal,

Gottschalk, K., Graham, S., Kreger, H. and Snell, J. (2002) 'Introduction to Web Services Architecture', *IBM Systems Journal,* 41, (2).

Govindan, R., Hong, W., Madden, S., Franklin, M. J. and Shenker, S. (2002) *The Sensor Network as a Database.* University of Southern California (TR02-02-771).

Graham, S., Hull, D. and Murray, B. (2006) *Web Services Base Notification 1.3 (WS-BaseNotification).* Organisation for Advancement of Structured Information Standards (OASIS) (wsn-ws_base_notification-1.3-spec-os).

Grasso, V. F. and Singh, A. (2008) 'Global Environment Alert Service', *Advances in Space Research,* 41, (11), pp. 1836-1852.

Gray, J. (1981) 'The Transaction Conept: Virtues and Limitations'.*7th International Conference on Very Large Databases (VLDB).* Cannes, France,

Gray, J., Liu, D., Nieto-Santisteban, M., Szalay, A. and Heber, G. (2005) 'Scientific data management in the coming decade', *ACM SIGMOD Record,* 34, (4), pp. 34-41.

Gray, J. and Patterson, D. (2003) 'A conversation with Jim Gray', *ACM Queue,* 1, (4), pp. 8-17.

Greenfeld, J. S. (2002) 'Matching GPS observations to locations on a digital map'.*81st Annual Meeting of the Transportation Research Board.* Washington, D. C.,

Greenwald, R., Stackowiak, R. and Stern, J. (2008) *Oracle Essentials: Oracle Database 11g.* O'Reilly.

Grenon, P. and Smith, B. (2004) 'SNAP and SPAN: Towards Dynamic Spatial Ontology', *Spatial Cognition and Computation,* 5, (1), pp. 69-104.

Grimshaw, A. (2003) 'Grid Services extend Web Services', *SOA Web Services Journal,* 506.

Gropp, W., Lusk, E. and Skjellum, A. (1999) *Using MPI: Portable Parallel Programming with the Message-passing Interface.* MIT Press.

Grossman, R. L., Gu, Y., Mambretti, J., Sabala, M., Szalay, A. and White, K. (2010) 'An overview of the Open Science Data Cloud*', 19th ACM International Symposium on High Performance Distributed Computing.* Chicago, Illinois, An overview of the Open Science Data Cloud: ACM, pp. 377-384.

Gudgin, M., Mendelsohn, N., Nottingham, M. and Ruellan, H. (2005a) *SOAP Message Transmission Optimization Mechanism.* W3C

Gudgin, M., Mendelsohn, N., Nottingham, M. and Ruellan, H. (2005b) *XML-binary Optimized Packaging.* W3C

Gupta, R., Pande, S., Kleanthis, P. and Sarkar, V. (1999) 'Compilation techniques for parallel systems', *Parallel Computing,* 25, pp. 1741-1783.

Han, L., Potter, S., Beckett, G., Pringle, G., Sung-Han, K., Upadhyay, R., Wickler, G., Berry, D., Welch, S., Usmani, A., Torero, J. and Tate, A. (2010) 'Firegrid: An e-infrastructure for the next-generation emergency response support', *Journal of Parallel and Distributed Computing,* 70, (11), pp. 1128-1141.

Hansen, P. B. (1993) 'Model Programs for Computational Science: A Programming Methodology for Multicomputers', *Concurrency & Computation: Practice and Experience,* 5, (5), pp. 407-423.

Hart, P. E., Nilsson, N. J. and Raphael, B. (1972) 'Correction to "A Formal Basis for the Heuristic Determination of Minimum cost Paths"', *SIGART Newsletter,* 37, pp. 28-29.

Hart, Q. and Gertz, M. (2005) 'Querying streaming geospatial image data: The GeoStreams project'.*17th International Conference on Scientific and Statistical Database Management.* University of California, Santa Barbara, Havlik, D., Schimak, G. and Barta, R. (2008) 'Advanced Cascading Sensor Observation Service'.*International Congress on Environmental Modelling and Software.* Barcelona, Spain,

Havlik, D., Schimak, G. and Bleier, T. (2009) 'Cascading and replicating the OGC Sensor Observation Service'.*18th World IMACS/MODSIM Congress.* Cairns, Australia,

Hawick, K. A., Coddington, P. D. and James, H. A. (2003) 'Distributed frameworks and parallel algorithms for processing large-scale geographic data', *Parallel Computing,* 29, pp. 1297-1333.

Healey, R. and Desa, G. B. (1990) 'Transputer-Based Parallel Processing for GIS Analysis: Problems and Potentialities'.*Auto-Carto 9.* Baltimore, Maryland, USA,

Healey, R., Dowers, S., Gittings, B. M. and Mineter, M. J. (1998) *Parallel Processing Algorithms for GIS.* Taylor & Francis.

Herring, J. R. (2006) *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option.* Open Geospatial Consortium Inc. (06-104r3).

Hickman, B. L., Bishop, M. P. and Rescigno, M. V. (1995) 'Advanced Computational Methods for Spatial Information Extraction', *Computers & Geosciences,* 21, (1), pp. 153-173.

Hickson, I. and Hyatt, D. (2008) *HTML 5.* W3C

Higgins, C. (2008) *SEE-GEO*. Available at: http://edina.ac.uk/projects/ seesaw/index.html (Accessed: 18/11/2008).

Higgins, C., Lee, C. A. and Sekiguchi, S. (2008) 'OGC-OGF Collaboration Workshop Final Report*', OGF-22.* Cambridge, MA, USA, OGC-OGF Collaboration Workshop Final Report.

Hingne, V., Joshi, A., Houstis, E. and Michopoulos, J. (2003) 'On the grid and sensor networks*', Grid Computing, 2003. Proceedings. Fourth International Workshop on.* On the grid and sensor networks: pp. 166-173.

Hluchy, L., Habala, O., Tran, V., Gatial, E., Maliska, M., Simo, B. and Slizik, P. (2005) 'Collaborative Environment For Grid-Based Flood Prediction', *Computing & Informatics,* 24, pp. 1001-1022.

Hoare, C. A. R. (1978) 'Communicating sequential processes', *ACM,* 21 (8), pp. 666-677.

Hobona, G., Fairbairn, D. and James, P. (2007) 'Workflow Enactment of Grid-Enabled Geospatial Web Services'.*UK e-Science All Hands Meeting.* Nottingham,

Hoef, J., Peterson, E. and Theobald, D. (2006) 'Spatial Statistical Models that Use Flow and Stream Distance', *Environmental and Ecologial Statistics,* 16, pp. 449-464.

Hogue, C. and Graves, D. (1994) *Power Fortran Accelerator User's Guide.* Silicon Graphics Inc.

Hong-chun, Z., Hai-ying, L., Tao, J. and Ji-zian, Z. (2009) 'Research on Remote sensing data processing strategy and application based on grid operation', *Procedia Earth and Planetary Science,* 1, pp. 1180-1185.

