Newcastle University

School of Computing Science

Partitioning Workflow Applications over Federated Clouds to meet Non-Functional Requirements

Zhenyu Wen

Submitted for the degree of Doctor of Philosophy in the School of Computing Science, Newcastle University

June 2016

Abstract

With cloud computing, users can acquire computer resources when they need them on a pay-as-you-go business model. Because of this, many applications are now being deployed in the cloud, and there are many different cloud providers worldwide. Importantly, all these various infrastructure providers offer services with different levels of quality. For example, cloud data centres are governed by the privacy and security policies of the country where the centre is located, while many organisations have created their own internal "private cloud" to meet security needs.

With all this varieties and uncertainties, application developers who decide to host their system in the cloud face the issue of which cloud to choose to get the best operational conditions in terms of price, reliability and security. And the decision becomes even more complicated if their application consists of a number of distributed components, each with slightly different requirements.

Rather than trying to identify the single best cloud for an application, this thesis considers an alternative approach, that is, combining different clouds to meet users' non-functional requirements. Cloud federation offers the ability to distribute a single application across two or more clouds, so that the application can benefit from the advantages of each one of them. The key challenge for this approach is how to find the distribution (or deployment) of application components, which can yield the greatest benefits. In this thesis, we tackle this problem and propose a set of algorithms, and a framework, to partition a workflow-based application over federated clouds in order to exploit the strengths of each cloud. The specific goal is to split a distributed application structured as a workflow such that the security and reliability requirements of each component are met, whilst the overall cost of execution is minimised.

To achieve this, we propose and evaluate a cloud broker for partitioning a workflow application over federated clouds. The broker integrates with the e-Science Central cloud platform to automatically deploy a workflow over public and private clouds.

We developed a deployment planning algorithm to partition a large workflow application across federated clouds so as to meet security requirements and minimise the monetary cost.

A more generic framework is then proposed to model, quantify and guide the partitioning and deployment of workflows over federated clouds. This framework considers the situation where changes in cloud availability (including cloud failure) arise during workflow execution.

Declaration

I declare that this thesis is my own work unless otherwise stated. No part of this thesis has previously been submitted for a degree or any other qualification at Newcastle University or any other institution.

Zhenyu Wen

June 2016

Publications

Portions of the work within this thesis have been documented in the following publications:

JOURNAL

1. **Z.Wen**, J.Cala, P.Watson and A.Romanovsky, *Cost Effective, Reliable and Secure Workflow Deployment over Federated Clouds*, submitted to IEEE Transactions on Services Computing

CONFERENCE

1. **Z.Wen**, J.Cala, P.Watson and A.Romanovsky, *Cost Effective, Reliable and Secure Workflow Deployment over Federated Clouds* The IEEE International Conference on Cloud Computing (CLOUD), June 2015
2. **Z.Wen**, J. Cala and P.Watson, *A Scalable Method for Partitioning Workflows with Security Requirements over Federated Clouds*, The IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Dec 2014.
3. **Z.Wen** and P. Watson, *Dynamic Exception Handling for Partitioned Workflow on Federated Cloud*, The IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Dec 2013.

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Paul Watson, for his patient guidance, motivating encouragements and unreserved advice throughout my whole PhD study. He has supported me academically and emotionally from the beginning to the end of finishing this thesis. I have been extremely fortunate to have such a supervisor who cares so much about not only my research work, but also my personal career. Furthermore, I wish to thank Prof. Alexander Romanovsky who has been demonstrative and generous in his guidance and support throughout my whole PhD study. I am very grateful to Dr. Jacek Cala for his guidance and valuable suggestions on my research. I also wish to give thanks to Dr. Michael Rovatsos for offering an opportunity for my future research.

I would also like to thank my examiners, Dr. Adam Barker and Dr. Paul Ezhilchelvan, who provided encouraging and constructive feedbacks. It is not an easy task, reviewing a thesis, and I am grateful for their thoughtful and detailed comments.

I would like to thank my office-mates: Tudor Miu, Rawaa Qasha, Hugo Firth, Faris Llwaah, Rebecca Simmonds and Lesego Peter; and my friends in the School of Computing Science: Bowen Li, Qi Shen, Zequn Li, Xingjie Wei, Shidong Wang, Tong xin and Matthew Forshaw for their kind help, support and company. I am also grateful for my YD group: Shiwei He, Qiwei Wu, Wenqi Deng, Yi Liu, and friends in Newcastle: Pengcheng Zeng, Haimeng Wu, Yuwei Liang, Yaxi Ye, Jun Ye, Nanlin Jin, who granted me great support and patience and really made my PhD colourfully.

Additionally, I wish to thank the brothers and sisters in Newcastle Chinese Christian fellowship for their prayers and supports during my hardest time of my study, and also I would like to thanks buddies from CS United football team and Chinese United football team, who made my PhD experience brighter.

Finally, and most specially, I would like to thank my parents for their inspiration, patience and support over the years. In particular, I am very grateful to my fiancee Qi for her continuous support and motivation on the long journey.

Contents

List of Figures

List of Tables

1

Introduction

Contents

Introduction

The importance of scientific computing has increased significantly over the past decades, encouraged by rapid advances in the speed and capacity of computer technology. This has led to increasing amounts of data being stored and processed, and as a result the field of e-Science has grown to address large-scale scientific data processing issues.

The scientific workflow now plays a crucial role in large-scale scientific processing, data modelling and operations such as loading input data, data analysis and outputting results. Typically, a scientific workflow is represented as a directed graph of operations. Using a workflow encourages re-use of operations, and prevents scientists from having to implement complex processing mechanisms such as scheduling operations and respecting processing dependencies. In some cases it also provides further facilities such as provenance capture [81] [9].

In recent years, Cloud computing has provided users with cheap, elastic and diverse computing resources. This allows scientists to easily hire the resources they need to store data and execute workflows. As a result, cloud computing has been exploited as the main computing infrastructure underpinning scientific workflows.

However, existing systems are unable to coordinate processing over different clouds in order to optimally allocate application services to meet users' requirements. A collection of a set of clouds is usually called "cloud federation",and these can, for example, be exploited to meet security and reliability requirements, where work is allocated to different clouds based on their security levels and data are replicated across clouds to meet availability needs.

Cloud security is one of the key issues affecting customers' decision-making concerning which applications should be deployed in the cloud. The influential Berkeley report [12] placed data confidentiality and audibility high on the list of concerns that may deter organisations from moving to or creating applications in the cloud. A Cloud Security Alliance report [8] indicated that a malicious insider is one of the primary security concerns, and one risk is that customers may lose direct control over potentially business sensitive and confidential data. Furthermore, such dears are made worse by the lack of transparency cloud providers' processes and procedures. For example, customers do not know how the cloud providers access physical and virtual assets, how they monitor the status of their data-centres, and how they analyse and report on policy compliance. This may put customers' sensitive data at risk and, in more serious situations, result in financial impacts, brand damage, and huge productivity losses [8].

Cloud reliability is another key issue that users may consider before moving to cloud computing. For example, the common SLA (service-level agreement) for cloud computing is 99.95%. In practice, this means that, in any given month, the service can only be offline for about 20 minutes, which represents only about 250 minutes per year.

In early 2011, several high-profile technical companies experienced problems when Amazon's EC2 service suffered an outage [2]. This lasted for almost 11 hours, which violated the 99.95% SLA. In addition, an outage at GoDaddy took down millions of websites [62]

Considering these issues, the cloud federation has the potential to provide a solution that facilitates just-in-time, opportunistic, and scalable provisioning of application services, consistently fulfilling user requirements under variable workload, resource and network conditions [27]. Using a federated cloud, users are able to deploy their applications over a set of clouds from different cloud providers, across different geographical locations, therefore bringing various advantages such as leveraging unique cloud specific services, providing higher availability and redundancy, disaster recovery and geo-presence. However, to achieve this, several unsolved problems need to be addressed; these are the basis for the research questions that are addressed in this thesis.

1.1 Research Question

With the variety of clouds now available, application developers who decide to host their system on a cloud face the issue of how to choose the best platform in terms of price, reliability and security. The decision becomes even more complicated if the application consists of a number of distributed components, each with different requirements.

Considering these problems, cloud federation enables a single application to be dis-

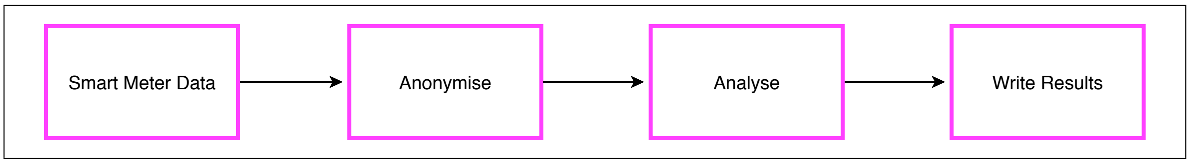


Figure 1.1: A Smart Meter Analysis Workflow

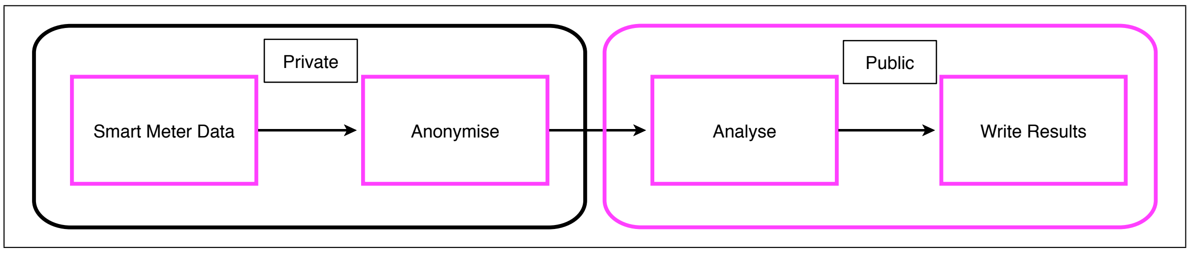


Figure 1.2: Deploying the Smart Meter Analysis Workflow Over Federated Cloud

tributed on two or more clouds, so that the application can combine the benefits from each cloud [116]. For example, consider the smart meter analysis workflow, run by an energy company, shown in Figure 1.1. This consists of four tasks. The "Smart Meter Data" reads customers' personal information and so a decision is made to run it on a private cloud. However, the "Analyse" task operates on anonymised data, but is computationally intensive which may result in the energy company spending a large amount of money if they have to upgrade their internal private cloud hardware to support this analysis for many customers simultaneously. Furthermore, it is likely that due to fluctuations in demand, these resources will be idle for much of the time. Meeting the security requirements while minimising cost can be achieved by cloud federation. The workflow is split into two partitions as shown in Figure 1.2. The sensitive data is stored and anonymised in the private cloud but then processed in the public cloud, benefiting from the scalable computing resources and the pay-as-you-go model.

Currently, this partitioning and deployment must be carried out in a manual and ad-hoc manner, with a human expert considering possible deployment plans and deciding if they meet the security requirements while driving down costs. However, human error may compromise security, and the decision cannot be verified because of the lack of an auditable explanation [144]. So the main goal in this thesis is to explore *How to distribute (or deploy) a workflow over a federated cloud in order to meet specific requirements.*

In this thesis, an attempt is made to tackle this problem and propose a set of algorithms and a framework to partition workflow-based applications over a federated cloud in order to exploit the strengths of each constituent cloud. The following research questions are explored:

- How can the security levels of the clouds, services and data be defined?
- How can the reliability of the clouds and services be defined and measured?
- How can the services and data be mapped to the available clouds to meet different requirements?
- How can the dynamics of cloud federation be handled; for example, when a cloud fails, joins or leaves the federation?

Taking into account the research queastion above, the main contributions of this thesis are described in next section.

1.2 Contributions

Two novel static algorithms (considering security, reliability and monetary cost) are proposed to partition scientific workflow over a federated cloud. In addition, a generic framework is also developed to dynamically deploy workflow applications over a federated cloud. A cloud broker is designed and implemented to interact with a set of cloud based workflow platforms. The main contributions made are as follows:

1. An overview is given of existing work on scientific workflow resource allocation and deployment in cloud computing. The existing methods or algorithms are categorised and analysed.
2. A cloud broker is designed and implemented to partition workflow applications over a federated cloud. The broker is evaluated by interfacing it to e-Science Central (thereafter e-SC)- which is a science cloud platform [81] used to deploy e-SC workflows over public and private clouds.

3. A low complexity deployment planning algorithm is proposed to partition a large workflow application across federated clouds so as to meet security requirements and reduce the price paid for executing the workflow. The algorithm takes into account the three main sources of financial cost in the cloud: computation, data transfer and data storage.

4. A framework is designed and implemented for modelling, quantifying and guiding the partitioning and deployment of a workflow over a federated clouds to meet security requirements while minimising costs. The framework also considers the situation where a change arises in the set of available clouds during the execution of the workflow.

5. An algorithm is designed and evaluated to partition a workflow application across federated clouds taking into account reliability as well as security requirements, while reducing the monetary cost incurred by computation, data transfer and data storage.

6. A simulation tool is implemented by combining WorkflowSim [40] and DynamicCloudSim [26].

7. The algorithms and the framework are evaluated by using this simulation tool, and an implementation of the cloud broker.

1.3 Thesis Structure

Chapter 1 describes the motivation behind the work carried out as part of this thesis, and highlights the main contributions of the research.

Chapter 2 presents background material and a summary of work closely related to the original research described in this thesis.

Chapter 3 describes a new automatic architecture for dynamically partitioning applications across a set of clouds in an environment in which clouds can fail during workflow execution.

Chapter 4 proposes a novel algorithm to deploy complex workflow-based applications over federated clouds, while meeting security requirements and minimising financial costs.

Chapter 5 explores a workflow deployment framework to dynamically deploy scientific workflows over federated clouds to meet security requirements while minimising financial costs. The framework employs a set of static algorithms for optimising the deployment of workflows over federated clouds, and a dynamic rescheduling algorithm for handling changes in cloud availability.

Chapter 6 proposes a novel algorithm to deploy workflow applications on federated clouds, taking into account security, reliability and monetary cost. Firstly,an entropy-based method is introduced to quantify the reliability of workflow deployments. Secondly, an extension of the Bell-LaPadula Multi-Level security model is applied to meet application security requirements. Finally, deployment is optimised in terms of its entropy and also its monetary cost, taking into account the cost of computing power, data storage and inter-cloud communication.

Chapter 7 summarises and provides the conclusion of the work presented in this thesis and proposes directions for further work in the area.

2

Literature review

Contents

Summary

This chapter starts by describing some of the background information concerning the overall topic, including a brief primer on cloud computing, cloud federation, workflow systems and workflow deployment. In section 2.5, research in area of *workflow deployment optimisation* which relates to the main focus of this thesis is investigated. At the same time, gaps in current research are highlighted and is then briefly illustrated and how this thesis fills these gaps.

2.1 Cloud Computing

Cloud computing has become another buzzword, after Web 2.0, and yet it has various definitions. However, there is some consensus on defining the cloud [11], [63] as *A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualised, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.*

This transforms computing into an utility models much like water, electricity, gas and telephony. In such a model, users can access services according to their requirements regardless of where the services are hosted and how they are delivered. Furthermore, using a pay-as-you-go fashion, users can pay for their computing resources on demand.

Services in cloud computing have been defined in terms of three popular computing models [115]: 1) *Infrastructure as a Service (IaaS)*– the cloud resources are offered in the form of independent raw virtual machines. Users take the responsibility for and have the flexibility to install operating systems and software, such as Amazon EC2 [83]and Microsoft Azure [103], as required 2) *Platform as a Service (PaaS)*– in this model, the cloud providers offer a computing platform which consists of operating systems, programming language execution environments, web servers and databases. This provides an environment for developers to create, host and deploy applications, saving developers from managing the complexities of the infrastructures (setting up and configuring). Examples of PaaS include Google App Engine (GAE) [84] and Slesforce.com [85] 3) *Software as a Service (SaaS)*– in this model, cloud providers install and operate application software on the cloud and end users can access the software from cloud clients. Examples of SaaS include Microsoft Office 365 and Dropbox.

While cloud computing brings numerous promises and benefits, it also introduces many problems and challenges. In paper [11], the authors introduced ten prominent obstacles that cloud computing faces, including Availability, Data Lock-in, Data Confidentiality and Cloud Audibility, Data Transfer Bottlenecks, Performance Unpredictability, Scalable Storage, Bugs in Large Distributed Systems, Scaling Quickly, Reputation Fate Sharing and Software Licensing. There are currently increasing numbers of researchers dedicated to the exploration and discovery of novel but practical solutions to overcome these issues.