Horiuchi, S., Negishi, K., Abe, A., Kamimura, A. and Fujinawa, Y. (2005) 'An automatic processing system for broadcasting earthquake alarms', *Bulletin of the Seismological Society of America,* 95, pp. 708-718.

Huber, W., Lädke, M. and Ogger, R. (1997) 'Extended floating car data for the acquisition of traffic information'.*4th World Congress on Intelligent Transport Systems.* Berlin, Germany,

Hughes, D., Greenwood, P., Coulson, G., Blair, G., Pappenberger, F., Smith, P. and Beven, K. (2006) 'An Intelligent and Adaptable Flood Monitoring and Warning System', *UK E-Science All Hands Meeting 2006*.

Hummel, B. (2006) 'Map Matching for Vehicle Guidance', in Drummond, J.(ed), *Dynamic and Mobile GIS: Investigating Changes in Space and Time.*

Humphrey, M., Wasson, G., Kiryakov, Y., Park, S.-M., Del Vecchio, D., Beekwilder, N. and Gray, J. (2005) 'Alternative Software Stacks for OGSA-based Grids'.*ACM/IEEE Supercomputing.* Seattle, WA, USA:IEEE.

Hwang, K. and Xu, Z. (1996) 'Scalable Parallel Computers for Real-Time Signal Processing', *IEEE Signal Processing Magazine,* 30, (4), pp. 50-56.

Hwu, W., Ryoo, S., Sain-Zee, U., Kelm, J. H., Gelado, I., Stone, S. S., Kidd, R. E., Baghsorkhi, S., S., Aqeel, A. M., Tsao, S. C., Navarro, N., Lumetta, S. S., Frank, M. I. and Patel, S. J. (2007) 'Implicitly Parallel Programming

Models for Thousand-Core Microprocessors'.*44th Annual Design Automation Conference.* San Diego, California, USA:ACM.

Jagadeesh, G. R., Srikanthan, T. and Zhan, X. D. (2004) 'A Map Matching Method for GPS Based Real-Time Vehicle Location', *The Journal of Navigation,* 57, pp. 429-440.

Jang, J. S. R. and Sun, C. T. (1996) *Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence.* Prentice-Hall Inc.: Upper Saddle River, NJ, USA.

Kalogeraki, V., Gunopulos, D., Sandhu, R. and Thuraisingham, B. (2008) 'QoS Aware Dependable Distributed Stream Processing'.*11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC).* Orlando, Florida, USA:IEEE.

Kanamori, H., Hauksson, E. and Heaton, T. (1997) 'Real-time seismology and earthquake hazard mitigation', *Nature: International Weekly Journal of Science*, (390), pp. 461-464.

Karonis, N., Toonen, B. and Foster, I. (2003) 'MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface', *Journal of Parallel and Distributed Computing* 63, (5), pp. 551-563.

Katz, M. J., Overmars, M. H. and Sharir, M. (1991) 'Efficient hidden surface removal for objects with small union size*', 7th International Symposium on Computational Geometry.* Efficient hidden surface removal for objects with small union size: pp. 31-40.

Kaye, D. (2003) *Loosely Coupled: The Missing Pieces of Web Services.* RDS Strategies LLC: USA.

Keller, R., Krammer, B., Mueller, M., Resch, M. and Gabriel, E. (2003) 'MPI Development Tools and Applications for the Grid*', Workshop on Grid Application and Programming Tools: GGF8 Meetings.* Seattle, WA, USA, MPI Development Tools and Applications for the Grid.

Kerry, K. E. and Hawick, K. A. (1998) 'Kriging Interpolation on High-Performance Computers', *Lecture Notes in Computer Science,* 1401, pp. 429-438.

Kidner, D. B., Rallings, P. J. and Ware, A., J. (1997) 'Parallel processing for Terrain Analysis in GIS: Visibility as a Case Study', *Geoinformatica,* 1, (2), pp. 183-207.

Kiehle, C., Greve, K. and Heier, C. (2006) 'Standarized Geoprocessing - Taking Spatial Data Infrastructures one Step Further'*, 9th AGILE Conference on Geographic Information Science.* Visegrad, Hungary, Standarized Geoprocessing - Taking Spatial Data Infrastructures one Step Further.

Kim, S., Shekhar, S. and Min, M. (2008) 'Contraflow Transportation Network Reconfiguration for Evacuation Route Planning', *IEEE Transactions on Knowledge and Data Engineering,* 20, (8).

Klopfer, M. and Kanellopoulos, I. (eds.) (2008) *Orchestra: An open service architecture for risk management.* The Orchestra Consortium.

Kobialka, T., Buyya, R., Leckie, C. and Kotagiri, R. (2007) 'A Sensor Web Middleware with Stateful Services for Heterogeneous Sensor Networks'.*3rd International Conference on Intelligent Sensors, Sensor Networks and Information. ISSNIP 2007.*

Kopetz, H. (1999) *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers.

Kothuri, R., Godfrind, A. and Beinat, E. (2007) *Pro Oracle Spatial for Oracle Database 11g.* Apress.

Koutroumpas, M. and Higgins, C. (2008) 'A Pipeline Processing Approach to GIS'.*11th AGILE International Conference on Geographic Information Science.* University of Girona, Spain,

Kraak, M. J., Sliwinski, A. and Wytzisk, A. (2005) 'What happens at 52N? An Open source approach to education and research'.*Joint Commissions Seminar "Internet-Based Cartographic Teaching and Learning: Atlases, Map Use and Visual Analytics.* Madrid,

Krishnamurthy, S., Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J., Wei, H., Madden, S., Raman, V., Reiss, F. and Shah, M., A. (2003) 'TelegraphCQ: An Architectural Status Report', *IEEE Data Engineering Bulletin,* 26, (1).

Krüger, A. and Kolbe, T. (2008) 'Mapping Spatial Data Infrastructures to a Grid Environment for Optimised Processing of Large Amounts of Spatial Data'*, XXXVII ISPRS Congress: Special Session: Spatial Data Infrastructure (SDI) and Spatial Information Grid (SIG).* Beijing, China, Mapping Spatial Data Infrastructures to a Grid Environment for Optimised Processing of Large Amounts of Spatial Data: pp. 1559.

Kruvoruchko, K. and Gribov, A. (2004) 'Geostatistical Interpolation and Simulation with Non-Euclidean Distances'.*GeoENV IV: International Conference on Geostatistics for Environmental Applications.* Neuchatel, Switzerland:Kluwer Academic Publishers.

Kurzbach, S., Pasche, E., Lanig, S. and Zipf, A. (2009) 'Benefits of Grid Computing for Flood Modeling In Service-Oriented Spatial Data Infrastructures', *GIS Science,* 3, pp. 89-97.