2.2 Cloud Federation

Cloud computing as a paradigm aims to provide on-demand elasticity at different levels. Cloud federation offers the potential for the just-in-time, opportunistic, and scalable provisioning of application services in order to achieve Quality of Service (QoS) targets under variable workload, resource and network conditions. It can provide support for the dynamic expansion or contraction of capabilities to handle variations in users' demands.

However, studies of cloud federation are relatively new in the field of cloud computing. So far, the literatures is sparse and some of the concepts involved are yet to be clearly established. Buyya [27] mentioned that a cloud federation must have the following three characteristics: (1) the capacity to dynamically expand or resize resources to fulfil incoming demand; (2) the ability to operate as part of a market directed to resource lending; and (3)the capacity to deliver reliable services at competitive costs and complying with established quality of service requirements.

The authors of papers [14], [88] and [60] have pointed out that federated cloud service providers who joined together to offer more resources to their users can mitigate problems related to lack of Service Level Agreement (SLA) compliance. Therefore, the federated cloud can provide the following advantages: (i) *performance guarantees* – by lending resources, it is possible to maintain the necessary levels of performance for the services they render; (ii) *availability guarantees* – location diversity for data and services allows the migration of services out from one cloud to another, for example in disaster prone areas, maintaining higher availability for the client; (iii) *convenience* – federation is convenient to clients as it provides a unified view of the services of different providers; and (iv) *dynamic workload distribution* – geographic distribution makes it possible to spread the load taking into account the client's location.

Manno et al. [112] represent cloud federation as a geographically dispersed community with the shared resources available to achieve a common object in terms of the federation's contract. This contract includes its economic and technical aspects, defining the policies, restrictions, standards and penalties related of cooperation. Moreover, each cloud, which is seen as an autonomous domain, can leave the community any time as needed.

Consequently, a cloud federation is an inter-cloud organisation with its own independence. It is presented as a geographically dispersed and well-defined commercial system, composed of a set of autonomous and heterogeneous clouds with a federal contract. It is able to offer sufficient and effective resources to meet users' functional requirements (typically hardware) and non-functional requirements so as to realise the dynamic distribution of participation. To design a set of federated clouds, two issues must be considered: interoperability and the federation model. These are looked into in detail next.

2.2.1 Interoperability

Interoperability provides the opportunity to create a federation, as it allows systems in the community to interact with each other. An investigation of existing clouds shows that cloud providers offer application programming interfaces (APIs) for their customers to interact with their cloud services, whereas there is a lack of standard APIs fostering cloud interoperability. Thus, building mediator components among all the interacting providers in federation is a possible solution in such a situation.

The Service Oriented Architecture (SOA) can be a lesson for providing interoperability in a federated cloud [139]. Here, each cloud operates as a service with standard APIs, and the communication among services is based on the remote procedure call (e.g. REST, RMI).

2.2.2 Federation Model

The cloud federation model is becoming more popular as the cloud paradigm matures. This section discusses four different approaches to model cloud federation which vary in terms of business focus, internal components, the service of interest, and the target market:

- *Semantics based*: A federated cloud model has been proposed in [112] based on semantics in order to integrate heterogeneous resources which are deployed as services in different clouds. An specific ontology is utilised to present each cloud as an independent instantiation. However, these instantiations can interact within a shared resource environment. The main challenge in utilising this model concens interoperability, which is a critical aspect of cloud federation. Therefore, the ontology used must be able to understand and model the differences between the various implementation schemes used by each cloud.

- *Market-oriented*: Buyya et al [28]. introduced a model that structures the cloud infrastructure resources as a service market. The model includes four main components: (1) Cloud providers: are where resources are located and services are offered to users. (2) Cloud broker: is a middleware responsible for interaction and operation of users and the federated Clouds. (3) Cloud coordinator: is a located in each cloud responsible for maintaining and managing the federated community. (4) Concentrator: prices the resources and acts as the market mechanism.

- *Reservoir*: this model aims to provide the cloud resources as an utility analogous to electricity [148] [126]. In this model, a number of cloud providers offer computing resources, which are consumed and made available as a commodity by a distributor. Users interact with the distributor to access the resources. This eliminates limitations such as the lack of interoperability, the difficulties that small providers have in providing scalability, and the lack of support for business service management (an approach which is used to manage business IT services) [151].

- *Service-layers-oriented*: It has been proposed [140] that layered cloud services (Saas, PaaS and IaaS) can leverage multiple independent clouds by creating a federation. The authors mentioned that the federation of clouds should be isolated into identical layers - each of which has a broker, and which offers a different service to meet user requirements. For example, a SaaS broker is mainly based on the users' requirements and Service Level Agreements (SLA) between different cloud providers. Meanwhile a PaaS broker is mainly based on the application's requirements in terms of deployment (e.g. compilation framework) and runtime support (e.g. libraries), and an IaaS broker is mainly based on combining different types of resources such as VMs (with different cpu, ram and storage space etc) from different locations.

2.3 Scientific Workflow Systems

Scientific researchers now run increasingly complex, data-intensive simulations and analyses and there is a corresponding increase in the use by scientific communities of workflow technologies to manage this complexity [135]. Workflows are used to schedule computing tasks on shared resources, manage dependencies among tasks, and stage datasets in and out of execution sites.

The workflow life cycle can be classified in four areas [45]: composition, mapping, execution and provenance. The following sections, focus on these four phases and give a general taxonomy of scientific workflow.

2.3.1 Workflow Composition

The workflow allows users to represent the steps and dependencies required for a desired analysis. In some cases workflows are designed by representing both the computational steps and the data flows. In other cases, the composition is split into two levels: one an abstract level describing the high-level workflow, and the other consists of instances of the abstract level with actual data [81]. According to the existing literature, current methods for composing workflows can be categorised as follows.

- *Textual workflow editing*: these are workflow systems that can be composed by using a particular workflow language through a plain text editor. For the scientific workflow, the script is usually generated by a high-level language such as Python or Ruby which generate the lower-level workflow primitives. For example, the workflows in Pegasus [47] are represented in a form of a Directed Acyclic Graph in XML format (DAX), and DAX can be generated by any type of scripting language.

- *Graphical workflow editing*: workflow systems provide a graphical tool for creating workflows. This can simplify the lives of scientific users. However, graphical renderings are only feasible for small workflows with fewer than a dozen tasks. To solve this problem, most graphical tools such as Kepler [9], Triana [134] and

Vistrails [33] allow some form of graphical nesting based on sub-workflow hierarchies.

- *Semantic composition* of a workflow provides an approach to represent and reason about workflows and their components so that [69]:
  - **workflow templates and instances are semantic objects**
  - **data collections are specified with intensional descriptions in workflow templates and extensional descriptions in workflow instances**
  - **intensional descriptions of node collections that offer appropriate abstractions for the repetitive structure of the workflows**

  Therefore, scientists must not only to describe the constraints of control flow, but also those of the involved tasks in order to enable a workflow engine to discover concrete services during runtime. Furthermore, the description of existing services with semantics for discovery, selection, invocation and composition is also necessary. A typical semantic workflow system such as Wings [68] combines semantic representations of workflow components with the formal properties of correct workflows with the assistance of artificial intelligence planning techniques [67].

2.3.2 Mapping a Workflow to Computational Resources

A workflow describes the process flow of a scientific analysis, and this then needs to be mapped to computing resources. This can be performed by the user in some cases, but can also be done by the workflow system directly. Even in the latter case, users may also be able to direct the distribution of their workflow to target execution environments through low-level commands. Therefore, depending on the execution model, the mapping consists of finding and binding the execution resources for the high-level tasks or applications to meet functional or non-functional requirements.

In general, resources are represented as computing resources, such as Cloud, High performance computer (HPC) or workstation. However, workflow systems have a different meaning regarding the concept of resource, which can even be a person; for example, where a step of a workflow requires a scientist to verify the correctness of the processing. In other cases, the resources may be a required application that is deployed, requiring the outputs of the application as inputs.

Mapping the workflow onto available resources needs a broker to schedule jobs or tasks to the resources. Brokers may be external or internal. The external broker is not included as part of the workflow system. For example, P-GRADE [94] uses DAGMan to control the node dependencies of the workflow graph and provides interfaces to support other brokers, such as GTBroker [95], LHC-Broker [61], and gLite-broker [34], to schedule the workflow onto execution environments. Another type of brokers are included in the workflow system, such as Trinan and Karajan[64], and these supports the dynamic binding of their workflow tasks or services to resources. These two types of broker both support the mapping of a workflow to multiple distributed resources in order to simplify the orchestration process for distributed execution. The former is more flexible than the latter, and can support different workflow systems, and the distribution of their tasks to computing resources. However, to support different workflow systems, the workflow APIs need to be bound with a broker, which may make the broker too complicated, and this may therefore impacting on the usage.

Adam Barker et al. [20] have proposed a MultiAgent Protocol (MAP) to a capture scientific process. This allows a scientific workflow to be executed in an agent-based, decentralised, peer-to-peer architecture. A Large Synoptic Survey Telescope (LSST) was used to demonstrate how the agent-based approach is helpful to classify previously unknown objects. Each agent taking part in the interaction adopts a role, according to which the agent refers to a reasoning Web Service that implements all the decision procedures required for that role type.

2.3.3 Workflow Execution

Workflows are executed through the workflow execution engine and the enactment subsystem. Pegasus and Askalon can distribute their tasks or services to remote resources, and use a workflow engine (such as DAGMan) for the execution of jobs. Besides these, the Java CoG kit [4] supports hierarchical workflows based on GAGs. This design allows each subsystem to be submitted to different resources to achieve scalability.

Scalability has been demonstrated by scaling hundreds of thousands of jobs.

2.3.4 Provenance

Provenance in workflow means the provenance of data, which records the history of the creation of a data object. It includes the chain/graph of processes such as the time stamp, program version number, component or service version number, execution host and library versions, etc, as well as intermediate data products back to the source data used to initialise the workflow.

For the scientific workflow, data provenance gives users the capacity to reproduce and verify the results, which is a crucial component of the scientific method. Furthermore, it also has benefits for the workflow optimisation. For example, based on the execution logs of a workflow, the makespam of this workflow can be optimised.

Data provenance is also required for transformed workflow execution; where for example, the inputs of a workflow are the outputs of the components of the executed workflow. Therefore by tracking the provenance of the executed workflow, the outputs can be reused immediately.

However, the challenges facing data provenance is that a workflow system needs different ways to record provenance information in order to interact with various workflow systems. Some systems use internal structures to manage provenance information, while others rely on external services which can be very generic.

Triana, for example, has its own internal format to record provenance information and also to interact with external services [37]. Karma [130] is a provenance system for representing large independent workflow. It can gather data from various workflow systems in several different ways, and provides a searchable database of data provenance that has extensive capabilities for formulating data provenance queries.

2.4 Deploying Workflow Systems on the Cloud

The previous section, we has described how the workflow has to be mapped to the execution environments to complete the process. As the cloud becomes an attractive host for computing resources, workflow systems are increasingly being moved into the cloud. Before introducing the deployment of cloud workflow systems, some tools which can automatically provision cloud resources, which are normally virtual machines, have to be considered.

2.4.1 Cloud Provisioning Tools

Cloud provisioning tools are designed to allow the automagical setting-up of VM.

Eucalyptus [3] is designed to be an open-source answer to the commercial EC2 cloud. It saves users from having to deal with complex underlying systems by asking users to set the size of available processors, memory and hard drive space, and then the VMs can be automatically set up. Furthermore, Eucalyptus also provides software configuration, distributed storage system and network configuration to enable users to easily maintain and manage their Amazon VMs.

OpenNebula [1] permits a huge amount of customisability. It exposes more of the underlying features of libvirt to cloud users and administrators. For example, it provides a shared file system for all disk images and all files for running OpenNebula functions. To initialise a VM, users have to provide a configuration file containing parameters which will be fed into the VM command line, allowing the configuration of any required resources such as memory, processors, networks, disk space and so on. Furthermore, OpenNebula is suitable for researchers in computer science who wish to conduct experiments combining cloud systems with other technologies.

Nimbus [7] is affiliated to the Globus project, which is used as a disk image repository on GridFTP (one of the Globus projects). There are some customisations available in Nimbus, yet it is different from OpenNebula. In Nimbus the customisations are mostly open exclusively to the administrator, and the underlying technical details of VM creation are more protected. Moreover, among these three tools, Nimbus is the one which pays most attention to capacity optimisation. It gives different users different lease limits of hiding VMs, so Nimbus needs the support of scheduling. After Cloud resources have been provisioned by the tools either automatically or manually, scientific workflow systems can be deployed and executed on the Clouds. The following section, reviews existing research on how to deploy workflows on Clouds.

2.4.2 Deployment of Cloud Workflow Systems

Aneka [138] is a cloud platform for .NET applications. The Gridbus workflow management system was used in this platform to represent and maintain workflows, Aneka can also be used to distribute jobs to a cloud federation.

Galaxy [104] is one of the most widely used scientific workflow management systems. It is an open-source web portal platform, addressing the problem of accessibility, reproducibility and transparency. The system aggregates bioinformatics tools and connects them in pipelines. Galaxy consists of three aspects: computational analysis, publishing workflow and sharing data, and extending researcher tools with the help of system deployment. The Galaxy workflow system has been developed by adding a number of tools to achieve data management capabilities, domain specific analyses, automatic deployment on the cloud, and support for validating the correctness of workflows. The automatic deployment of Galaxy on the cloud is achieved by using a Globus Provision-based method [10], which is a tool for automatically deploying distributed computing system.

Madduri et.al [109] demonstrated how to use Globus to deploy Galaxy on EC2. The Globus Online file transfer system is used to transfer the data from remote sequencing centre or database, and the On-demand computing resources were provided by Amazon EC2 via the Globus provision cloud manager.

PRECIP (Pegasus Repeatable Experiments for the Cloud in Python) [18] plays the role of a management API to support cloud interoperability, whereby researchers can run their experiments across multiple clouds. Furthermore, this tool allows experiments to be automatically logged and reproduced, and it provides a simple way to maintain computing resources and basic fault tolerant mechanisms.

Vockler et al. [141] has demonstrated how to execute a scientific workflow application in a set of clouds. Firstly, well-configured workflow execution environments which are deployed over a set of clouds can be prepared either by hand or some automated tools such as Nimbus Context Broker [92], Wrangler [91] and others. Secondly, Pegasus analyses the abstract workflow and maps them onto the available resources. The Workflow is executed by using DAGMan to add the executable workflow to a Condor queue, and then Condor sends the jobs to the available Condor workers (the available clouds). Finally, the cloud resources are released either manually or by using tools after the workflow execution was finished.

CloudDragon [157] is a cloud platform based on OpenNebula that can be integrated with a scientific workflow management system, automating scientific analysis and discovery on clouds. Furthermore, the platform provides a set of approaches to recycle cloud virtual machine instances and improves the efficiency of the integration between the workflow system and clouds.

2.5 Workflow Deployment Optimisation

Deployment optimisation is an activity that takes into account the requirements of the software to be deployed along with the resources of the target environment on which the software will be executed, and decides on the details of implementation and how and where the software will be run in that environment [119]. Therefore, through optimisation, the workflow deployment or execution can lead to: (1) reduction of monetary costs and execution makespan (execution time), (2) increase in the security and reliability, (3) improvement in QoS, (4) improvement in resource utilisation. Paper [100] shows that by using MER (Maximum Effective Reduction) algorithm, resource consumption can be reduced by 54% but only increasing makespan by 10%.

2.5.1 Financial Cost-driven Workflow Deployment

Cloud computing provides a pay-as-you-go model. Therefore, in terms of cloud providers, the pricing should be well-defined in order to attract users and save their operation costs. Meanwhile users want to use the least money to meet their requirements (such as performance). This thesis attempts to investigate a framework for the optimisation of workflow deployment, it therefore, existing studies attempting to minimise workflow execution cost in cloud computing are reviewed as follows.

2.5.1.1 Single Cloud

Microsoft Azure provides various VMs at different prices, but in general the more powerful VMs are more expensive. However, the most powerful VM may not meet the performance requirements for executing a scientific workflow. Therefore, distributing the workflows (or tasks of a workflow) over different VMs can increase the execution performance and reduce the monetary cost (if possible). These studies discussed bellow aimed to optimise the cost of execution through scheduling workflows (or tasks of a workflow) over a set of VMs within a single cloud provider.

Zhou and He in [160] introduced a workflow optimisation algorithm to minimise the cost of running a workflow on the cloud while guaranteeing a deadline. The optimisation was achieved by modifying the structure of the workflow. For example, executing two parallel tasks at the same time usually requires hiring two VMs. However, if these two tasks are running on more powerful VMs in sequence, it may cheaper and faster than running them in parallel.

Mao et al, in [113] applied an “auto-scaling” mechanism to assign tasks to the cost-efficient VMs in order to meet the deadline and minimise the monetary cost.

The authors of another study [30], they first described an Integer Linear programming (ILP) model to represent the optimisation problem in terms of referring to cost and performance. Next, forward and backward scheduling methods were applied to allocate the tasks of the target workflow to VMs based on the results generated from the ILP model.

In [120] and [76], the authors have presented a heuristic algorithm based on particle swarm optimisation (PSO) to optimise execution time (including data transfer and processing time) and the monetary cost of workflow processing on a cloud.

Byun et al.[29] developed the Partitioned Balanced Time Scheduling (PBTS) algorithm for cost optimisation and deadline constrains in the execution of a workflow on a cloud. The cost is reduced by minimising the number of VMs hired through partitioning the tasks into different groups.

Montage [44] users often need several workflows with different parameters to generate a set of image mosaics that can be combined into a single, large mosaic. The Galactic Plane ensemble [6], which generates several mosaics at different wavelengths, consists of 17 workflows each of which contains 900 sub-workflows. The workflows or sub-workflows also have dependencies to each other, for example, where a workflow’s outputs are the inputs of another workflow. Therefore, maximising the number of completed workflows from an ensemble under both budget and deadline constraints is also a very important research problem.

DPDS (dynamic provisioning dynamic scheduling) and SPSS (static provisioning static scheduling) algorithms have been introduced to solve this problem [110] [111]. The DPDS is an online scheduling algorithm which is based on resource utilisation to adjust the numbers of VMs according to how well they are utilised by the workflows. A priority queue was used to store workflows which are ready to execute, therefore distributing the workflows from the queue to idle VMs. Otherwise, if there are not VMs available, new VMs will be created for the workflows in the queue. Moreover, if the queue is empty but there are idle VMs, these should be shut down. However, the SPSS aims to prevent over provisioning. The algorithm always prefers to extend the runtime of existing VMs before allocating new VMs, so that a new VM is created only when the tasks cannot be finished within their deadlines.

Single cloud workflow deployment reduces the diversity of cloud computing resources. Even a transnational corporation, playing role of cloud provider cannot provider most types of clouds, especially at the level of SaaS. This thesis therefore attempts to provide solutions for federated clouds to obtain the great benefits of Multiple Clouds in terms of the diversity of different clouds.

2.5.1.2 Multiple Clouds

As discussed in section 2.2, the clouds involved in optimisation problems may have different characteristics. Therefore, users can minimise their workflow execution costs and meet other non-functional requirements by distributing work to different clouds (from different providers). Compared with a single cloud, multiple clouds have more types of computing resources, but also bring communication costs for transferring data from one cloud to another. Microsoft Azure [5], for example, charges users for downloading data from the cloud. This section reviews the studies that have deployed workflows over federated clouds to minimise the financial costs.

Kllapi et al. [97] modelled execution of the workflows (dataflows) and the monetary cost of running workflows over clouds. Furthermore, a set of state-of-the art scheduling algorithms were studied and adapted to optimise monetary cost and performance.

In another study [25], an algorithm was in introduced that schedules workflow applications in hybrid clouds composed of a public cloud and a private cloud. The public cloud has more high-performance computing resources but charges for their use. The private cloud is owned by the users and it was assume that this can be utilised without any cost. The algorithm first schedules the whole workflow to the private cloud. However, if this cannot meet the deadline, the public cloud is considered. The Ptah Clustering Heuristic (PCH) [24] is used to generate the execution order of each task.

Next, a cost model, including computation and communication costs, helps to set the priorities of each task. Finally, the tasks are rescheduled onto the public cloud to meet the deadline, with three rescheduling policies designed to minimise the cost.

In a similar study [82], the private cloud (the user's own machines) was also assumed to be a free computing resource, yet with limited computing power. A public cloud such as Amazon EC2 can meet the users' performance requirements, but the cost still needs to be minimised. A framework, called PANDA (PAreto Near optimal Deterministic Approximation) was designed to schedule a workflow across both public and private clouds with the best trade-off between performance; hence, Pareto-optimality. For example, if a schedule σ is dominated by schedule σ', where the values of both of cost and time metrics are worse than or equal to σ' and strictly worse in at least one criterion, therefore a schedule σ^* which is not dominated by any other schedule is Pareto-optimal.

Subsequent studies [57] and [56] introduced a Pareto-based algorithm to make the best deployment decision for the two trade-off dimensions of cost and performance. This algorithm firstly lists K trade-off solutions, and then applies a *crowding distance* [42] method to choose the solution with the highest crowding distances.

In a commercial multi-cloud environment, individual providers focuse on increasing their own revenue regardless of the utility for users and other providers. Therefore, the information provided by cloud providers may not be fully trustworthy. For example, one study [137] found that scalability perform once is not enhanced with cloud virtual machines by increasing the numbers of virtual cores. Fard et al [59] introduced a pricing model and truthful mechanism for scheduling workflows to different clouds, taking into consideration monetary cost and completion time. In order to solve the trade-off between cost and performance, a Pareto-optimal solution was adapted in the scheduling algorithm.

Relevant research focuses on how to solve trade-offs between monetary cost and performance in the execution of workflows over multiple clouds. However, realistic tools which apply optimisation algorithms to partition workflows over federated clouds are lacking. Thus, this thesis is the first study which develops a cloud broker taking into consideration security issues to partition workflows over different clouds.

2.5.2 Security-driven Workflow Deployment

Cloud resources can be offered by more than one provider, and even a single one provider may also want to diversify its offerings. For example, Microsoft Azure provides both private Clouds and public Clouds according to the security levels. The private cloud is considered to be more secure than the public cloud, as it is hosted on a share-nothing server architecture and access is by VPN (virtual private network). Therefore, the user can increase the security of their application by deploying the components of the workflow on more secure resources, e.g. by allocating sensitive components to the high security cloud with more expensive resource, and vice versa.

Mace et al. in [108] explored the current information security issue with the public cloud and provided general security solutions to choose what workflows or subsets of workflows can be executed in a public cloud, while meeting the security requirements.

The Bell-LaPadula security model [21] has applied to partition workflow based applications over a federated cloud to meet the security requirements [143] and [146]. Firstly, the components of the workflow and clouds were assigned with security levels. Next, the workflow was mapped to a set of clouds to meet the security requirements. Consequently, the cheapest mapping was explored using various optimisation algorithms. Goettelmann et al. [70] applied the same security model to map a workflow application over a federated Clouds, but they focused on minimising the communication cost to increase the reliability of their system.

A task priority mechanism has been introduced to rank the security priority of tasks [150]. Scheduling was driven by the priority ranking to achieve the highly secure deployment of an application in a heterogeneous distributed system. In addition, minimising the makespan of the workflow execution was also considered in the scheduling algorithm.

The SABA (Security-Aware and Budget-Aware workflow scheduling strategy)[154] provided a static workflow deployment solution over multiple clouds for optimising security, makespan and monetary cost. The optimisation in this work was based on a heuristic list which ranks the priority of each task of the workflow by using a normalisation function. Therefore, each task was scheduled onto the clouds based on its priority.

Another model has been proposed [106] which distributes a scientific workflow to clouds based on the security levels of intermediate data in the tasks (including inputs and outputs). Firstly, the security of the datacenter and data were measured, based on confidentiality, integrity and authentication. An ACO (ant colony optimisation)-based algorithm was used to dynamically map the workflow to clouds (data centres).

In another study, a decentralised workflow system was introduced [15] to solve performance bottlenecks, and a Chinese Wall Security Model was applied to the decentralised workflow environment in order to resolve the conflict-of-interest problems.

To increase the security of the cloud, most efforts focus on the cloud provider side by upgrading hardware, software of data centres, in order to provider more secure cloud. However, in this thesis a method is proposed which increases the security by deploying workflow over different clouds.

2.5.3 Reliability-driven Workflow Deployment

Node and network failure can have a detrimental impact on workflow performance. Therefore, a distributed workflow mapping algorithm which takes into reliability consideration is highly desirable to avoid or reduce the workflow execution failures.

One study [75] introduced the problem of mapping distributed workflows onto system where nodes and links are subject to probabilistic failures. Throughput and reliability was considered within the mapping problem, and a decentralised layer-oriented method was proposed to handle the high throughput of data flows while satisfying a pre-specified overall failure limit. An extension of the LDP(Layer-oriented Dynamic Programming) workflow mapping algorithm [74] was used to minimise the transferring time at global bottleneck links through a distributed manner under a reliability constraint.

Assayad et al. [13] attempted to explore the trade-off between reliability, power consumption and execution time. A heuristic scheduling algorithm was designed, which obtained the Pareto front of each object, and then approached two pre-defined constraints of two objects such as reliability and power consumption respectively and finally minimised execution time.

To reduce the effect of failures on an application executing on a failure-prone system, an algorithm has been proposed [52] which not only minimise the execution time but also considers the probability of the failure of the application when deploying the workflow over distributed environments. Both execution time and reliability rate are taken into account in one cost function, and then this function is optimised to find the best deployment solution.

Similarly, a BSA(Bi-Objective Scheduling Algorithm) [79] can quickly make a deployment decision for a workflow over heterogenous distributed computing systems, considering performance and reliability via a bi-objective compromise function. This function includes two steps. Firstly, the minimal solutions for both reliability and performance of each task are generated, and next, a weight parameter θ is used to privilege one of the objectives.

Dongarra et al. [53] introduced a pareto curve-based algorithm to optimise makespan and the reliability of workflow execution. The algorithm firstly generated the pareto-front of reliability for each task of a target workflow, without considering the order of the tasks. Next, based on the pareto-front, the makespan was refined to meet the time constraint M for each task, and record the computing resources. Finally, a solution was generated based on the recording, maximising the reliability and meeting the makespan constraint.

Another study [142] applied a method to measure the reliability rate dynamically, relating the time to the computing resource failure rate. In addition, a version of genetic algorithm was developed to optimise both the makespan and reliability of a workflow application.

Cao and Zhu [35] attempted to optimise the reliability and performance in a distributed workflow system, in which the measure of reliability includes the communication link and computing resources. A method was designed to combine iterative critical path search and Layer-based priority assigning techniques (CPL) to minimise the end-to-end delay (EED) by focusing on the optimal allocation of tasks on the critical path. Then a refinement method was applied to the tasks which were not on the critical path, so as to increase the reliability of the whole system.

In paper [105], the authors proposed an approach to the optimise the performance and reliability of scheduling workflow systems over clouds. Like BSA [79], the Pareto fronts were generated as two vectors as step one. However, they extended the Euclidean Distance [50] to define distance between the two vectors, in order to function two objectives. Therefore, the optimal solution can be found based on the function.

Fault Tolerance

Another way to increase the reliability of workflow deployment is by introducing methods to add fault tolerance. Ideally, this should avoid reproducing and redeploying the entire workflow when nodes or links fail during execution.

AHEFT [152] is a rescheduling algorithm based on [136]. It was used to handle the grid environment changes such as computing node failures, and supports the resumption of unfinished executions. A workflow was initially planned and executed over distributed computing resources, minimising the makespan. When the availability of resources is changed, a new schedule is generated based on the current computing resources and unfinished tasks, aiming to maximise the performance.

VgrADS [123] is a tool that enables the execution of a workflow over multiple computing resources such as clouds and Grids. Furthermore, fault tolerance techniques were added to increase the probability of success for each workflow task. The fault tolerance techniques analyse the initial mapping of the workflow, determine the mapping of the replicated tasks on the available slots and return the mapping to the planner. The replicated tasks are selected based on the predicted failure rate of each tasks of in the initial mapping.

QAFT [161] is a fault-tolerant scheduling algorithm that can tolerate permanent failures of one node in a heterogeneous cluster, during real-time tasks with QoS requirements. The fault-tolerant model extended the conventional primary-backup tolerant model[122] through scheduling two copies of a task on two different resources. This increases the success of the execution and can also verify correctness by comparing two versions of the results. Based on the fault-tolerant model, a reliability model can be used to quantitatively evaluate the system's level of fault tolerance.

AhmedIbrahim et al. in [127], introduced a robust mechanism (called AGS) which can detect resource failures and continue to offer functionality. A mobile agent model was used to decentralise the workflow execution and enhance the resource discovery and monitoring processes. Once a mobile agent has detected a failure in a computing resource, the agent and the tasks which were assigned to this failed resource should be migrated to another computing resource through ITT (in advance Task Transmission). Workflow execution can then be resumed.

Most relevant research uses the power method to measure the reliability of workflow deployment. However, this can only guarantee the reliability of the whole workflow, whereas a more advantageous entropy-based method is introduced and utilised in this thesis.

2.5.4 Performance-driven Workflow Deployment

To minimise the makespan of workflow execution is the most explored aim in workflow optimisation. For scientific workflows, execution performance is the most important issue and how to maximise performance in the presence of constraints has been thoroughly investigated. In HPC (high performance computing), researchers aim to scale execution by paralleling tasks and distributing them to multiple computing resources. However, with the pay-as-you-go model of cloud computing, users are considering how to maximise the utilisation of the resources they have been paid for. Different algorithms or techniques proposed to optimise the workflow execution performance are now investigated.

It has been shown [137] that cloud environments are very dynamic, especially in terms of fluctuations in the level of performance delivered. For example, it was found that the scalability of cloud virtual machine will not increase by increasing the number of virtual cores above a certain limits. In another study [32] an algorithm was proposed that uses the idle time of provisioned resources and surplus budget to replicate tasks so as to increase the likelihood of meeting deadlines. At the same time, the economic cost of execution can also be minimised by carefully planning the provision of VMs.

Although authors [102] was investigating a bag-of-tasks application which consists of a large number of independent tasks, the optimisation method developed can also be adapted to workflow applications. The cloud resources were assumed to be more reliable than grid resources for completing a job. The workflow was initially scheduled over grid resources, but if there is a delayed task and it impacts on meeting deadlines, this task will be rescheduled to a cloud.

A utility function-based method has also been used [99] to minimise the sum of the execution times for a set of workflows. Here the workflow execution was described as an optimisation problem for the utility function. Then, the authors attempted to uncover the space of possible assignments to maximise utility for limited computing resources.

ADOS (adaptive dual objective scheduling)[101] has been adapted to dynamically manage performance when changes resources occur. Firstly, a workflow was initially assigned to the resources by a method based on a branch-and-bound technique and a genetic operator, in order to minimise makespan and maximise resource usage. Next, a rescheduling event was triggered if a job finished later than expected and the delay completion resulted in an increase in the overall application completion time.

Zhang et all. [156] proposed a hybrid re-scheduling mechanism, and the rescheduling algorithm used is similar to AHEFT. However, the authors considered only the communication costs. This is because migrating some tasks to new resources may significantly increase these costs, thereby reducing the performance. Using a model that combines performance and communication costs, the potential penalty of excessive extra communication costs can be avoided.

Other authors [132] have introduced a taxonomy of grid workflow scheduling policies based on the amount of dynamic information used in the scheduling process. Seven dynamic workflow scheduling policies were applied to assign workflow tasks to multiple computing resources, including Round Robin [47], Single Clusters, AllCluster File-Aware, Coarsening, Cluster minimisation and HEFT. After analysing the performance of these scheduling policies, it was found that no single policy gave good performance across all the scenarios investigated. Another finding was that the limitations of the head-nodes of the grid clusters may lead to performance loss.

Another study [107] first systematically investigated a general exception handling framework for the automatic and cost-effective (including monetary cost and time overheads) handling of temporal violations in scientific workflow systems. The framework included several levels of predefined fine-grained temporal violations and a set of corresponding handling strategies were required. The higher the level of temporal violation, the more computing resources were required to recover the time deficit in order to guarantee a deadline was met.

A dynamic task rearrangement and rescheduling algorithm has been described which exploits the scheduling flexibility from precedence constraints among tasks [38]. This algorithm aims to deal with the delay arising from inaccuracy in performance information from previous execution logs in multiple workflows. If a delay in one of the workflow executions impacts on the execution performance of other workflows, the tasks in other workflow applications are rearranged. For dynamically rearranging tasks to other resources, a dynamic search tree-based algorithm was developed to rearrange the matches along an augmenting path to replace the edges that have not been processed.