Kussul, N., Shelestov, A. and Skakun, S. (2009) 'Grid and sensor web technologies for environmental monitoring', *Earth Science Informatics,* 2, pp. 37-51.

Kussul, N., Shelestov, A., Skakun, S. and Kravchenko, O. (2008) 'Data Assimilation Technique for Flood Monitoring & Prediction', *International Journal 'Information Theories & Applications',* 15, pp. 76-83.

Lahrmann, H. (2007) 'Floating Car Data for Traffic Monitoring'.*i2TERN* Aalborg, Denmark,

Lämmel, L., Rieser, M. and Nagel, K. (2010) 'Large Scale Microscopic Evacuation Simulation ', *Pedestrian & Evacuation Dynamics 2008,* 2, pp. 547-553.

Langran, G. (1992) *Time in Geographic Information Systems.* Taylor & Francis: London, UK.

Lanig, S. and Zipf, A. (2009a) 'Interoperable processing of digital elevation models in grid infrastructures', *Earth Science Informatics,* 2, pp. 107-116.

Lanig, S. and Zipf, A. (2009b) 'Towards Generalization Processes of LiDAR Data based on Grid and OGC Web Processing Services'.*Geoinformatik.* Osnabruck, Germany,

Lanthier, M., Nussbaum, D. and Jorg-Rudiger, S. (2003) 'Parallel implementation of geometric shortest path algorithms', *Parallel Computing,* 29, pp. 1443-1479.

Leavitt, N. (2010) 'Will NoSQL Databases Live Up to Their Promise?', Computer, 43, 2, p.12-14.

Lee, C. and Hamdi, M. (1995) 'Parallel image processing of applications on a network of workstations', *Parallel Computing,* 21, pp. 137-160.

Lee, C. and Percivall, G. (2008) 'Standards-Based Computing Capabilities for Distributed Geospatial Applications', *Computer,* 41, (11), pp. 50-57.

Lee, W.-H., Tseng, S.-S. and Tsai, S.-H. (2009) 'A knowledge based real-time travel time prediction system for urban network', *Expert Systems with Applications,* 36, pp. 4329-4247.

Lehning, M., Dawes, N., Bavay, M., Parlange, M., Nath, S. and Zhao, F. (2009) 'Instrumenting the Earth: Next-Generation Sensor Networks and Environmental Science', in Hey, T., Tansley, S. and Tolle, K.(eds) *The Fourth Paradigm: Data-Intensive Scientific Discovery.* Microsoft Research.

Li, B., Zhao, H. and Lv, Z. (2010) 'Parallel ISODATA Clustering of Remote Sensing Images on MapReduce*', International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery.* Huangshan, China, Parallel ISODATA Clustering of Remote Sensing Images on MapReduce: IEEE, pp. 380-383.

Li, X., Plale, B., Vijayakumar, N., Ramachandran, R., Graves, S. and Conover, H. (2008) 'Real-time Storm Detection and Weather Forecast Activation through Data Mining and Events Processing', *Earth Science Informatics,* 1, (2), pp. 49-57.

Liang, S. H. L., Chen, S., Huang, C. Y., Li, R. Y., Chang, D. Y. C., Badger, J. and Rezel, R. (2010) 'GeoCENS: Geospatial Cyberinfrastructure for Environmental Sensing*', GIScience 2010.* Zurich, Switzerland, GeoCENS: Geospatial Cyberinfrastructure for Environmental Sensing.

Liang, S. H. L., Croitoru, A. and Tao, C. V. (2005) 'A Distributed Geospatial Infrastructure for Sensor Web', *Computers & Geosciences,* 31, (2), pp. 221-231.

Liang, S. H. L., Tao, C. V. and Croitoru, A. (2003) 'The design and prototype of a distributed geospatial infrastructure for smart sensor webs'.*6th AGILE Conference on Geographic Information Science.* Lyon, France, 24th-26th April 2003.Presses Polytechniques et Universitaires Romandes.

Liu, C. and Meng, X. (2008) 'Determination of Routing Velocity with GPS Floating Car Data and WebGIS-Based Instantaneous Traffic Information Dissemination', *The Journal of Navigation,* 61, pp. 337-353.

Luckham, D. (2006) *What's the Difference Between ESP and CEP?* Available at: http://complexevents.com/?p=103 (Accessed: 25/01/2010).

Luckham, D. and Schulte, R. (2008) *Event Processing Glossary - Version 1.1*. Available at: http://www.ep-ts.com/component/option,com_docman/task,doc_download/gid,66/Itemid,84/ (Accessed: 25/01/2010).

Lv, Z., Hu, Y., Zhong, H., Wu, J., Li, B. and Zhao, H. (2010) 'Parallel K-Means Clustering of Remote Sensing Images Based on Map-Reduce', *Lecture Notes in Computer Science,* 6318, pp. 162-170.

Lynch, N. and Gilbert, S. (2002) 'Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services', *ACM SIGACT News,* 33, (2), pp. 51-59.

MacDougall, E. B. (1984) 'Surface mapping with weighted averages in a microcomputer', in *Spatial Algorithms for Processing Land Data with a Microcomputer: Lincoln Institute Monograph #84-2.* Lincoln Institute of Land Policy: Cambridge, MA, USA.

MacKenzie, M., McCabe, F., Brown, P., Metz, R. and Hamilton, B. A. (2006) *Reference Model for Service Oriented Architecture.* OASIS (wd-soa-rm-cd1).

MacLaren, J., Sakellariou, R., Krishnakumar, K. T., Garibaldi, J. and Ouelhadj, D. (2004) 'Towards Service Level Agreement Based Scheduling on the Grid'.*Workshop on Planning and Scheduling for Web and Grid Services (in conjunction with ICAPS-04).* Whistler, BC, Canada,

Madden, S. (2002) *Query Processing for Streaming Sensor Data.* PhD Qualifying Exam Proposal thesis. Computer Science Division, University of Berkeley, California, USA.

Magillo, P. and Puppo, E. (1998) 'Algorithms for Parallel Terrain Modelling and Visualisation', in Healey, R., Dowers, S., Gittings, B. M. and Mineter, M. J.(eds) *Parallel Processing Algorithms for GIS.* Taylor & Francis: London, UK.

Malarvizhi, N. and Uthariaraj, V. R. (2009) 'A New Mechanism for Job Scheduling in Computational Grid Network Environments', *Lecture Notes in Computer Science,* 5820, pp. 490-500.

Marchel, F., Hackney, J. and Axhausen, K. W. (2005) 'Efficient Map Matching of Large Global Positioning System Data Sets: Tests on Speed-Monitoring

Experiment in Zurich', *Transportation Research Record,* 1935, pp. 93-100.

Marco, J. and Marco, R. (2003) 'First Prototype of the CrossGrid Testbed'.*1st European AcrossGrids Conference.* Santiago de Compostella, Spain,

Martinez, K., Hart, J. K. and Ong, R. (2004) 'Environmental sensor networks', *Computer,* 37, (8), pp. 50-56.