Performance optimisation is one the most important branches of workflow optimisation research either in Grid computing, or cloud computing. In cloud computing well-defined pricing has brought a new and interesting challenge for researchers concerning how to solve the trade off between performance and monetary cost. However, this thesis does not tackle the issue of performance in optimisation problems. This will be investigated in the future.

2.5.5 Data Aware-driven Workflow Deployment

The data used in scientific workflows are often intensive and distributed geographically, published by scientists from a range of different institutions. Therefore, the movement of data becomes a challenge in situations where tasks need to process data from different data centres. Even if the application data can be moved to the task, moving data from one cloud to another can bring large performance and monetary costs, especially if the volume of data is large. There can also be implications in moving sensitive data to a cloud that is not sufficiently secured. The following, discussion considers using data awareness to improve workflow execution.

The High Performance Computing (HPC) community is currently trying to understand how to design or adapt applications to be able to exploit heterogeneous multicore architectures such as multi-GPU clusters. In order to optimise performance, superfluous data transfers should be avoided. Therefore, an asynchronously policy has been introduced [16] to minimise the impact of communications on computations. This policy decides whether or not it is worth moving data. The time required to process data is estimated, and then a pre-fetch mechanism is designed to transfer input data in the background while the previous tasks are still being executed.

In distributed cloud workflows, large amounts of data may need to be stored in different data centres. Yuan et al. [153] proposed a matrix-based k-means clustering strategy for data placement in scientific workflows. This attempted to schedule the tasks to the date centre that holds the most datasets in order to minimise the total data movement during the initial scheduling. During workflow execution, data was pre-allocated to data centres to reduce the waiting time. In addition, logs of the intermediate data produced by each task in a workflow were studied. Therefore, the cost of generating and storing intermediate data can be predicted. The authors attempted to optimise costs by deciding whether an intermediate dataset should be stored or deleted.

ADAS(adaptive data-aware scheduling) was developed to scale workflow execution over multi-clouds through increasing parallelisation opportunities [155]. Firstly, the dependencies between tasks were analysed and partitioned into different data-centres, increasing the parallelisation of execution. At runtime, dynamic data movement was overlapped with task execution so as to reduce the waiting time required for the data.

An efficient data and task co-scheduling strategy has also been proposed [49] that can load-balance input datasets and group the most inter-related datasets and tasks together. This was achieved by using the k-means clustering method that is based on analysing the catalogue of dependencies. Moreover, data staging is used to overlap task execution with data transmission in order to shorten the starting time of tasks.

Barker et al. [19] proposed a service-oriented architecture which facilitates the combination of an orchestration model and a choreography model to optimise data transport in workflow execution. This architecture avoids unnecessary data transfer, in order to solve the problem of the engine becoming a bottleneck when executing a workflow.

FlexIO [158] is a tool (middleware) with simple abstractions and diverse data movement methods for coupling simulation and analytics. Data-aware scheduling policies can be exploited and it can also cooperate with the workflow enactment engine to optimise workflow execution.

2.5.6 Multi-objective Resource Allocation for Workflow Deployment

Large-scale distributed computing resources such as clouds enable a variety of geographically dispersed resources to be interconnected and shared. In deciding how to do this, we may be faced with a set of potentially conflicting objectives such as monetary cost, performance, reliability, security and so on. Previous sections have discussed existing approaches that optimise one or two objectives, but what if there is the need to optimise an arbitrary number of objectives at the same time? The two most common approaches are: (1) to linearise the problem by assigning weights to the criteria and then optimising the weighted sum, (2) to optimise one criterion and keep the others constrained within predefined thresholds. The following discussion explores existing methods for optimising multiple objectives.

Szabo and Kroeger proposed a method to optimise makespan, communication overhead and monetary cost [133]. Two types of chromosome were used to represent the allocation and execution order of each task. Obviously there is a trade-off between execution performance, communication overhead and monetary cost. To handle this, the NSGA-II algorithm [43] was used to optimise both the allocation and ordering strategies simultaneously. Therefore, the evaluation method returns three-dimensional arrays referring to the communication overhead, makespan and financial cost. A linear equation was designed to include the three objects and then to minimise the equation.

Another study [131] presented a hybrid PSO algorithm to maximise workflow execution performance, and to minimise the monetary cost and energy consumption. The Pareto front of both execution performance and monetary cost were firstly generated, thereby combining them to be a candidate set. Secondly, the method finds a solution in the candidate set which minimises energy consumption.

A generic multi-object optimisation framework has been presented [58] which is supported by a list heuristic for scientific workflows in heterogeneous distributed computing infrastructures. The framework was demonstrated to address four objectives: makespan, economic cost, energy consumption and reliability. The makespan of each task of a workflow was sorted in ascending order according to its lowest level. In other words, the value of each task was the execution time from the given task until the last task through the longest path. Then, a resource was selected for each task by using the Euclidean distance corresponding to the longest weighted distance. Furthermore, a method (called $sched_w$) was used to improve the selection, which is a Pareto optimal.

Jrad et al.[89] proposed a cloud broker to a schedule larger scientific workflow over federated clouds to match the QoS and cost requirements. To optimise the objects, a quasi-linear utility function [98] was applied, including user preferences for QoS and monetary cost. Therefore, the monetary cost can be minimised based on the set of the preference values of QoS.

Security is also an important part of QoS, but none of the work mentions above considers security along with other QoS factors to optimise the deployment of workflow application over federated clouds. This thesis therefore takes reliability, security, and monetary cost into account while optimising deployment.

2.6 Conclusion

Deploying workflow applications over clouds are usually aims to meet functional requirements such as specific execution environments. In real-world scenarios, non-functional requirements such as performance, security, reliability and monetary cost also significantly influence workflow execution. Most current work in cloud computing focuses on improving the infrastructure to meet users' functional requirements, but very few researchers have considered partitioning workflow applications over different type of clouds in order to meet non-functional requirements.

Although some studies have investigated how to make the best deployment decisions to solve trade-off problem between performance and the monetary cost, few consider reliability and security. In sections 2.5.2 and 2.5.3, research has been discussed which tried to improve the security and reliability of workflow execution by scheduling a workflow over different computing resources. However, most of these studies focus on single cloud or Grid computing.

Moreover, most of the work required is designed specifically for a typical type of workflow, whereas this thesis explores a generic framework for partitioning a scientific workflow over federated clouds to meet non-functional requirements.

3

Dynamic Exception Handling for Partitioned Workflows running on Federated Clouds

Contents

Summary

This chapter introduces a new automatic method for dynamically partitioning applications across a set of clouds in an environment in which clouds can fail during workflow execution. The method deals with exceptions that occur when clouds fail, and selects the best way to repartition the workflow whilst still meeting security requirements. This avoids the need for developers to have to code ad-hoc solutions to address cloud failure, or simply accepting that an application will fail when a cloud fails.

3.1 Introduction

Cloud computing can provide cheap scalable storage and processing on demand in a wide range of domains. It is becoming increasingly important in both business and scientific research. Furthermore, as discussed in chapter 2, cloud federation provides the possibility to meet security and dependability requirements, with work allocated to clouds based on security levels and data replicated across clouds to meet availability needs.

Cloud security is one of the key issues affecting decisions on which applications should be deployed in the cloud. The influential Berkeley report [12] placed data confidentiality and auditability high on the list of concerns that may deter organisations from moving to or creating applications in the cloud. Using a federated clouds is one way to address security concerns. For example, some organisations combine their internal clouds (private cloud) with an external, public cloud such as Amazon EC2, as they judge that the private cloud is more secure. Sensitive applications are deployed on the private cloud, while others are deployed on a public cloud so that its scalability can be exploited.

Dependability is also an important issue to be considered in cloud-based applications. Individual nodes may fail during an execution, and in some extreme cases whole clouds have been unreachable for several hours. These frequency of failures may cause huge problems for an organisation, thus improving the dependability of cloud applications has become a pressing issue. This chapter does not consider improving the dependability at the infrastructure level; instead it is addressed this by introducing a method that handles failure for applications structured as workflows running over federated clouds so as increase dependability.

A few studies have considered deploying workflows over federated clouds to meet security requirements. One model based on the Security Alliance's Cloud Controls Matrix and Consensus Assessments Initiative Questionnaire has been described [23] to find a quality cloud which will meet users' requirements. Furthermore, as mentioned in chapter 2, two studies [150] and [154] have considered both monetary cost and security requirements. However, this chapter not only considers how to optimise the cost of meeting security requirements, but also considers how to deal with exceptions when the workflow applications are running on a set of clouds.

The remainder of this chapter is structured as follows. In section 3.2, a method is introduced to partition a workflow application across a federated cloud to meet security requirements. The following section then describes how to apply the DMI model is applied to integrate with the workflow over a federated clouds. In section 3.4, an example is given, showing the exceptions that may happen in a simple workflow running over a federated cloud, and then is shown that the proposed method gives the best solution is one exists for each exception. Before conclusions are draw, an example of how the Exception Handler works for a real cloud platform is given in section 3.5.

3.2 Security Model

This section extends our previous work [143],based on the Bell-LaPadula [21] Multi-Level Security model [36], to demonstrate how it is adapted to our system. With this adaptation, the security levels of the clouds, data and services are incorporated to achieve a secure deployment for the workflow over a federated cloud.

A workflow-based application consists of a set of services and data. It is modelled as a Directed Acyclic Graph (DAG), $G = (S, E)$, where S is the set of services, and E is a set of dependencies between those services. Services are represented by the graph vertices and the edges represent the dependencies between those services. Although a workflow-based application can have several different types of dependency relationships, the present study considers the data dependency (this is the most common dependency relationship in scientific workflow applications). In this type of dependency, a data item is generated from a source service and consumed by a destination service. For example, $e_{i,j}$ represents data dependency between service s_i and service s_j. To represent data dependencies we use a distance matrix $D = [d_{i,j}]$ of size $|S| \times |S|$ where a positive value of $d_{i,j}$ indicates a dependency from s_i to s_j as well as the size of transmitted data. $(s_i \rightarrow s_j)$ o denotes either D or S, and $\mathcal{O}$ is a set of o, noting $o \in \mathcal{O}$. Furthermore, C represents a set of clouds which are available for deployment.

3.2.1 Representing Security Requirements

In our security model, each service S has two security levels: "Clearance" and "Location". "Clearance" represents the services' highest security level, and "Location" is the required operation security level of the service in a specific application. The data D and cloud C only have "Location". $l(o)$ and $c(o)$ represent the security of location of o and the clearance of o respectively.

W represents the security constants, including three rules:

- ***NWD*** "no-write-down": denotes that a service cannot write data which has a lower security level (required security level) than its own.

$$NWD(d_{i,j}, s_j) = \begin{cases} true, & c(s_j) \geq l(d_{i,j}) \\ false, & \text{Otherwise} \end{cases}$$

- ***NRU*** "no-read-up": means a service cannot read data if the data's location security is higher than the service's clearance security.

$$NRU(s_i, d_{i,j}) = \begin{cases} true, & l(d_{i,j}) \geq l(s_i) \\ false, & \text{Otherwise} \end{cases}$$

- ***SIC*** "security in cloud computing" (SIC): defines the location security level of a cloud that should be greater than or equal to the location security level of any service or data that are hosted on this cloud.

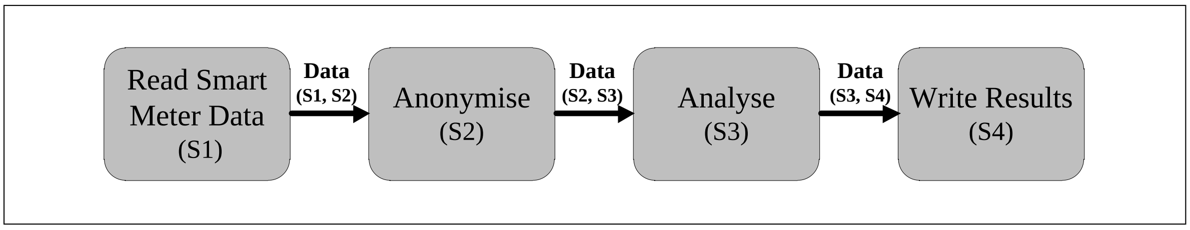


Figure 3.1: A Smart Meter Analysis Workflow

$$SIC(d^c_{i,j}, s^c_j) = \begin{cases} true, & l(c) \geq l(s_i) \\ & \text{and} \\ & l(c) \geq l(d_{i,j}) \\ false, & \text{Otherwise} \end{cases}$$

Where $d^c_{i,j}$ and s^c_j represent both data $d_{i,j}$ and service s_i to be deployed on cloud c.

To show how the security model can be applied to a workflow, we have applied the above security rules to the workflow shown in Figure 3.1. The workflow is a four services pipeline workflow for analysing Smart Meter Data.

By apply NWD as shown in figure 3.2, the Clearance of $s2$ ($c(s2)$) must be greater than or equal to the Location of $data(s1, s2)$ ($l(data(s1, s2))$). Moreover, NRU requires that the Location of $s1$ ($l(s1)$) must less than or equal to $l(data(s1, s2))$. Regarding to SIC, the Location of cloud c ($l(c)$) must be greater than or equal to the location the of data or service that have been deployed, $l(s1)$ and $l(data(s1, s2))$.

Figure 3.2 shows the full picture of applying these security rules to the smart meter analysis workflow, where arrows represent $\geq$ relationships and c here can be different valid clouds for hosting the services and data.

3.2.2 Valid Deployment Options

The security levels are assigned to each o of the given workflow, as shown in Table 3.1. For simplicity, only two levels are used: 0 (representing low security) and 1 (representing high security). Moreover, two available clouds $C1$ and $C2$, representing public cloud and private cloud, are considered to host the workflow. The private cloud has a higher security level than the public cloud, where $l(C1) = 0$ and $l(C2) = 1$.

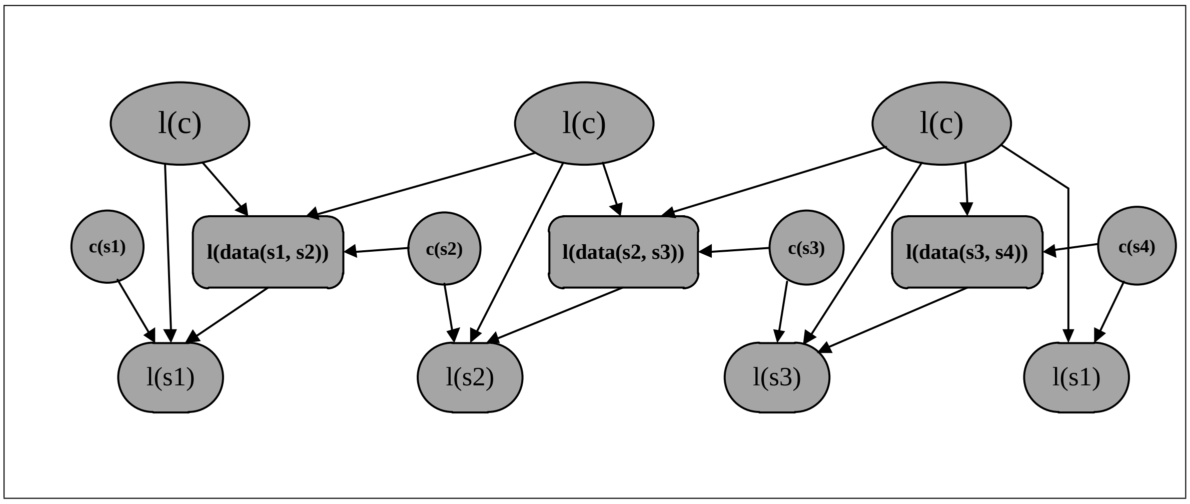


Figure 3.2: The security lattice for Smart Meter Analysis Workflow

Object	Clearance	Location
S_1	1	1
S_2	1	0
S_3	0	0
S_4	1	0
$Data_{s1,s2}$		1
$Data_{s2,s3}$		0
$Data_{s3,s4}$		0

Table 3.1: Workflow Security Levels

$S1$	$S2$	$S3$	$S4$
$C2$	$C2$	$C1$	$C1$
$C2$	$C2$	$C2$	$C1$
$C2$	$C2$	$C1$	$C2$
$C2$	$C2$	$C2$	$C2$

Table 3.2: Valid Options

$C2$	$C2$	$C2$	$C2$	$C1$	$C1$	$C1$
$S1$	$Data_{s_1,s_2}$	$S2$	$Data_{s_2,s_3}$	$S3$	$Data_{s_3,s_4}$	$S4$

Table 3.3: One Option

This example includes two clouds and four services, and the total number of deployment options is 2^4. However, only four options meet the security requirements, as shown in Table 3.2.