Marzolla, M., Andreetto, P., Venturi, V., Ferraro, A., Memon, S., Twedell, B., Riedel, M., Mallmann, D., Streit, A., van de Berghe, S., Li, V., Snelling, D., Stamou, K., Shah, Z. A. and Hedman, F. (2007) 'Open Standards-Based Interoperability of Job Submission and Management Interfaces across the Grid Middleware Platforms gLite and UNICORE*', e-Science and Grid Computing, IEEE International Conference on.* Open Standards-Based Interoperability of Job Submission and Management Interfaces across the Grid Middleware Platforms gLite and UNICORE: pp. 592-601.

Maso, J., Pomakis, K. and Julia, N. (2010) *OpenGIS Web Map Tile Service Implementation Standard.* Open Geospatial Consortium Inc. (OGC 07-057r7).

Mather, P. M. (2004) *Computer Processing of Remotely-Sensed Images.* John Wiley and Sons Ltd.: Chichester, UK.

Matheus, A. and Higgins, C. (2009) 'A Shibboleth Service Provider for OGC Web Map Services'.*16th ACM Conference on Computer and Communications Security (CCS '09).* Chicago, IL, USA:ACM.

Mazzetti, P. (2010) 'Grid Enablement of OpenGeospatial Web Services: the G-OWS Working Group'.*Geophysical Research Abstracts.* Vienna, Austria:European Geosciences Union.

McBryan, O. A. (1994) 'An overview of message passing environments', *Parallel Computing,* 20, (4), pp. 415-678.

McGough, A. and Colling, D. (2006) 'The GRIDCC Project: the GRIDCC Collaboration'.*First International Conference on Communication System Software and Middleware.*IEEE.

Mennis, J., Viger, R. and Tomlin, C. D. (2005) 'Cubic Map Algebra Functions for Spatio-Temporal Analysis', *Cartography and Geographic Information Science,* 32, (1), pp. 17-32.

Merrill, D. (2008a) *Secure Addressing Profile 1.0.* Open Grid Forum (GFD-R-P.131).

Merrill, D. (2008b) *Secure Communication Profile 1.0.* Open Grid Forum (GFD-R-P.132).

Metcalfe, S. E., Whyatt, J. D. and Derwent, R. G. (1995) 'A comparison of model and observed network estimates of sulphur decomposition across Great Britain for 1990 and its likely source attribution', *Quarterly Journal of the Royal Meteorological Society,* 121, pp. 1387-1411.

Metropolis, N. and Ulam, S. (1949) 'The Monte Carlo Method', *Journal of the American Statistical Association,* 44, (247), pp. 335-341.

Mineter, M. J. and Dowers, S. (1999) 'Parallel processing for geographical applications: A layered approach', *Journal of Geographical Systems,* 1, pp. 61-74.

Mitas, L. and Mitasova, H. (1999) 'Spatial Interpolation', in Longley, P., Good-child, M., Maguire, D. and Rhind, D.(eds) *Geographical Informations Systems: Principles, Techniques, Management and Applications.* Vol. 1 Wiley: London, pp. 481-492.

Miwa, T., Sakai, T. and Morikawa, T. (2008) 'Route Identification and Travel Time Prediction Using Probe-Car Data', *International Journal of ITS Research,* 2, (1), pp. 21-28.

Mourelatos, A. P. (1978) 'Events, processes, and states', *Linguistics and Philosophy,* 2, (3).

Mower, J. E. (1996) 'Developing parallel procedures for line simplification', *International Journal of Geographical Information Science (IJGIS),* 10, (6), pp. 699-712.

Muehlen, M., Nickerson, J. and Swenson, K. (2005) 'Developing web services choreography standards - the case of REST vs SOAP', *Decision Support Systems,* 40, pp. 9 -29.

Müller, M., Bernard, L. and Brauner, J. (2010) 'Moving Code in Spatial Data Infrastructures - Web Service Based Deployment of Geoprocessing Algorithms', *Transactions in GIS,* 14, (S1), pp. 101-118.

Murty, J. (2008) *Programming Amazon Web Services: S3, EC2, SQS, FPS and SimpleDB.* O'Reilly: Sebastopol, CA, USA.

Muzik, I. and Chang, C. (1993) 'Flood Simulation assisted by a GIS'*, HydroGIS: Application of Geographic Information Systems in Hydrology and Water Resources.* Vienna, Austria, Flood Simulation assisted by a GIS: IAHS Press, pp. 531 - 540.

Na, A. (2007) SensorML and TransducerML, personal communication list, S. m.

Na, A., Priest, M., Cox, S., Botts, M., Robin, A., Walkowski, A., Simonis, I., Echterhoff, J., Liang, S., Davidson, J. and Niedzwiadek, H. (2007) *Sensor Observation Service.* Open Geospatial Consortium Inc. (OGC 06-009r6).

Nagaratnam, N., Janson, P., Dayka, J., Nadalin, A., Siebenlist, F., Welch, V., Foster, I. and Tuecke, S. (2002) *The security architecture for open grid services.* The Globus Project

Nagel, K. and Rickert, M. (2001) 'Parallel implementation of the TRANSIMS micro-simulation', *Parallel Computing,* 27, (12), pp. 1611-1639.

Nakamura, H., Horiuchi, S., Wu, C., Yamamoto, S. and Rydelek, P. (2009) 'Evaluation of the real-time earthquake information system in Japan', *Geophysical Research Letters,* 36, pp. L00B01.

Nekovee, M. (2005) 'The promise and challenges of vehicular ad hoc networks*', Workshop of Ubiquitous Computing and e-Research.* Edinburgh, UK, The promise and challenges of vehicular ad hoc networks

Nelson, M. R. (2009) 'Building an Open Cloud', *Science Magazine,* 324, (5935), pp. 1656-1657.

Niblett, P. and Graham, S. (2005) 'Events and service-oriented architecture: The OASIS Web Services Notification specification', *IBM Systems Journal,* 44, (4).

Nicolescu, C. and Jonker, P. (2002) 'A data and task parallel image processing environment', *Parallel Computing,* 28, pp. 945-965.

Noh, S. H. and Kim, T. J. (1998) 'A comprehensive analysis of map matching algorithms for ITS', *Hongik Journal of Science and Technology* 9, pp. 303-313.

Ochieng, W. Y., Quddus, M. A. and Noland, R. B. (2004) 'Map Matching in Complex Urban Road Networks', *Revista Brasileira de Cartografia (Brazilian Journal of Cartography),* 55, (2), pp. 1-18.

Oh, S., Bulut, H., Uyar, A., Wu, W. and Fox, G. (2005) 'Optimised Communication using the SOAP Infoset For Mobile Multimedia Collaboration Applications'.*International Symposium on Collaborative Technologies and Systems.* Saint Louis, Missouri, USA,

Oh, S. and Fox, G. (2005) '*HHFR: A new architecture for Mobile Web Services Principles and Implementations', in* Comm. Grids Technical Paper.

Openshaw, S. (2000) 'Geocomputation', in Openshaw, S. and Abrahart, R. J.(eds) *Geocomputation.* Taylor & Francis: New York.

Padberg, A. and Greve, K. (2009) 'Gridification of OGC Web Services: Challenges and Potential', GIS.Science: Die Zeitschrift fur Geoinformatik, 3, p.77-81.

Padberg, A. and Kiehle, C. (2009) 'Towards a grid-enabled SDI: Matching the paradigms of OGC Web Services and Grid Computing', *article under review for the International Journal of Spatial Data Infrastructures Research, Special Issue GSDI-11*.

Pallickara, S. and Fox, G. (2003) 'A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids*', Middleware 2003.* Rio de Janeiro, Brazil, A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids: pp. 41-60.

Panagiotis, P. and Vretanos, A. (2010) *OpenGIS Web Feature Service 2.0 Interface Standard.* Open Geospatial Consortium Inc. (OGC 09-025r1).

Peacock, C., Goode, A. and Brett, A. (2004) 'Automatic forensic face recognition from digital images', *Science & Justice,* 44, (1), pp. 145-155.

Percivall, G. (2002) *ISO19119 and OGC Service Architecture.* NASA / Global Science and Technology, Inc.

Percivall, G., Reed, C., Leinenweber, L., Tucker, C. and Cary, T. (2008) *OGC Reference Model.* Open Geospatial Consortium Inc. (08-062r4).

Phillips, P. J., Wechsler, H., Huang, J. and Rauss, P. J. (1998) 'The FERET database and evaluation procedure for face-recognition algorithms', *Image and Vision Computing,* 16, pp. 295-306.

Pidd, M., de Silva, F. N. and Eglese, R. W. (1996) 'A simulation model for emergency evacuation', *European Journal of Operational Research,* 90, pp. 413-419.

Planas, E., Pastor, E., Presutto, F. and Tixier, J. (2008) 'Results of the MITRA project: Monitoring and intervention for the transportation of dangerous goods', *Journal of Hazardous Materials,* 2008, (152), pp. 516-526.

Plaza, A., Benediktsson, J. A., Boardman, J. W., J., B., Bruzzone, L., Camps-Valls, G., Chanussot, J., Fauvel, M., Gamba, P., Gualtieri, A., Marconcini, M., Tilton, J. C. and Trianni, G. (2009) 'Recent advances in techniques for hyperspectral image processing', *Remote Sensing of Environment,* 113, pp. 5110-5122.

Portele, C. (2007) *OpenGIS Geography Markup Language (GML) Encoding Standard version 3.2.1.* Open Geospatial Consortium Inc. (07-036).

Prescod, P. (2002) *Second Generation Web Services*. Available at: http://www.xml.com/lpt/a/915 (Accessed: 27/4/2008).

Pritchett, D. (2008) 'BASE: An ACID Alternative', *Queue,* 6, (3).

Quinn, M. (1994) *Parallel Computing: Theory and Practice.* New York, NY, USA.

Racine, P. (2010) 'Introducing PostGIS WKT Raster: Seamless Raster/Vector Operations in a Spatial Database*', FOSS4G.* Barcelona, Spain, Introducing PostGIS WKT Raster: Seamless Raster/Vector Operations in a Spatial Database.

Reichardt, M. (2005) *Sensor Web Enablement: An OGC White Paper.* Open Geospatial Consortium (OGC 07-165).

Rejaie, A. and Shinozuka, M. (2004) 'Reconnaissance of Golcuk 1999 earthquake damage using satellites', *Journal of Aerospace Engineering,* 17, (1), pp. 20-25.

Reuter, A. and Haerder, T. (1983) 'Principles of Transaction-Oriented Database Recovery', *Computing Surveys,* 15 (4), pp. 287-317.

Richards, A. (2006) *The Codeplay Sieve C++ Parallel Programming System.*

Rueda-Velasquez, C. A. (2007) *Geospatial Image Stream Processing: Models, Techniques and Applications in Remote Sensing Change Detection.* PhD thesis. University of California, Davis.

Rueda, C., Gertz, M., Ludascher, B. and Hamann, B. (2006) 'An extensible infrastructure for processing distributed geospatial data streams*', 18th International Confererence on Scientific and Statistical Database*

Management (SSDBM). An extensible infrastructure for processing distributed geospatial data streams: pp. 285-290.

Samet, H. (1984) 'The Quadtree and related hierarchical data structures', *ACM Computing Surveys (CSUR),* 16, (2), pp. 187-260.

Samofalov, V. V. and Konovalov, A. V. (1996) 'Technology of debugging programs for computers with mass parallelism', *Mathematical modeling of physical processes*, (4), pp. 52-56.

Sandoz, P., Pericas-Geertsen, S., Kawaguchi, K., Hadley, M. and Pelegri-Llopart, E. (2003) *Fast Web Services*. Available at: http://java.sun.com/developer/technicalArticles/WebServices/fastWS (Accessed: 3/8/2009).

Sano, K., Kobayashi, Y. and Nakamura, T. (2004) 'Differential coding scheme for efficient parallel image composition on a PC cluster system', *Parallel Computing,* 30, pp. 285-299.

Sawyer, M. (1998) 'The Software Environment and Standardisation Initiatives', in Healey, R., Dowers, S., Gittings, B. M. and Mineter, M. J.(eds) *Parallel Processing algorithms for GIS.* Taylor & Francis: London.

Schad, J. (2010) 'Flying Yellow Elephant: Predictable and Efficient MapReduce in the Cloud'.*VLDB 2010: 36th International Conference on Very Large Databases, PhD workshop.* Singapore,

Schaeffer, B. (2008) 'Towards a Transactional Web Processing Service (WPS-T)'.*GI Days.* Munster, Germany,

Schäfer, R. P., Thiessenhusen, K. U. and Wagner, P. (2002) 'A Traffic Information System by Means of Real-time Floating-car Data'.*ITS World Congress.* Chicago, USA,

Scharl, A. and Tochtermann, K. (2007) *The GeoSpatial Web: How Geobrowsers, social software and the web 2.0 are shaping the network society.* Springer: London, UK.

Schrijver, A. (2005) 'On the history of combinatorial optimisation (till 1960)', in Aardal, K., Nemhauser, G. L. and Weismantel, R.(eds) *Discrete Optimization.* Vol. 12.

Schut, P. (2007) *OpenGIS Web Processing Service version 1.0.0.* Open Geospatial Consortium (OGC 05-007r7).