To be precise, in the first option, for example, $S1$ and $S2$ are deployed on $C2$, but $S3$ and $S4$ are assigned to $C1$, as shown in Table 3.3. True of applying NWD to this option, we can find $c(S2) == l(Data_{s_1,s_2})$, $c(S3) == l(Data_{s_2,s_3})$ and $c(S4) > l(Data_{s_3,s_4})$. Regarding to NRU, we also can get true based on $l(Data_{s_1,s_2}) == l(S1)$,$l(Data_{s_2,s_3}) == l(S1)$ and $l(Data_{s_1,s_2}) == l(S1)$. The third rule SIC gives the true as well by $l(C2)$ is great or equals to $l(S1)$, $l(Data_{s1,s2})$, $l(S2)$ and $l(Data_{s2,s3})$. Moreover, we assume that put $Data_{s2,s3}$ either in $C2$ or $C1$ is the same option, if it does not violate the security rules. In the rest of the thesis, we assign the data to the cloud as the same as its source service.

3.3 Dependable Multiparty Interaction Framework

Multiparty interaction is a mechanism that encloses multiple parties executing a set of activities together. Zorzo and Stroud in [162] proposed a *dependable multiparty interaction* (DMI) in terms of using multiparty interaction for handling concurrent exceptions and the synchronisation of participants (all participants have to wait until the whole interaction finishes).

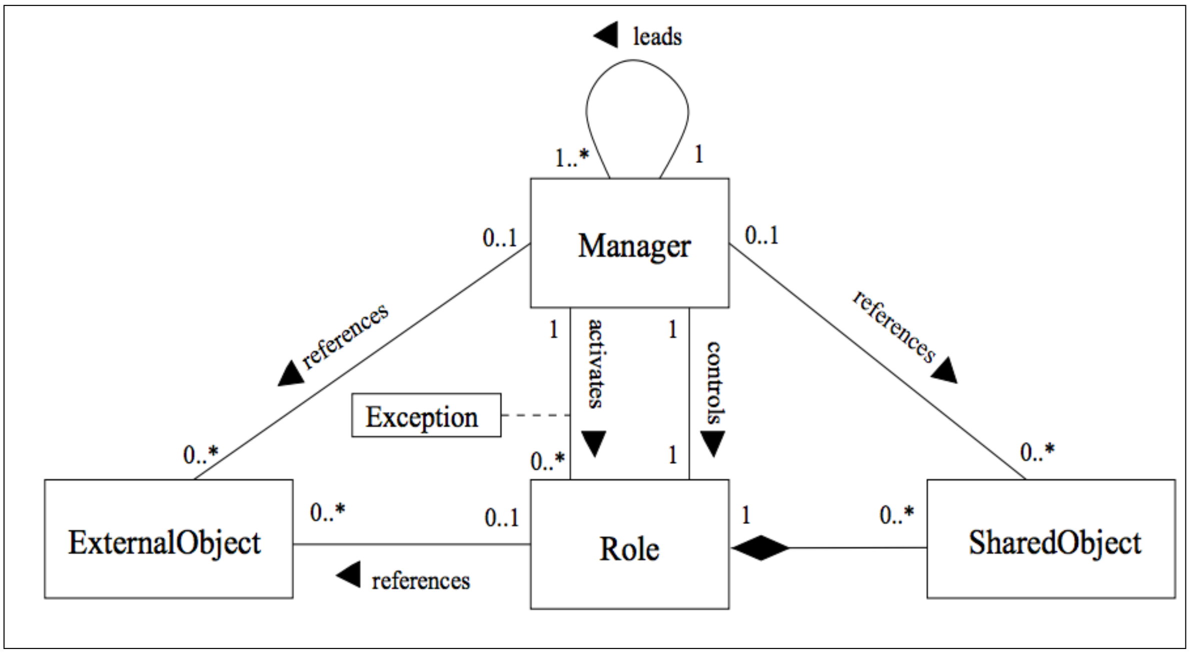


Figure 3.3: The Architecture of DMI

3.3.1 DMI Framework

The system is composed of a *Manager*, *Role*, *ExternalObject* and *SharedObject* as shown in Figure 3.3. The *Manager* is responsible for monitoring of the components of the interaction, managing the synchronisation of participants, testing the pre-and post-condition for the interaction, and deciding upon which exception is to be handled by the participants in the interaction. *Role* hosts the set of operations for the participants of the interaction. *Sharedobjects* are used for cooperation between the participants. The state of the system in and out of the interaction is carried in *Externalobjects*. Therefore, there is no failure only *Manager* is activated for the basic interaction, and others are used for dealing with exceptions that may be raised during the execution of interactions.

3.3.2 Integrating DMI with the Workflow Partitions

The DMI is a distributed object-oriented programming framework. However, in the present study is extended and adapted to integrate with distributed workflow partitions.

In the adaptation, each partition is represented as a *Role*, and it interacts with

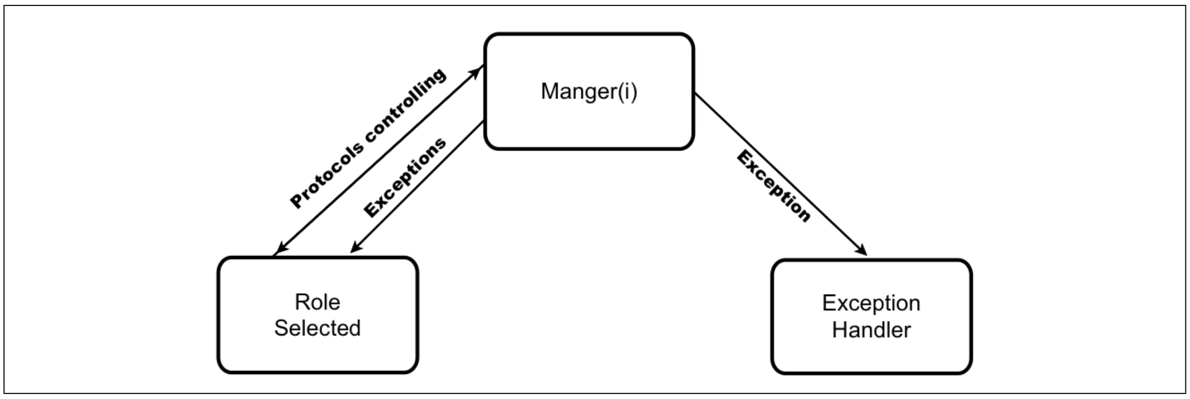


Figure 3.4: DMI Exception Handing Architecture

a corresponding *Manager*. Furthermore, all *Manager* communicate with a single *ShareObject* (called Exception Handler), providing the solutions to handle the exceptions.

Figure 3.4 shows the architecture of the DMI exception handler. *Role* represents the executing partition, and the state information of the *Role* is passed to the corresponding $Manger(i)$ through the dedicated channel to check that the partition is running correctly. If an exception is detected by the $Manager(i)$, it will send an exception notification to *Role*. *Role* is stopped and the system state is rolled back to the state before the *Role* was executed.

3.3.3 Extension to Clouds

This section describes how the DMI model is applied to a workflow enactment, and how it can be extended to a federated cloud. In the method used in a previous study [143], a workflow is split into several partitions and deployed onto a federated cloud. To integrate this approach with the DMI model, the DMI must monitor the cloud states in order to ensure that each partition runs reliably. There are 3 policies that have to be added into the *Manager*:

- Checking the state of the cloud hosting the partition that is the destination of a data transfer between two partitions, before transmitting data to it.
- Checking the state of the cloud on which a partition is to be run, before the partition starts running.

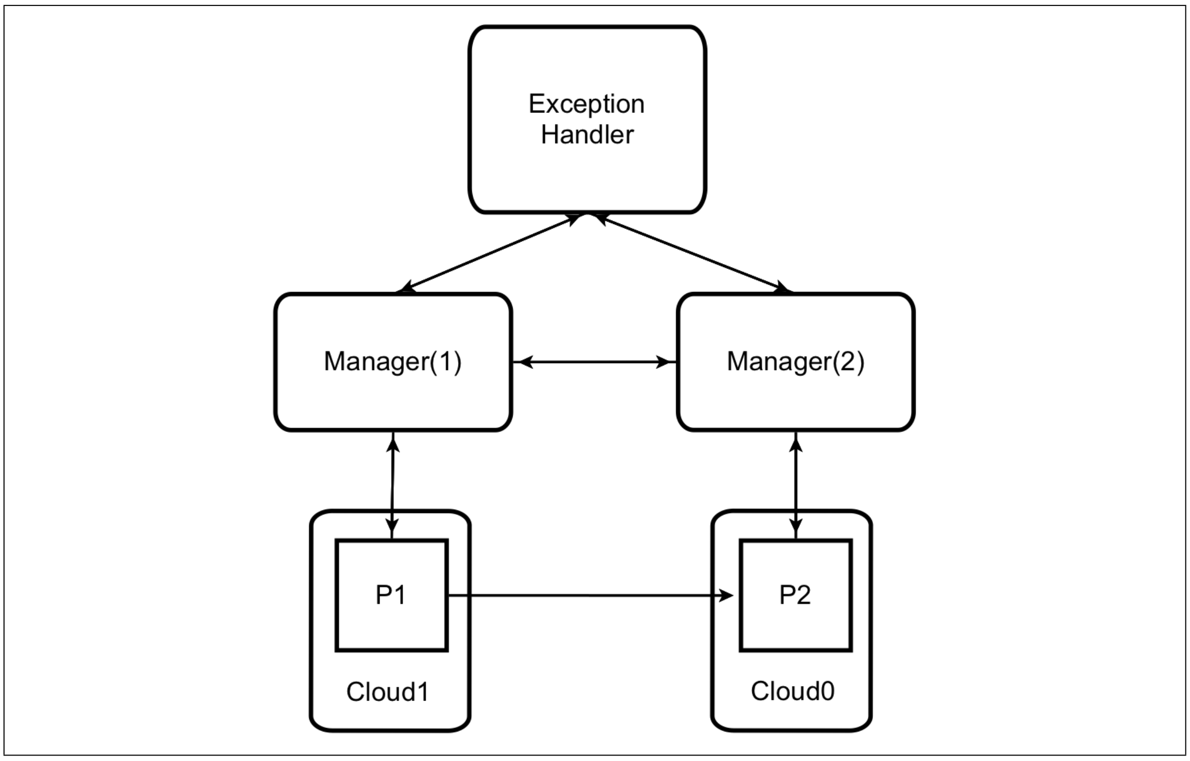


Figure 3.5: DMI applied to partitioning

- Checking the state of the clouds hosting the currently running partitions.

Figure 3.5 shows the DMI applied to a workflow partitioned over a federated cloud. A workflow is split into two partitions, P1 and P2, which are deployed onto two clouds: Cloud 0 and Cloud 1. Before P1 executes, the manager (1) must check the pre-condition that Cloud 1 is available for executing P1 and all the required data has already been loaded onto the cloud. If those requirements are met, Cloud 1 will execute P1. When P1 finishes, a notification is sent to manager (1), which then invokes manager (2) to tell it to execute P2. Manager (2) must then check the state of Cloud 0 to see whether or not it is ready to process P2. If every requirement is met, manager (2) will tell manager (1) to send a notification to P1. If manager (1) does not receive a confirmation notification, the exception handler will be invoked. When P1 receives the notification, the data generated by P1 will immediately be sent to P2. P2 is then executed after manager (2) checks all requirements for executing P2. The requirements include the availability of the data generated by P1.

3.4 Exception Handler

Exceptions are classified into two types: Type #1 partition fails and Type #2 cannot transfer data from one partition to another. An exception of Type #1 is caused by:

- A cloud failing while the workflow partition is running

An exception of Type #2 is caused by:

- Data being lost in transmission between clouds.
- A cloud being unable to consume the input from another cloud.

3.4.1 Handling Exceptions

After application of the security method, a set of partitioning options is generated, and then the set of partitions contained in those options are collected in a Valid Partition List. Next, the dependency relationships from each option are added to the partitions in the Valid Partitioning List, discard the duplicated paths and calculate the cost of each partition by applying a cost mode[143]. As a result, a Partition Tree is generated, which includes dependency relationships and the cost of each partition. A running example will be used to demonstrate how to create the Partition List and Partition Tree in next section.

When exceptions occur, the Exception Handler is invoked to deal with them. There are two possible solutions: sleeping for an interval of time before re-invoking the Exception Manager to try to re-execute the partition or finding a new Manager to invoke an alternative partition. These are dealt with by the Exception Handler as follows:

- If the cloud service fails when a partition is running in a cloud, the manager which corresponds to the partition will detect the failure when carrying out the post-condition test. Next, an exception notification is sent to the Exception Handler to invoke it to deal with the exception. In this situation, finding a new cloud to execute this partition is the only solution. Therefore *option1* is invoked and the

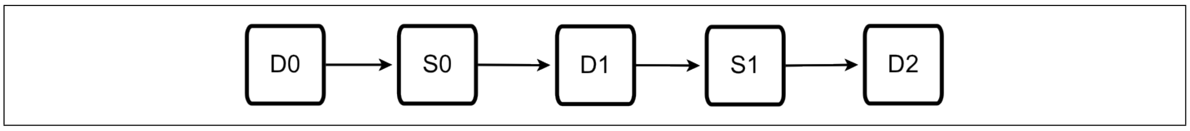


Figure 3.6: A Sample Pipeline Workflow

"Find a New Manager" function is activated. If one or more clouds which meet the requirements of by the partition which has raised the exception are available, the Exception Handler selects the cheapest one and invokes the *Manager* which corresponds to the selected partition. If not, an extension of *option1* is invoked, and the "Find a New Manager" function tracks back to the previously executed partitions, and *option1* is used to find a suitable partition. If the function cannot find a replacement partition, this workflow has to be abandoned.

- Data may be lost in transferring from one cloud to another because of network problems, or if the leader *Manager* (corresponding to the running partition) finds that the next partition is not able to accept the data sent to it. When an Exception Handler receives this exception notification and analyses it in "Exception Analysis", then *option2*: Thread Sleep is activated to wait for a small period of time before retrying. If the Retry fails after a set number of times then, *option1* will be invoked.

3.4.2 Example of Exception Handling

The exception handling algorithm is illustrated using the running example from the previous study [143], showing how it can be extended to deal with cloud failure. Figure 3.6 shows the workflow ($\rightarrow$ denotes a data dependency, D and S represent datum and service respectively, while the subscript is a unique identifier for each block).

The example of a pipeline workflow is partitioned over a federated cloud that is composed of two clouds: a public cloud and a private cloud. The previously described method [143] uses a static partitioning algorithm that generates six valid partitioning options (Figure 3.7).