Schwiegelshohn, U., Badia, R. M., Bubak, M., Danelutto, M., Dustdar, S., Gagliardi, F., Geiger, A., Hluchy, L., Kranzmuller, D., Laure, E., Priol, T., Reinefeld, A., Resch, M., Reuter, A., Rienhoff, O., Ruter, T., Sloot, P., Talia, D., Ullmann, K., Yahyapour, R. and Voigt, v. G. (2010) 'Perspectives on Grid computing', *Future Generation Computer Systems,* 26, pp. 1104-1115.

Scribner, K. and Stiver, M. (2000) *Understanding SOAP: The Authorative Solution.* SAMS: New York.

Seinstra, F. J., Koelma, D. and Geusebroek, J. M. (2002) 'A software architecture for user transparent parallel image processing', *Parallel Computing,* 28, pp. 967-993.

Shapiro, M. and Miller, E. (1999) 'Managing Databases with Binary Large Objects*', 16th IEEE Mass Storage Systems Symposium.* San Diego, California, Managing Databases with Binary Large Objects: pp. 185-193.

Shi, Y., Shortridge, A. and Bartholic, J. (2002) 'Grid Computing for Real Time Distributed Collaborative Geoprocessing*', Geospatial Theory, Processing and Applications (ISPRS Technical Commission IV Symposium).* Grid Computing for Real Time Distributed Collaborative Geoprocessing: ISPRS, pp. 197-208.

Shields, P. (2006) 'Electronic Networks, Enhanced State Surveillance and the Ironies of Control', *Journal of Creative Communications,* 1, (1), pp. 19.

Shu, Y., Zhang, J. and Zhou, X. (2006) 'A Grid-Enabled Architecture for Geospatial Data Sharing*', Services Computing, 2006. APSCC '06. IEEE Asia-Pacific Conference on.* A Grid-Enabled Architecture for Geospatial Data Sharing: pp. 369-375.

Silva, L. E. and Buyya, R. (1999) 'Parallel Programming Models and Paradigms', in Buyya, R.(ed), *High Performance Cluster Computing: Programming and Applications.* Vol. 2 Prentice Hall: NJ, USA, pp. 4-27.

Simonis, I. (2006) *OGC Sensor Alert Service Candidate Implementation Specification.* Open Geospatial Consortium Inc. (06-028r3).

Simonis, I., Dibner, P. C., Walkowski, A., Robin, A., Lansing, J., Greenwood, J., Echterhoff, J., Davidson, J., Priest, M., Botts, M. and Cox, S. (2007) *OpenGIS Sensor Planning Service Implementation Specification.* Open Geospatial Consortium Inc. (07-014r3).

Skillicorn, D. (2002) 'The Case for Datacentric Grids'.*International Parallel and Distributed Processing Symposium (IPDPS).* Fort Lauderdale, Florida, USA,

Snelling, D., Merrill, D. and Savva, A. (2008) *OGSA Basic Security Profile 2.0.* Open Grid Forum (GFD-R-P.138).

soKNOS (2010) *Service-orientierte ArchiteKturen zur Unterstutzung von Netzwerken im Rahmen Oeffentlicher Sicherheit (Service-oriented Architectures Supporting Networks of Public Security)*. Available at: http://www.soknos.de (Accessed: September 2010).

Solheim, A., Bhasin, R., De Blasio, F. V., Blikra, L. H., Boyle, S., Braathen, A., Dehls, J., Elverhoi, A., AEtzelmuller, B., Glimsdal, S., Harbitz, C., Heyerdahl, H., Hoydal, O. A., Iwe, H., Karlsrud, K., Lacasse, S., Lecomte, I., Lindholm, C., Longva, O., Lovholt, F., Nadim, F., Nordal, S., Romstad, B., Roed, J. K. and Strout, J. M. (2005) 'International Centre for Geohazards (ICG): Assessment, prevention and mitigation of geohazards', *Norwegian Journal of Geology,* 85, (1 + 2), pp. 45 - 62.

Sonnet, J. and Savage, C. (2003) *OWS 1.2 SOAP Experiment Report.* OpenGIS Consortium Inc (OGC 03-014).

Sorokine, A., Daniel, J. and Liu, C. (2005) 'Parallel visualization for GIS applications'.*GeoComputation 2005.* Ann Arbor, MI, USA,

Southworth, F. (1991) *Regional Evacuation Modelling: A state of the art review.* Centre for Transportation Analysis: Oak Ridge National Laboratory, USA

Stasch, C., Broring, A. and Walkowski, A. (2008) 'Providing Mobile Sensor Data in a Standardized Way - The SOSmobile Web Service Interface'.*11th AGILE International Conference on Geographic Information Science.* Girona, Spain,

Stolze, K. (2003) 'SQL/MM Spatial: The Standard to Manage Spatial Data in Relational Database Systems'.*10th BTW (Business, Web and Technology)* Leipzig, Germany,

Stonebraker, M. (1986) 'The Case for Shared Nothing', *Database Engineering,* 9, (1).

Stonebraker, M., Abadi, D. J., DeWitt, D., Madden, S., Paulson, E., Pavlo, A. and Rasin, A. (2010) 'MapReduce and Parallel DBMSs: Friends or Foes', *Communications of the ACM,* 53, (1), pp. 64 - 71.

Sun, Q., Chi, T., Wang, X. and Zhong, D. (2005) 'Design of Middleware based Grid GIS'*, IEEE International Geoscience and Remote Sensing Symposium IGARSS.* Seoul, S. Korea, Design of Middleware based Grid GIS: IEEE, pp. 4.

Szalay, A. and Blakeley, J. A. (2009) 'Gray's Law: Database-centric computing in science', in Hey, T., Tansley, S. and Tolle, K.(eds) *The Fourth Paradigm: Data-Intensive Scientific Discovery.* Microsoft Research.

Tatbul, N., Cetintemel, U., Zdonik, S., Cherniack, M. and Stonebraker, M. (2003) 'Load Shedding in a Data Stream Manager'.*29th International Conference on Very Large Databases (VLDB'03).* Berlin, Germany:Morgan Kaufmann.

Tehranian, S., Zhao, Y., Harvey, T., Swaroop, A. and Mckenzie, K. (2006) 'A robust framework for real-time distributed processing of satellite data', *Journal of Parallel and Distributed Computing,* 66, pp. 403-418.

Tham, C. and Buyya, R. (2005) *SensorGrid: Integrating Sensor Networks and Grid Computing.* Melbourne, Australia: Grid Computing and Distributed Systems Laboratory, University of Melbourne

Tiampo, K. F., Rundle, J. B., McGinnis, S. A. and Klein, W. (2002) 'Pattern dynamics and forecast methods in seismically active regions', *Pure Applied Geophysics,* 159, pp. 2429-2467.

Tisato, F. and de Paoli, F. (1995) 'On the Duality between Event-Driven and Time Driven Models'.*13th IFAC DCCS.* Toulouse, France,

Tomlin, C. D. (1991) 'Cartographic Modelling', in *Geographic Information Systems: Principles and Applications.* Longman Scientific and Technical: Essex, UK., pp. 361-74.

Torp, K. and Lahrmann, H. (2005) 'Floating car data for traffic monitoring'*, 5th European Congress and exhibition of intelligent transport systems and services.* Hannover, Germany, Floating car data for traffic monitoring.

Tralli, D. M., Blom, R. G., Zlotnicki, V., Donnellan, A. and Evans, D. L. (2004) 'Satellite remote sensing of earthquake, volcano, flood, landslide and coastal inundation hazards', *ISPRS Journal of Photogrammetry and Remote Sensing,* 59, (4), pp. 185-198.

Trewin, S. M. (1998) 'High-Level Support for Parallel Programming', in Healey, R., Dowers, S., Gittings, B. M. and Mineter, M. J.(eds) *Parallel Processing Algorithms for GIS.* Taylor & Francis.

Tu, Y., Liu, S., Prabhakar, S. and Yao, B. (2006) 'Load Shedding in Stream Databases: A Control-Based Approach'.*32nd International Conference on Very Large Databases.* Seoul, Korea:ACM.

Uslander, T. (2009) *Specification of the Sensor Service Architecture (SensorSA).* Open Geospatial Consortium Inc. (OGC 09-132r1).

Vallecillo, A. (2001) 'RM-ODP: The ISO Reference Model for Open Distributed Processing', *DINTEL Edition on Software Engineering,* 3, pp. 66-99.

Vambenepe, W., Graham, S. and Niblett, P. (2006) *Web Services Topics 1.3.* OASIS (wsn-ws_topics-1.3-spec-os).

van Engelen, R. (2003) 'Pushing the SOAP envelope with Web services for scientific computing'.*International Conference on Web Services (ICWS).* Las Vegas, USA,

van Lint, J. W. C. (2004) *Reliable Travel Time Prediction for Freeways, Bridging Artificial Neural Networks and Traffic Flow Theory.* Ph.D. thesis. Delft University of Technology.

Vaquero, L. M., Rodero-Merino, L., Caceres, J. and Lindner, M. (2008) 'A break in the clouds: towards a cloud definition', *ACM SIGCOMM Computer Communication Review,* 39, (1), pp. 50-55.

Vowles, G. (2007) *Geospatial Digital Rights Management Reference Model (GeoDRM RM).* Open Geospatial Consortium Inc. (06-004r4).

W3C (1999) *Introduction to WSDL*. Available at: http://www.w3c.org (Accessed: 27/11/2007).

Wagner, D. F. and Scott, M. S. (1995) 'Improving the Performance of Raster GIS: A comparison of Approaches to Parallelization of Cost Volume Algorithms'.*AutoCarto* Charlotte, North Carolina, USA:American Society for Photogrammetry and Remote Sensing.

Wang, C. A. and Tsin, Y. H. (1987) 'An O(log n) time parallel algorithm for triangulating a set of points in the place', *Information Processing Letters,* 25, pp. 55-60.

Wang, J. and Gong, H. (2009) *A Load-On-Demand Approach to Handling Large Networks in the Oracle Spatial Network Data Model.* Oracle Corporation

Wang, L., Tao, J., Kunze, M., Kramer, D., Karl, W. and Castellanos, A. C. (2008a) 'Scientific Cloud Computing: Early Definition and Experience*', 10th IEEE International Conference on High Performance Computing and Communications.* Dalian, China, Scientific Cloud Computing: Early Definition and Experience: IEEE, pp. 825-830.

Wang, S. and Armstrong, M. P. (2003) 'A quadtree approach to domain decomposition for spatial interpolation in Grid computing environments', *Parallel Computing,* 29, (10), pp. 1481-1504.

Wang, S. and Armstrong, M. P. (2009) 'A theoretical approach to the use of cyberinfrastructure in geographical analysis', *International Journal of Geographical Information Science (IJGIS),* 23, (2), pp. 169-193.

Wang, X. and Kockelman, K. M. (2009) 'Forecasting Network Data: Spatial Interpolation of Traffic Counts Using Texas Data', *Transportation Research Record: Journal of the Transportation Research Board,* 2105, pp. 100-108.

Wang, Y., Beullens, P., Liu, H., Brown, D., Thornton, T. and Proud, R. (2008b) 'A Practical Intelligent Navigation System based on Travel Speed Prediction'.*11th International IEEE Conference on Intelligent Transportation Systems.* Beijing, China, 12-15 October 2008.IEEE.

Werder, S. and Krüger, A. (2009) 'Parallelizing Geospatial tasks in Grid Computing', *GIS Science: Die Zeitschrift fur Geoinformatik,* 3, pp. 71.

White, C. E., Bernstein, D. and Kornhauser, A. L. (2000) 'Some map matching algorithms for personal navigation assistants', *Transportation Research Part C,* 8, pp. 91-108.

Whiteside, A. (2005) *OpenGIS Web Services Architecture Description.* Open Geospatial Consortium Inc. (05-042r2).

Whiteside, A. and Greenwood, J. (2010) *OGC Web Service Common Standard.* Open Geospatial Consortium Inc. (06-121-r9).

Wilkinson, B. and Allen, M. (1999) *Parallel Programming: Techniques and Applications using Networked Workstations & Parallel Computers.* Prentice Hall: Upper Saddle River, New Jersey, USA.

Williams, M., Cornford, D., Bastin, L., Jones, R. and Parker, S. (2009) 'Automatic processing, quality assurance and serving of real-time weather data over lightweight protocols'.*StatGIS 2009.* Milos, Greece, Wilson, G. (1995) *Parallel Programming for Scientists and Engineers.* MIT Press: Cambridge, MA, USA.

Woolf, A. and Shaon, A. (2009a) 'An approach to encapsulation of Grid Processing within an OGC Web Processing Service', GIS.Science: Die Zeitschrift fur Geoinformatik 3, p.82-88.

Woolf, A. and Shaon, A. (2009b) *Web Processing Service Change Request – method for controlling asynchronous process*. Available at: (https://portal.opengeospatial.org/files/?artifact_id=34550 (Accessed: 07/12/2009).

Woolf, A. and Shaon, A. (2009c) *Web Processing Service Change Request – methods for controlling, and checking the status of asynchronous process*. Available at: https://portal.opengeospatial.org/files/?artifact_id=35070 (Accessed: 07/12/2009).

Worboys, M. and Duckham, M. (eds.) (2004) *GIS: A Computing Perspective.* CRC Press.

Wu, X., Carceroni, R., Fang, H., Zelinka, S. and Kirmse, A. (2007) 'Automatic alignment of large-scale aerial rasters to road-maps'.*15th annual ACM international symposium on Advances in geographic information systems.*

Xing, J., Wu, C., Tao, M., Wu, L. and Zhang, H. (2004) 'Flexible Advance Reservation for Grid Computing', *Lecture Notes in Computer Science,* 3251, pp. 241-248.