Once the valid options for deploying the services and data to the clouds have been determined, it is necessary to select an option for execution, and a cost model is used

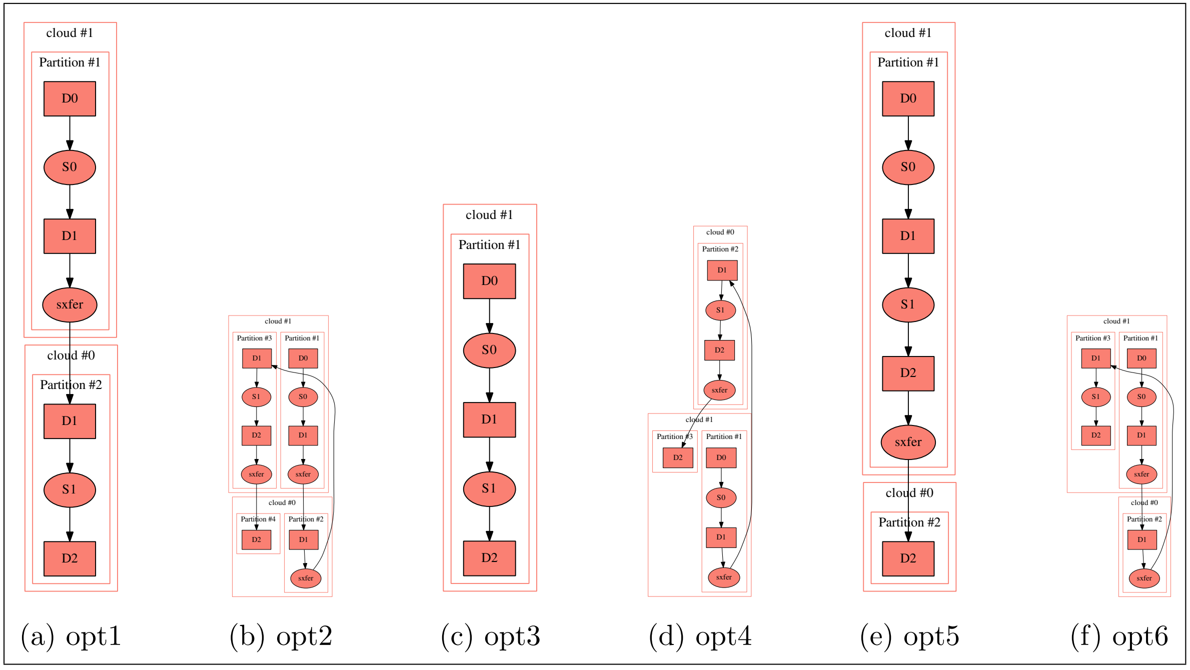


Figure 3.7: The Six Valid Cloud Mappings

Cloud	Storage (GB / Month)	Transfer In (/GB)	Transfer Out (/GB)	CPU (/s)
c_0	10	10	10	10
c_1	20	5	5	20

Table 3.4: Cloud Costs: Example

to find the best option. The factors that affect costs are the price of the Storage, Transfer and CPU. Table 3.4 gives an example of costs for the two clouds, while Table 3.5 gives examples of costs for the workflow blocks

In the prior study [143], the cost model was used to choose the best option, which was then executed. The model is summarised as follows:

Block	Size (GB)	Longevity (months)	CPU (s)
D_0	10	12	
S_0			60
D_1	5	0	
S_1			100
D_2	20	12	

Table 3.5: Block Info

$$
\begin{aligned}
cost &= \sum_{d_{i,j} \in D} store(d_{i,j}^{c_i}) * size(d_{i,j}) * longevity(d_{i,j}) + \\
&= \sum_{s_j \in S} exec(c_i) * time(s_j) + \\
&= \sum_{d_{i,j} \in D} (c_j^{out} + c_i^{in}) * size(d_{i,j})
\end{aligned}
$$

where $store(d_{i,j}^{c_i})$ is the cost of storing a unit data in c_i. The $size()$ and $longevity()$ represent the data size and storage time respectively. Moreover, $exec(c_i)$ is the unit price of c_i, which multiplied by with the execution time of s_j. c_j^{out} and c_i^{in} give the cost of transferring a unit data out c_j and that in c_i.

However, no consideration was given as to the action to take if there was a cloud failure during execution which prevented its completion. The key contribution of this chapter is to propose a method for extending the previous model to allow the exception handler to decide on the best way to deal with exceptions that occur during workflow execution. The aim is to choose the cheapest available option at the point when the exception is raised.

Firstly, the options generated by the partitioning algorithm (see Figure 3.7) are combined into a Partition Tree. This is achieved as follows:

1. A Partition Set is created, consisting of all partitions found in the partitioning options (Figure 3.8 shows all the partitions for the running example)

2. Create a directed graph from this set of partitions by adding directed edges between partitions wherever a data dependency between those partitions exists in the partitioning options

3. Prune the graph to replace multiple edges between two partitions with a single edge

4. Label the graph with the cost of executing each partition (including any costs of transferring any input data into that partition)

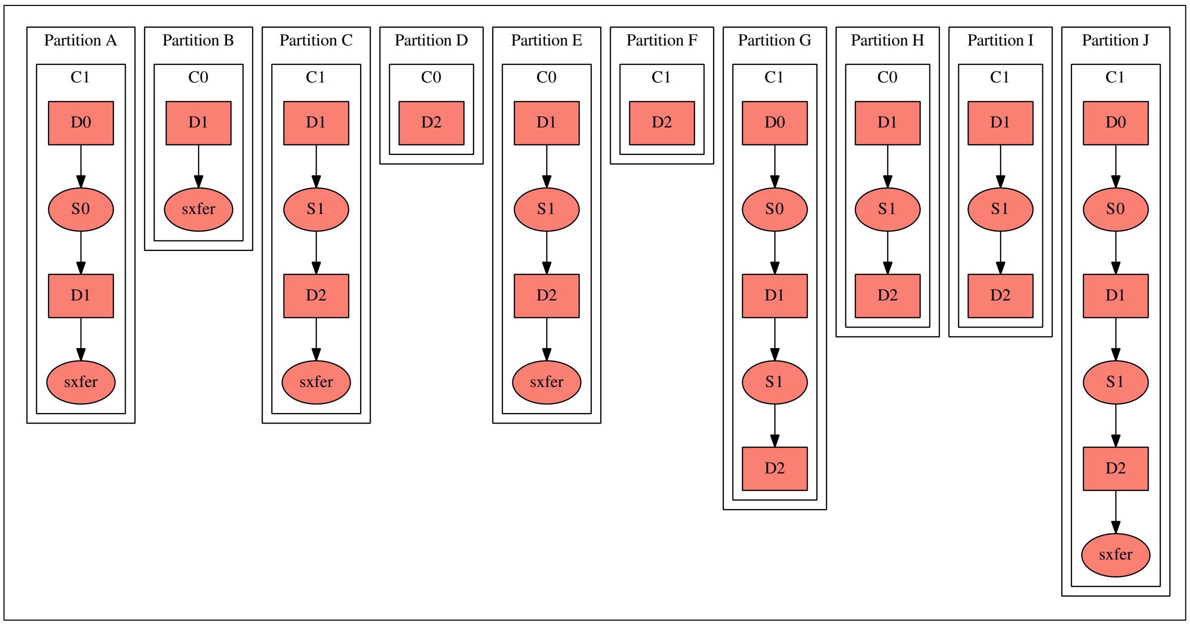


Figure 3.8: Valid Partitioning List

Figure 3.9 shows the Partition Tree for the running example. The yellow nodes indicate that the partitions are deployed on Cloud 1, while the salmon colour represents those deployed on Cloud 0. By convention, the cost of the initial partitions A, G and J, do not include the cost of transferring data in to the clouds on which they are deployed (we assume that the data is already there); likewise, no outgoing data cost is calculated for the terminal partitions B, E, G and J. The costs are shown as labels on the incoming edges to the partitions.

Next, the Partition Graph can be used to work out the best option for executing the workflow. Initially, from Figure 3.9, it is easy to see that the "best path" is $A \rightarrow H$.(the weight in each path represents the lowest cost from its start node to the leaf nodes.) In this case, the workflow application consists of two partitions, which are deployed onto two clouds, Cloud 0 and Cloud 1, as shown in Option1 of Figure 3.7a.

When an exception occurs in a partition currently being executed, the program identifies the type of exception first, and then decides to how to handle it. If the exception must be solved by replacing the currently running partition, an alternative partition is found using the Partition Tree as follows

For a Type1 (failed communication) exception all the other child partitions are considered, and tried in order of the cost of the path through that child to a terminal node

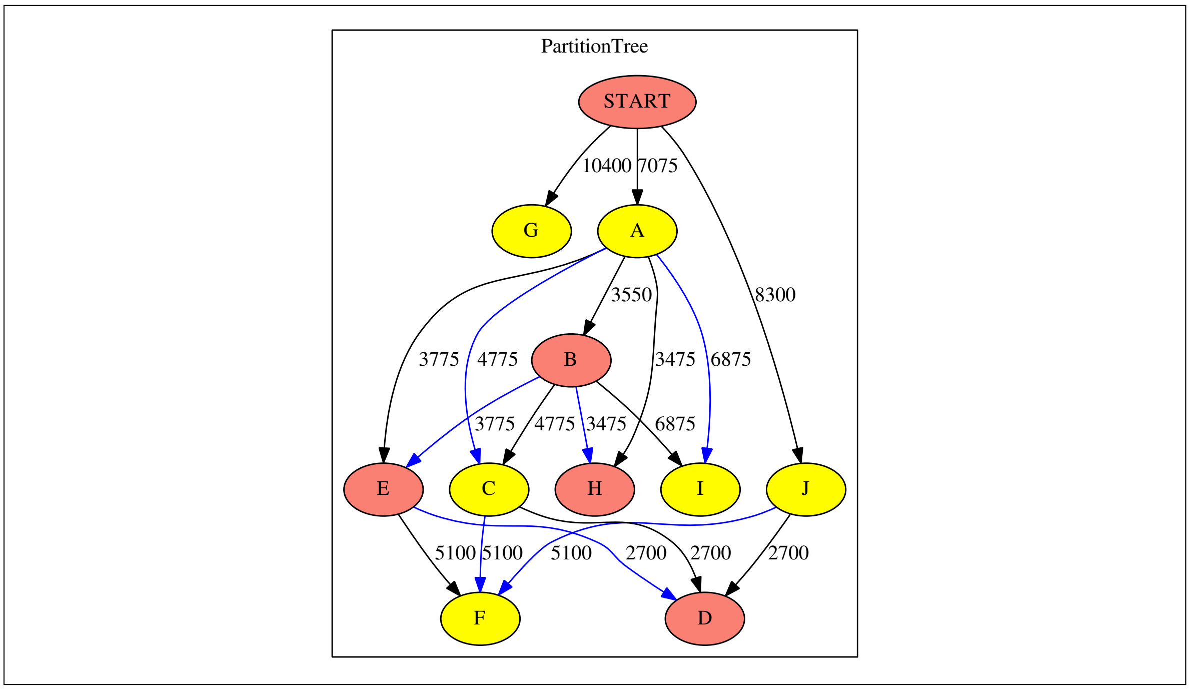


Figure 3.9: Partition Tree

(cheapest first). If this node does not have child nodes or none are able to accept the data then the same approach is taken, starting with the parent node of the partition that raised the exception. If this is unsuccessful, then the process is repeated for the grandparent partition, and so on, moving up the tree, checking all alternatives, until the data is safely delivered to another partition, or the "Start" node is reached, in which case there is no possibility of executing the workflow.

To deal with a Type 2 exception (partition execution fails), attempts are first made to re-execute the partition that generated the exception, after a time delay between each attempt. If this fails then, starting from the parent of the failed partition, all the other child partitions are considered, and tried in order of the cost of the path through that child to a terminal node (cheapest first). If this is unsuccessful, then the process is repeated for the grandparent partition, and so on, moving up the tree, checking all alternatives, until the data is safely delivered to another partition, or the "Start" node is reached in which case there is no possibility of executing the workflow.

For the running example, path $A \rightarrow H$ is the best initial option.

The actions that the exception handling system will take in some example cases are now considered.

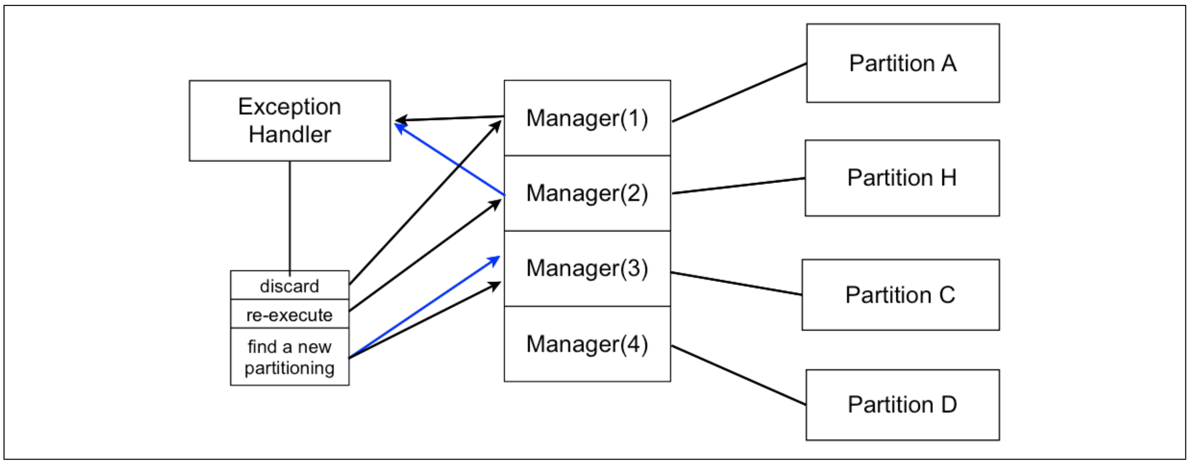


Figure 3.10: Exception Handler Works on Option1

1. The first case is a failure of Cloud 1 while partition A is running. When Manager(1) detects this exception, it abandons the workflow application as there is no a partition that can replace partition A.

2. If the exception is caused by losing $D1$ when it is transferred from Cloud 1 to Cloud 0 then, as Figure 3.10 shows, after a limited number of retries Manager(3), the controller of partition C, is invoked to replace partition H.

3. When partition A's execution has finished, the system will try to execute partition H. However, if Cloud 0 is not available, running H is not an option. Keeping the workflow running on Cloud 1 would be the only way to solve this problem. Therefore partition I or C from Option2 and Option6 respectively are both potential replacement partitions.

4. If the exception is caused by Cloud 0 failing while partition H is running, then partition C will be the best alternative partition because the path $C \rightarrow D$ is the cheapest option which is still executable. Cloud 0 may be available when partition C is finished; and even if it is not, partition D can also be replaced by partition F.

3.5 Evaluation

To allow the system to be used for real cloud applications, and to support a thorough evaluation, we built a tool that exploits e-Science Central[81]. e-Science Central is a

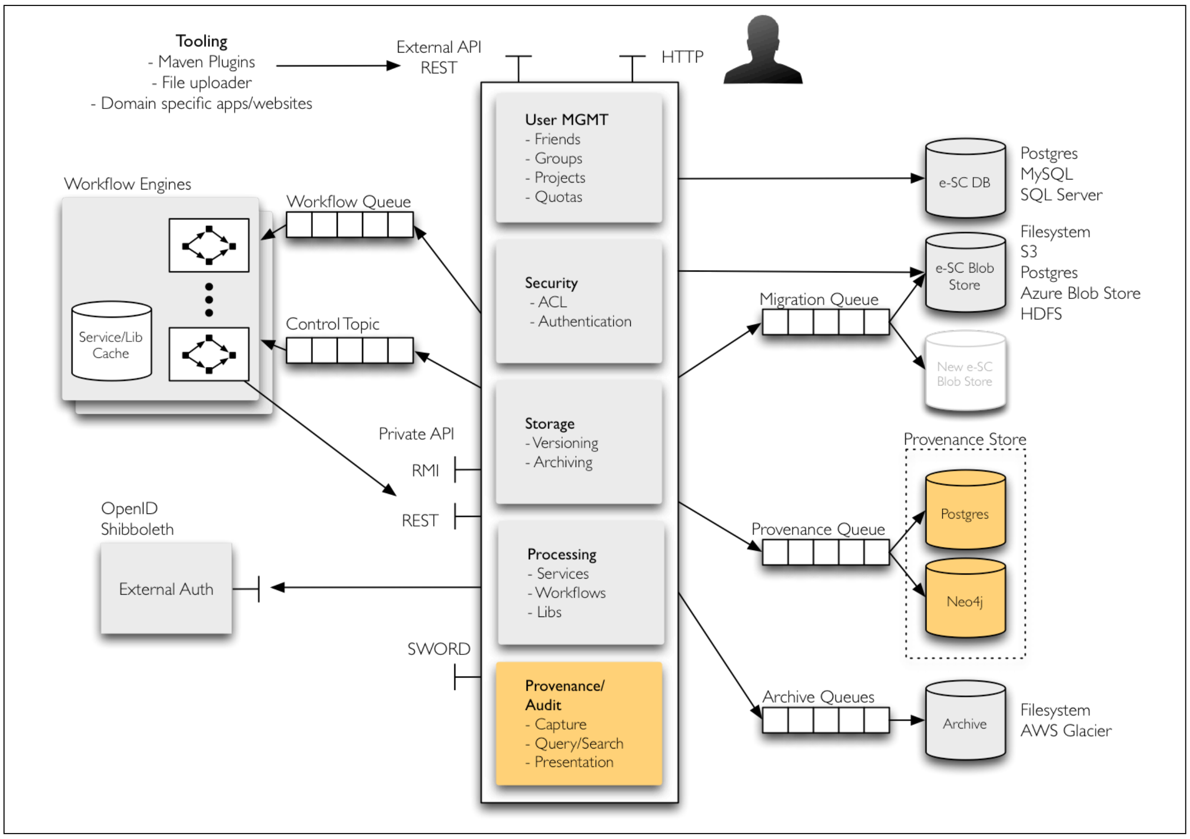


Figure 3.11: Architecture of the e-Science Central Cloud Platform

high-level cloud platform which can run on a range of clouds including public clouds (Amazon, Windows Azure and OpenShift) but also private clouds. Figure 3.11 shows the Architecture of e-Science Central. We built a tool that uses the method described above to split the original workflow into several partitions, and then deploy them across a set of clouds on which e-Science Central has been installed. It then handles any exceptions that arise.