Xiong, D. and Marble, D. F. (1996) 'Strategies for real-time spatial analysis using massively parallel SIMD computers: an application to urban traffic flow analysis', *International Journal of Geographical Information Science (IJGIS),* 10, (6), pp. 769-789.

Xue, Y., Wang, Y., Wang, J., Luo, Y., Hu, Y., Zhong, S., Tang, J., Cai, G. and Guan, Y. (2005) 'High Throughput computing for Spatial Information Processing (HIT-SIP) System on Grid Platform', in *Advances in Grid Computing EGC 2005.* Springer Berlin / Heidelberg, pp. 40-49.

Yagoubi, B. and Slimani, Y. (2007) 'Task Load Balancing Strategy for Grid Computing', *Journal of Computer Science,* 3, (3), pp. 186-194.

Yang, H., Dasdan, A., Hsiao, R.-L. and Stott Parker, D. (2007) 'Map-reduce-merge: simplified relational data processing on large clusters'.*2007 ACM SIGMOD International Conference on Management of Data.* Beijing, China:ACM.

Yang, J., Kang, S. and Chon, K. (2005) 'The map matching algorithm of gps data with relatively long polling time intervals', *Journal of the Eastern Asia Society for Transportation Studies,* 6, pp. 2561-2573.

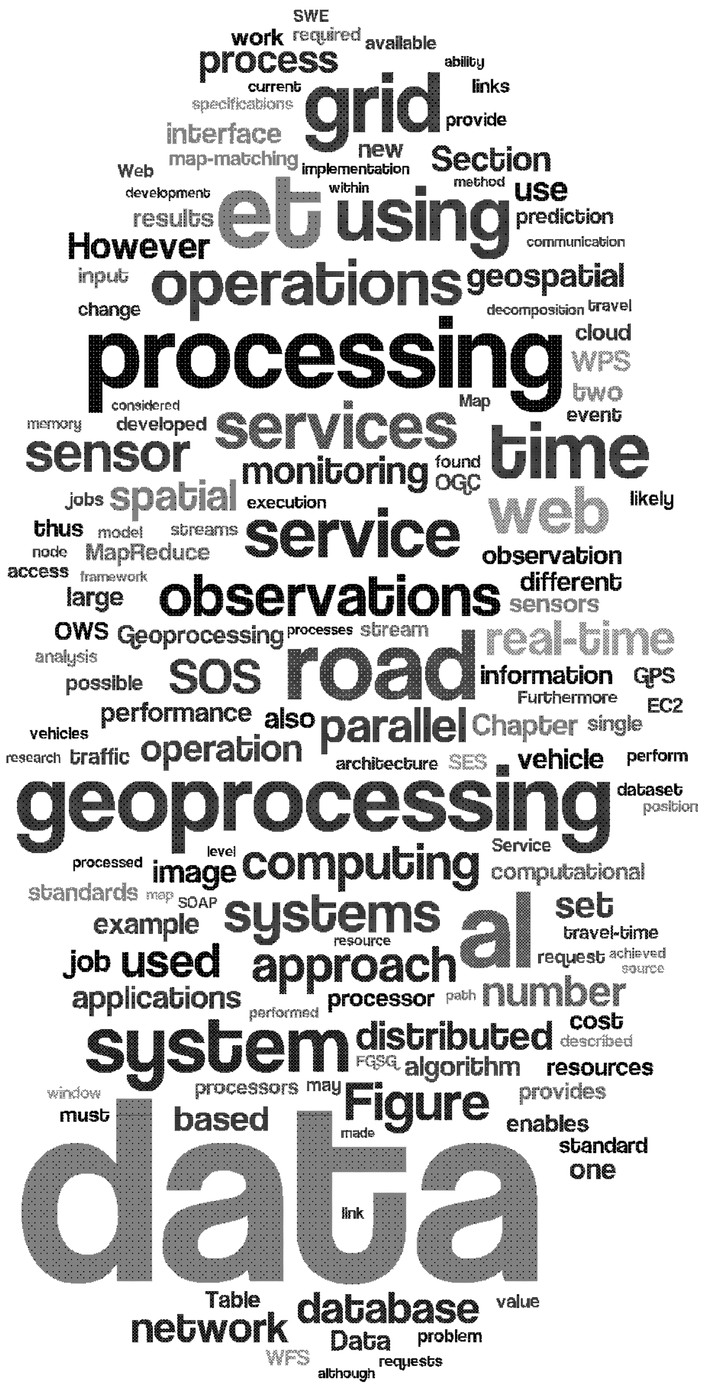
Zerger, A. and Smith, D. I. (2003) 'Impediments to using GIS for real-time disaster decision support', *Computers, Environment and Urban Systems,* 27, pp. 123-141.

Zhao, C., Zhao, Y., Meng, L. and Deng, S. (2005) 'The Key Techologic Issues of Parallel Spatial Database Management System for Parallel GIS'.*XXXVI ISPRS International Symposium on Spatio-temporal Modeling, Spatial Reasoning, Analysis, Data Mining and Data Fusion.* Peking University, China:ISPRS.

Zhou, G., Esaki, T., Mitani, Y., Xie, M. and Mori, J. (2003) 'Spatial probabilistic modeling of slope failure using an integrated GIS Monte Carlo simulation approach', *Engineering Geology,* 68, (3-4), pp. 373-386.

Zhou, X., Abel, D. J. and Truffet, D. (1998) 'Data Partitioning for Parallel Spatial Join Processing', *Geoinformatica,* 2, (2), pp. 175-204.

Zhu, H., Liu, H., Jiang, T. and Zhang, J. (2009) 'Research on remote sensing data processing strategy and application based on grid operation', *Procedia Earth and Planetary Science,* 1, pp. 1180-1185.


SWE
work
required
available
process
ability
current
links
specifications
grid
provide
interface
new
Section
map-matching
implementation
Web
method
use
development
within
et
using
results
prediction
communication
However
input
operations
geospatial
change
decomposition
travel
cloud
processing
WPS
two
considered
Map
memory
developed
services
event
sensor
time
monitoring
found
OGC
spatial
execution
jobs
web
likely
thus
model
streams
service
node
MapReduce
observation
access
framework
different
large
observations
sensors
OWS
Geoprocessing
processes
stream
real-time
analysis
possible
SOS
road
information
GPS
Furthermore
EC2
performance
also
parallel
Chapter
single
vehicles
traffic
operation
architecture
SES
vehicle
perform
research
geoprocessing
dataset
position
level
Service
processed
image
computing
computational
standards
map
SOAP
set
example
systems
al
travel-time
resource
job
used
approach
request
achieved
source
applications
number
performed
processor
path
cost
system
distributed
described
FGSG
algorithm
resources
window
processors
may
provides
must
based
Figure
enables
made
standard
one
data
link
Table
value
database
network
Data
problem
WFS
requests
although

THE END