e-Science Central provides a range of APIs to allow users to drive the system by using programmes, such as creating a workflow, executing a workflow, or uploading or downloading data etc. It is therefore possible to create a third-party tool to manage a number of e-Science Central instances which are installed in a set of clouds, in order to deploy and execute partitioned workflows over federated clouds. Figure 3.12 shows the architecture of the tool, which indicates how the $DeploymentManager$ communicates with a range of e-Science Central instances. $DeploymentManager$ generates several valid options by applying the method described above to a workflow. Furthermore, e-Science Central has a sophisticated provenance capture and analysis system[149],

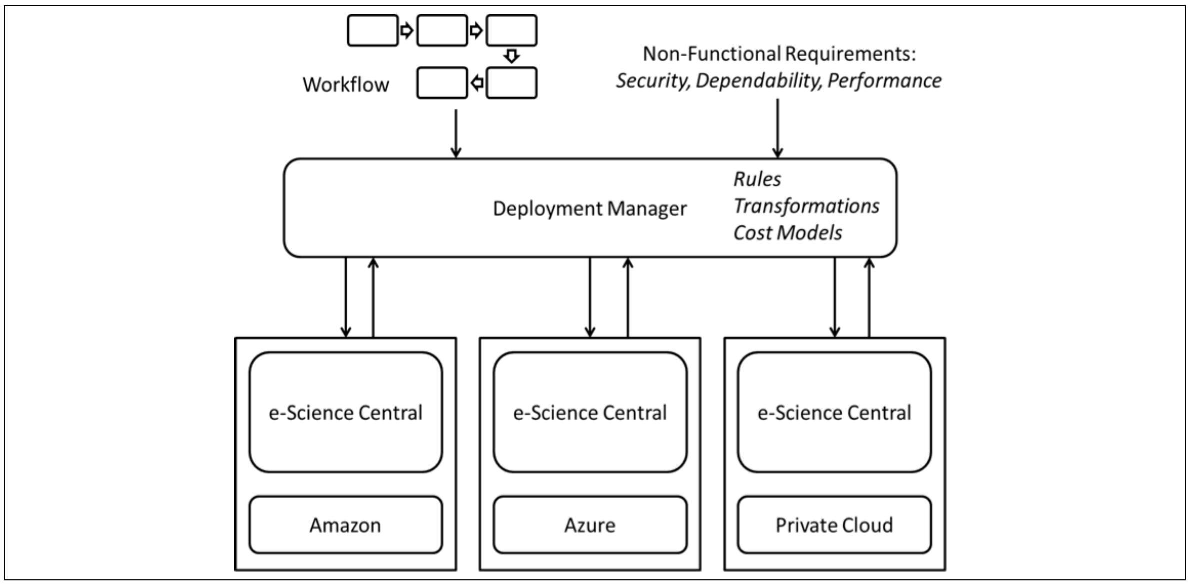


Figure 3.12: The Architecture of the General Cloud Workflow Partitioning Tool

and provides APIs to access this data which includes a record of service performance and data sizes. Therefore, the cost model can be calibrated from information derived from the past history of workflow execution.

The *DeploymentManager* could also monitor the availability of the clouds and the status of the running workflow partitions. For example, *DeploymentManager* could send regular notifications to e-Science Central deployments to get the running information for each partition of the workflow, and to find available clouds. This allows the *DeploymentManager* to select dynamically between possible workflow partitions according to the availability of the clouds, and to automatically handle exceptions generated by the running partitions.

3.5.1 Implementation

The above section has described the architecture of the tool and how it integrates with e-Science Central instances to distribute workflows over a set of clouds. This section now describes how it is implemented.

The first issue to be considered is how to implement the *DeploymentManger* which is responsible for analysing and partitioning the target workflow and distributing the partitions. Therefore, in this implementation the *DeploymentManger* includes three components as follows:

1. *Downloader* is a component in which the e-Science Central APIs are used to download the target workflow including services, data size and security information as in JSON format.

2. *Planner* is used to analyse the downloaded workflow, and then generate the deployment solution by partitioning the workflow over available clouds based on the workflow and cloud information.

3. *Dispatcher* distributes the partitions to each e-Science Central by using APIs to create services and upload the data. Furthermore, *Dispatcher* needs to initial a *manager* in the target cloud for each assigned partition.

4. *ExceptionHandler* is triggered when an exception occurs in one of the partitions, and it then re-partitions the unfinished services over available clouds while meeting the security requirements and minimising the cost.

From section 3.3.3, each partition corresponds to a *manager*, which it is composed of *monitor*, *messenger* and *transfer*. The *monitor* can execute the partition when the services are created and input data are ready, and monitor the execution status of the partition. In addition, the *messenger* sends the exception notifications to the Exception Handler if there is a failure in the corresponding partition. Otherwise, the *messenger* sends the normal notification to *DeployManger* to get running status as heart beat way. Moreover, if the cloud fails, the *ExceptionHandler* will not get notifications from the *manager* which are deployed on this cloud. The *transfer* is designed to transfer the output data of the corresponding partition to successor partitions through e-Science Central APIs.

3.5.2 Testing the Exception Handler

This section describes how the Exception Handler works with e-Science Central. To test the method described in this study, two virtual machines are created, each of which runs an instance of e-Science Central on which partitions can be deployed and executed. One plays the role of the private cloud, the other acts as a public cloud. For example, the partitioned workflow Option1 shown in Figure 3.7a is distributed onto two instances

Partitioning	Execution Time (s)
A	20
H	30
C	25
D	10

Table 3.6: Partitioning Info

of e-Science Central: eSC1 and eSC2. When *Partition*#1 has completed execution in eSC1, if eSC2 is not available to execute *Partition*#2, the *DeploymentManager* is triggered to find an alternative path to complete the workflow from the Partition Tree. The *DeploymentManger* calls eSC1's APIs to deploy the alternative partition in eSC1, and then sets the outputs of *Partition*#1 as the inputs for the alternative partition.

The Exception Handler is invoked when exceptions occur. In order to test the system, an "Exception Generator" component was developed to simulate the exceptions. Figure 3.13 shows the architecture of the testing system, where the "Deployment Manager" receives an exception from the "Exception Generator" and then the "Exception Handler" decides on which e-Science Central instance it can be deployed ("Exception Handler" is a module of the "Deployment Manager", which is used to analyse and handle exceptions).

Table 3.6 shows the execution time of the partitions when e-Science Central is deployed in two virtual machines which have the same performance level when the sample pipeline workflow (shown in Figure 3.6) and the clouds and workflow blocks cost information(list in Table 3.4 and Table 3.5 respectively) are load into our tool. Based on the "Partition Tree"' the option: $A \rightarrow H$ is selected as the "best" option. The following three cases demonstrate that the system works in the case of no-exception, Type #1 and Type #2 exceptions respectively.

CASE1 :

Both eSC1(e-Science Central 1) and eC2(e-Science Central 2) (representing Cloud1 and Cloud0 respectively) have the times when they are available ("up") defined in the *ExceptionGenerator*. In the case of this example, this is from 0 to 100 seconds

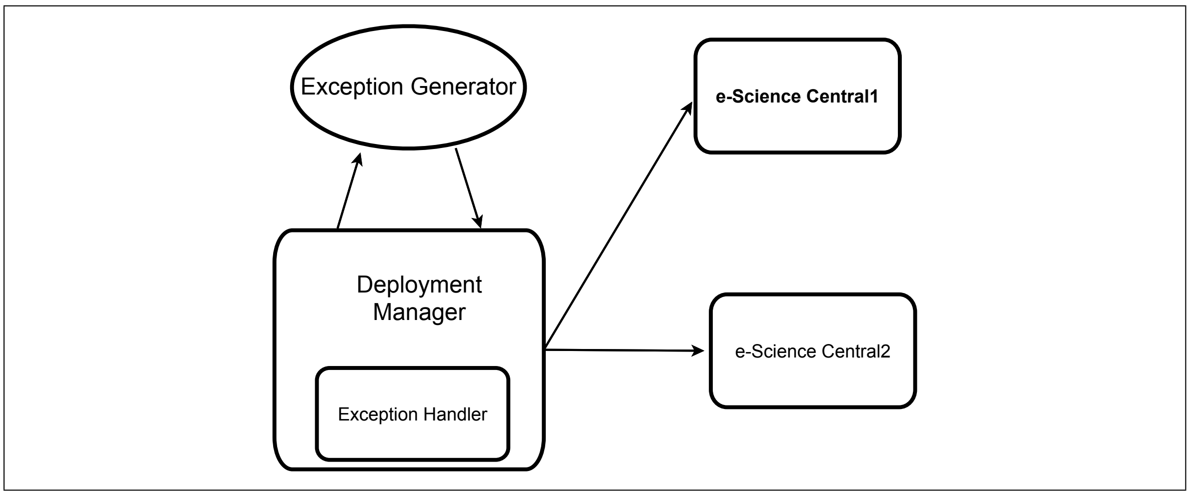


Figure 3.13: Testing System

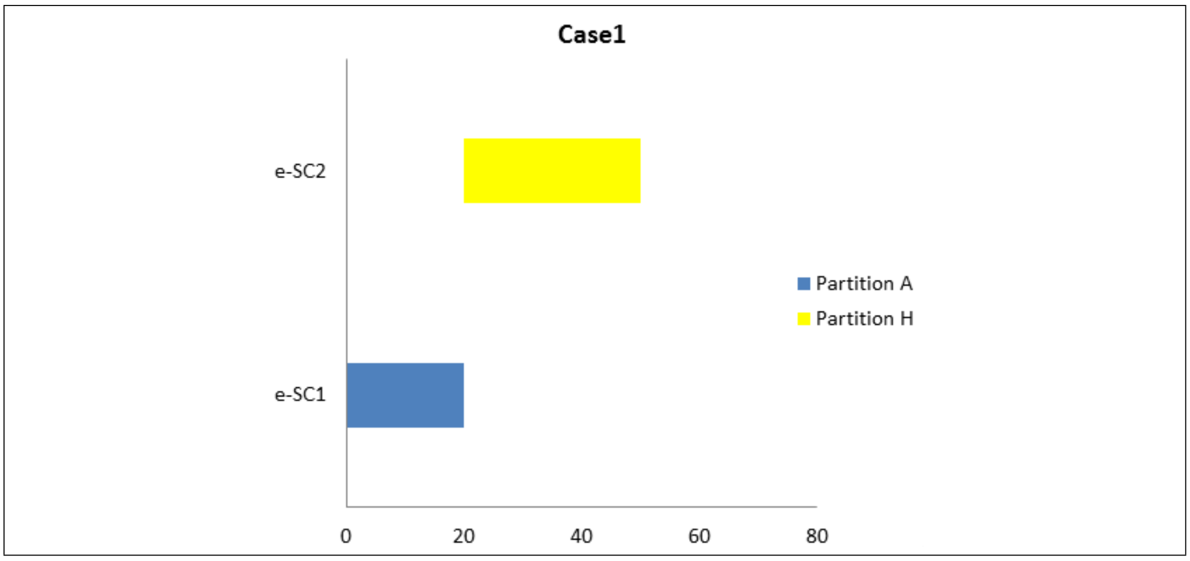


Figure 3.14: Case 1

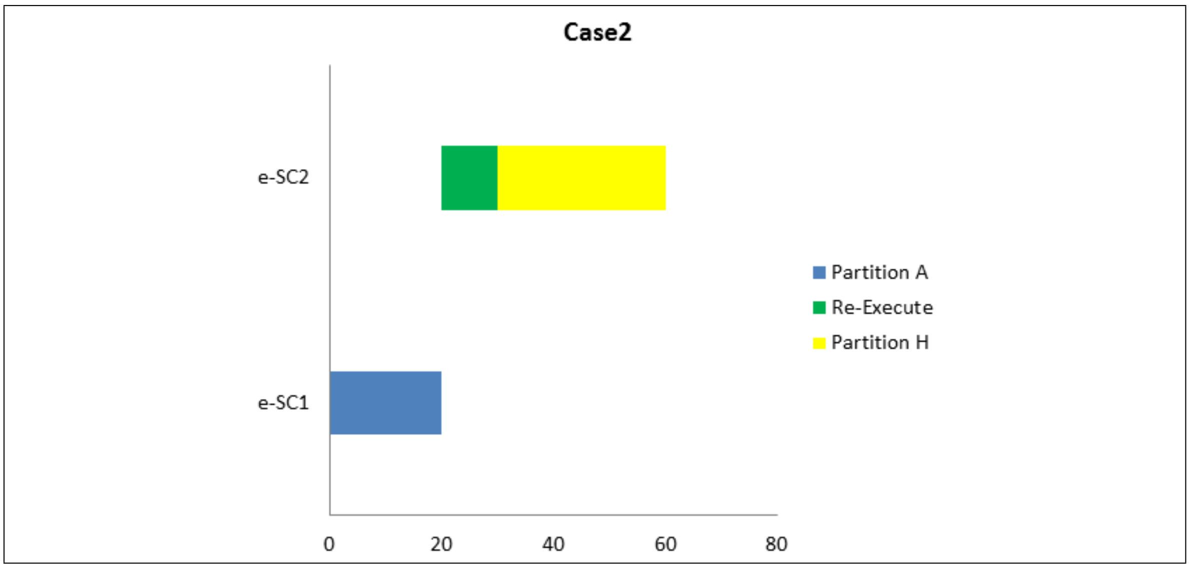


Figure 3.15: Case 2

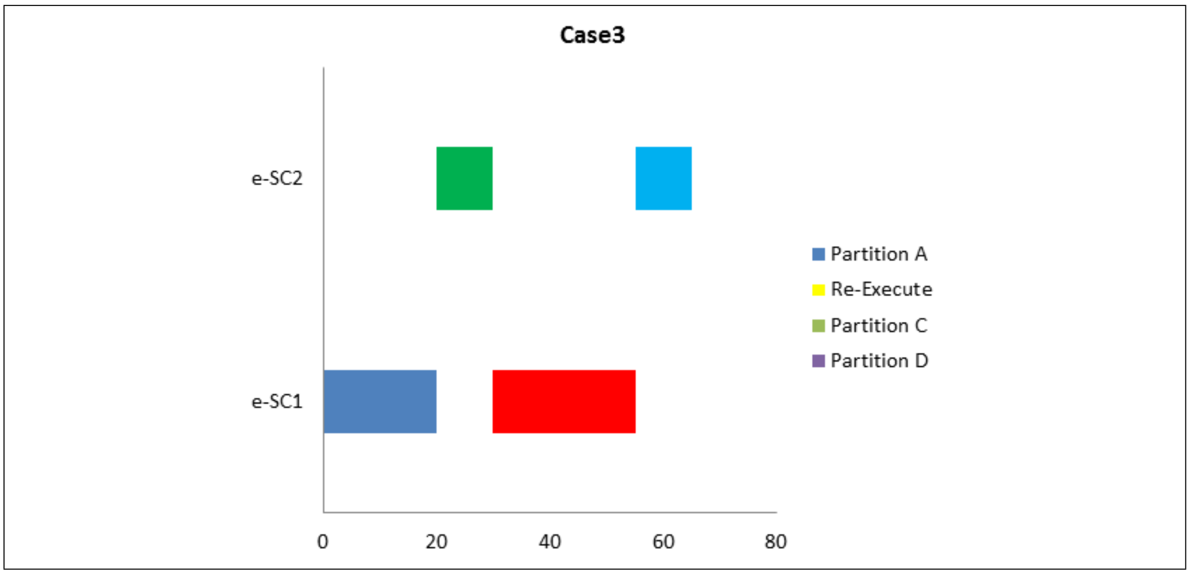


Figure 3.16: Case 3

for both clouds. The *DeploymentManager* requests the deployment of partition A. *DeploymentManager* is given information on the available clouds using feedback from *ExceptionGenerator*. In this case, eSC1 is available for partition A, so there is no exception. After partition A finishes, partition H can also be deployed, because the system time is 20 seconds and eSC2 is available for deployment. Figure 3.13 indicates the process.

CASE2 :

In this case, eSC1 is still available from 0 to 100 seconds, but eSC2 is unavailable from 0 to 30 second. This means that eSC2 is not ready to execute partition H when A has finished (at time 20). Figure 3.15 shows that 10 seconds are expended re-trying the execution of H on eC2 (as described above, this occurs when a type #2 exception is generated. At the end of this time, eSC2 becomes available again, and so Partition H can now be deployed and executed.

CASE3 :

The time eSC1 in which is available is from 0 to 100 seconds. However, eSC2 is only available between 50 and 100 seconds. As a result, it is not available after 10s of re-tries (the limit imposed for this example), and so partition H can not be deployed on it. Therefore the *ExceptionHandler* is invoked to find an alternative partition to handle the exception. After searching from the "Partition Tree", partition C is selected to replace H. Partition C has completed its execution at a system time of 55 seconds, at which time eSC2 is available for deploying partition D.

3.6 Conclusion

This chapter has described a new model which is used to automatically deal with cloud failure during the execution of a workflow on one or more clouds in a federation. The workflow is initially split into partitions, taking into account security and cost. These are deployed on a set of clouds, and execution begins. However, the contribution of this chapter is an algorithm that allows the execution to continue, if this is possible, when one or more clouds fail. The algorithm evaluates all the deployment options which can ensure that security requirements continue to be met, and determines the best possible option to complete the execution using the cloud(s) that is (are) still available.

The method extends the previous model [143] and integrates it with the DMI exception handling approach. It is shown how the DMI model can be applied to handle run-time exceptions and adapt the computation as necessary to find the best way to replace a failed workflow partition.

The approach has been applied to e-Science Central which is a high-level cloud platform, by creating a tool that implements the method. The evaluation shows that users can create or select a workflow and allow the tool to determine the "best" partitioning option. The partitions are automatically deployed across a set of available e-Science Central instances. If exceptions happen at run-time then the tool uses the method to work out the new best option, and automatically deploys the new partitions accordingly. The promise of the scheme is that it can exploit federated clouds to allow applications structured as workflows to run reliably even in the presence of cloud failure.

However, the partition method used in this chapter is not very scalable. It cannot handle the problem of partitioning complex workflows (with large numbers of services) over a federated clouds and meeting security requirements. Therefore, the next chapter describes and evaluates a scalable method to deploy large workflow applications over federated clouds.

4

A Scalable Method for Partitioning Workflows with Security Requirements over Federated Clouds

Contents

Summary

The previous chapter discussed how to partition a workflow application over federated clouds while meeting security requirements and minimising the monetary cost. However, the method that was presented cannot scale to handle large or complex workflows. The previous method ranks the cost of deployment solutions for the given cloud and workflow. The time complexity is exponential distribution. Therefore, a novel algorithm is desired for the deployment complex of workflow-based applications over federated clouds, while still meeting security requirements and minimising financial costs. The following sections introduce a method which can rapidly generate a deployment plan for a large complex workflow over a federated cloud which meets security requirements and minimises costs.

4.1 Introduction

There are now several cloud providers, including Microsoft, Google and Amazon, and each is making multiple, geographically distributed cloud data centres available. This enables customers to select an individual cloud data centre based on their requirements in terms of price, functionality, latency and governance regulations. However, it also opens the possibility of running applications over a set of cloud data centres to meet availability requirements. This has led to an interest in the idea of federated clouds [146][71], but the main drive for these has been security.

As discussed in the last Chapter, some organisations have sensitive data or services that they are not prepared to host on a public cloud. A potential solution to this also comes from the idea of federated clouds: deploy those parts of applications that are sensitive on trusted internal IT resources within an organisation (on what have come to be known as private clouds), but allow those parts with fewer security requirements to be deployed on public clouds where they can take advantage of their scalability and cost benefits.

Achieving this is not necessarily straightforward, especially for complex applications. There can be a very large number of options for how to deploy the data and services across a set of clouds, each of which will have a different cost model. Therefore, this chapter tackles one of the main problems faced by organisations wishing to exploit federated clouds: how to select a deployment option that meets their security requirements, while at the same time minimising the cost that a given deployment will incur.

From the literature review chapter, it is clear that a few work has been conducted of optimising cost through scheduling workflow over different resources. For example, In [59], the authors introduced a pricing model and a truthful mechanism for scheduling workflow applications to different clouds. Chen and Deelman have described in [39] how to use Genetic Algorithm (GA) to map a workflow to different clouds in order to minimise the monetary cost. While both of the papers are focused on minimising the monetary cost of completely executing workflow applications, neither consider the security issues that can limit the set of options.

In the study most relevant to the present research [70] an approach was proposed to partition business workflow applications over a set of clouds to satisfy security constraints. However, the authors focused on minimising communication to improve the reliability of the workflow enactment. In contrast, our aim is to minimise the monetary cost of the enactment of scientific workflows which are often characterised by high demand in terms of processing power and/or requiring the transfer and storage of large amounts of data.

Therefore, in this chapter we describe the design and evaluation of an algorithm to deploy an application structured as a workflow over a federated cloud in order to exploit the strengths of each cloud. The algorithm improves on the method presented in the previous chapter, making it suitable for large workflow applications. The previous method, based on the Bell-LaPadula [21] Multi-Level Security model [36], gave a solution for deploying a workflow over a set of clouds to meet certain security requirements. However, in order to find the cheapest workflow deployment, all of the potential deployment options need to be listed and ranked to find the cheapest. Although this method is optimal and guarantees that the result is the cheapest deployment (we use the term “ACO” which means Always Cheapest Option to represent this method in the following sections), it is not very scalable. For example, it can take more than 15 minutes to optimally partition a workflow comprising 12 blocks (services and data items) over 4 clouds. The complexity of the method is $O(c^s)$, where c is the number of clouds and s is the number of blocks. Therefore increasing the number of blocks in the workflow, raises the planning time exponentially.

Consequently, to handle more complex workflows and larger cloud federations our idea is to sacrifice some cost in order to reduce the time to produce a recommended deployment option. We present an approximate algorithm which still meets Bell-LaPadula security requirements. Its time complexity approximated closely to $O(2 \cdot s \cdot c)$ but gives an acceptable yet suboptimal result in terms of costs.

The rest of the chapter is structured as follows. The security problem and the cost issue are discussed in Section 4.2. Next, the algorithm is presented in Section 4.3, followed by an illustrative example. In Section 4.5 the complexity of the algorithm is briefly analysed. Finally, our experimental results are discussed and conclusion are drawn.

4.2 Problem Description

In this thesis we use the shared model (the workflow model in Chapter 3) to represent the workflow based application. Furthermore, we also share the same security model as last chapter. The following subsection introduces a novel cost model to calculate the cost of partitioning a workflow application over federated clouds.

4.2.1 Cost Model

We assume that the clouds are linked in a fully connected topology but the data can be freely transferred between clouds only if the security requirements described above are met. Additionally, a cloud can run several services at the same time. To represent cost we first define some basic metrics of our cost model:

- *Compu* is a $|S| \times |C|$ matrix that represents computation cost such that $Compu_{s_i}^{c_h}$ is the cost of running service s_i on cloud c_h.
- The matrix *Com* represents a unit cost of data transmission from one cloud to

another. For example, $Com_{c_h \to c_f}$ means the cost of transferring 1GB of data from cloud c_h to c_f.

- $CStore$ is a vector for describing the cost of a unit of data stored in a cloud for a unit time $CStore_{c_h}$, for instance, denotes the cost of stored 1GB of data for 1 hour on cloud c_h. This cost is only charged by source clouds when the data crosses cloud boundaries.

- The storage time of each unit of data is denoted in the matrix $TStore$. For instance, $TStore_{i,j}$ is the storage time of $d_{i,j}$, which is equal to the sum of the execution time of services s_i and s_j and plus the data transfer time if crossing cloud boundaries.

Given these basic metrics we now define a set of cost functions that we will use later in our algorithm:

- First is the data storage cost:

$$SCOST(d_{i,j}^{c_h}) = d_{i,j} \times TStore_{i,j} \times CStore_{c_h}$$

  where $d_{i,j}^{c_h}$ represents data $d_{i,j}$ stored on cloud c_h. Where data is transferred from cloud c_h to another cloud (to transfer it from one partition of the application to another held on a different cloud) we make the assumption that data only remains stored on the source cloud so as not to double-account for the cost. A reason for storing the output of a partition even after the data it generates has been sent to another cloud is that if the destination cloud fails, it provides a way to continue the computation on another cloud without having to restart the execution of the whole workflow.

- The communication cost of a set of data transferred from service s_i to s_j, which are deployed on cloud c_h and c_f respectively, can be defined as:

$$CCOST(s_i^{c_h}, s_j^{c_f}) = d_{i,j} \times Com_{c_h \to c_f}$$

Note that when both s_i and s_j are deployed on the same cloud, the communication cost is 0.

- $SCOST$ and $CCOST$ are used to derive the key functions in our algorithm, SOC and COD. $SOC(s_i^{c_f})$ is the costs incurred in bringing the input data consumed by service s_i to cloud c_f before the service is executed. This includes the storage cost and communication cost

$$SOC(s_i^{c_f}) = \sum_{s_n^{c_h} \in Pred(s_i)} CCOST(s_n^{c_h}, s_i^{c_f}) + \sum_{s_n^{c_h} \in Pred(s_i)} SCOST(d_{n,i}^{c_h})$$

where $Pred(s_i)$ represents the set of immediate predecessor services that produce data consumed by service s_i. Furthermore, $s_n^{c_h}$ indicates that one of s_i's predecessor service s_n was running on cloud c_h. If a service has no predecessors (i.e. it is one of the initial services in the workflow), then:

$$SOC(s_{entry}^{c_h}) = 0$$

The COD function denotes the extended cost of having a service deployed on a specific cloud. It is calculated by adding the computing cost of the service s_i to the transmission cost and storage cost of data sent from any of its predecessor services that are not in the same cloud.

$$COD(s_i^{c_f}) = Compu_{i^{c_f}} + SOC(s_i^{c_f})$$

- We define a $|S| \times |C|$ matrix $PrePLAN = [preplan_{i,h}]$ which is used to determine the initial deployment of workflow services to clouds. Each value in the matrix is defined as $preplan_{i,h} = COD(s_i^{c_h})$. Thus, each row of the matrix therefore contains COD values for a selected service as deployed on each specific cloud (except where deploying the service on a cloud would violate the security requirements). The initial deployment is then based on the smallest COD value in each row (i.e. for each service). Later, to improve this initial deployment we use the "Re-Deployment Method" (discussed in the following section) which generates the final deployment matrix for each service. The final deployment is stored in an $|S| \times |C|$ adjacency matrix DEP, where $dep_{i,h} = 1$ indicates that service s_i is deployed on cloud c_h. Consequently, the total COST of deploying a workflow application is:

$$COST = \sum_{\substack{s_i \in S \\ h \in C \; dep_{i,h}=1}} COD(s_i^h)$$

4.3 Algorithm

The COD function is used to determine the initial deployment of services to clouds. It does not provide optimal deployment because it takes into account only the local costs related to each single service considered in isolation. Thus, the idea of our algorithm is to use more information in the planning of the final deployment and to make a short term sacrifice for long term benefit. We call our algorithm NCF (Not Cheapest First). The algorithm consists of three steps. It starts by applying security rules to verify whether or not security requirements are met by the original workflow. We use rule CS to verify security for deploying services to clouds. Next, if the security requirements are met, we use COD to calculate the initial deployment matrix $PrePLAN$. Finally, the "Re-Deployment Method" is applied to improve the initial deployment, which works by combining services deployed in different clouds into a single cloud. The overall aim is to detect whether the costs can be reduced by avoiding intercloud communication and related storage costs.

The following sections describe the steps of the algorithm in more details.

4.3.1 Workflow Security Checking

The workflow is valid iff all return NRU and NWD values are true. Otherwise, the workflow is invalid, the security check returns an error and the whole algorithm stops. The pseudocode of the "Workflow Security Check" is shown in Algorithm 1.

Algorithm 1 Workflow Security Check

```
$D$ set of dependencies between related services
$S$ set of service
Secure: =True
for $d_{i,j} \in D$ do
    if not $(NRU(d_{i,j}, s_j)$ and $NWD(s_i, d_{i,j}))$ then
        Secure:=False
        Stop
    end if
end for
```

4.3.2 Initial Deployment

The initial deployment of services is based on the smallest COD value of each service taking into account security requirements checked by the CS rule. If a cloud c_h generates the smallest COD value but does not meet requirements imposed by CS, a cloud with the second smallest COD value is considered. The algorithm works until it finds a cloud that can meet the security requirements and the COD value associated with this cloud is stored in the vector REC. If no cloud is found that meets CS, the algorithm stops. Algorithm 2 shows pseudocode for the initial deployment step.

Algorithm 2 Initial Deployment

```
$S$ set of services in the workflow
$REC$ init with INF
$PrePLAN$ init with zero
for $s_i \in S$ do
    for $c_h \in Cloud$ do
        if $CS == 1$ then
            if $COD < REC_i$ then
                $REC_i = COD$
                $PrePLAN_{i,h} = 1$
            end if
        end if
    end for
end for
```

4.3.3 Re-Deployment Method

The core idea behind the “Re-Deployment Method” is to avoid scheduling services to clouds which bring excessive communication cost. To realise this we use two functions "detect" and "replace" which are defined as follows:

$$detect(s_i) = \begin{cases} case1, & a \ \& \ \neg b \\ case2, & \neg a \ \& \ b \\ case3, & a \ \& \ b \\ case4, & \neg(a \ \& \ b) \end{cases}$$

$$s_{i,max} = max_{s_j \in Child(s_i)}(REC(s_j))$$

$$SETP(s_i) = Parent(s_{max}) \cup s_{max}$$

$$SETC(s_i) = Child(s_i) \cup s_i$$

$$MIN(SET) = min_{c_h \in C}(SET)$$

$$\sum_{s_h \in SETP(s_i)} REC(s_h) > MIN(SETP(s_i)) \qquad \text{a}$$

$$\sum_{s_h \in SETC(s_i)} REC(s_h) > MIN(SETC(s_i)) \qquad \text{b}$$

From the descriptions above, the minimal COD value of each service and the pre-planned deployment are recorded in REC and $PrePLAN$ respectively. The *detect* function determines four different cases based on this and additional information:

- $s_{i,max}$ denotes the service s_i's child service with a maximum COD value.
- $SETP(s_i)$ is a set which includes service $s_{i,max}$ and all its parent services.
- $SETC(s_i)$ includes s_i and all its child services.
- $MIN(SET)$ is the minimum cost of deploying the services from SET on a single cloud which meets the security requirements of all services in SET.
- a is true if the cost of the initial deployment of the services in $SETP(s_i)$ is greater than the cost of $MIN(SETP(s_i))$
- b is true if the cost of the initial deployment of the services in $SETC(s_i)$ is greater than the cost of $MIN(SETC(s_i))$

Given all this information, *detect* returns four different sets of services. In case1 it returns $SETP(s_i)$, in case2 it returns $SETC(s_i)$, in case3 it returns one of $SETP$, $SETC$ which has the smaller MIN value. Finally, in case4 it returns $\{s_i\}$.

After the services are selected by *detect*, the *replace* function is invoked to assign these services into a cloud which minimises deployment cost. The pseudocode is shown in Algorithm 3.

Service	C0	C1
$S1$	50	100
$S2$	100	200
$S3$	150	250
$S4$	160	200

Table 4.1: The CPU Cost in Clouds

Cloud	C0	C1
C0	0	10
C1	20	0

Table 4.2: Cloud Communication Costs (per GB)

Data	Time (hour)	Size(GB)
$d_{s1,s2}$	15	10
$d_{s1,s3}$	15	20
$d_{s2,s4}$	5	8
$d_{s3,s4}$	5	6

Table 4.3: Data Size and Data Storage Time

Cloud	Cost
C0	0.1
C1	0.2

Table 4.4: Cloud Storage Cost of Clouds (per GB Hour)

Algorithm 3 Re-Deployment Method

```
$US = S$
**for** $s_i \in US$ **do**
    switch(*detect*($s_i$))
    case1:*replace*($SETP(s_i)$)
    case2:*replace*($SETC(s_i)$)
    case3:
    **if** $MIN(SETP) > MIN(SETC)$ **then**
        *replace*($SETC(s_i)$)
    **else**
        *replace*($SETP(s_i)$)
    **end if**
    case4: *replace*($\{s_i\}$)
**end for**
```

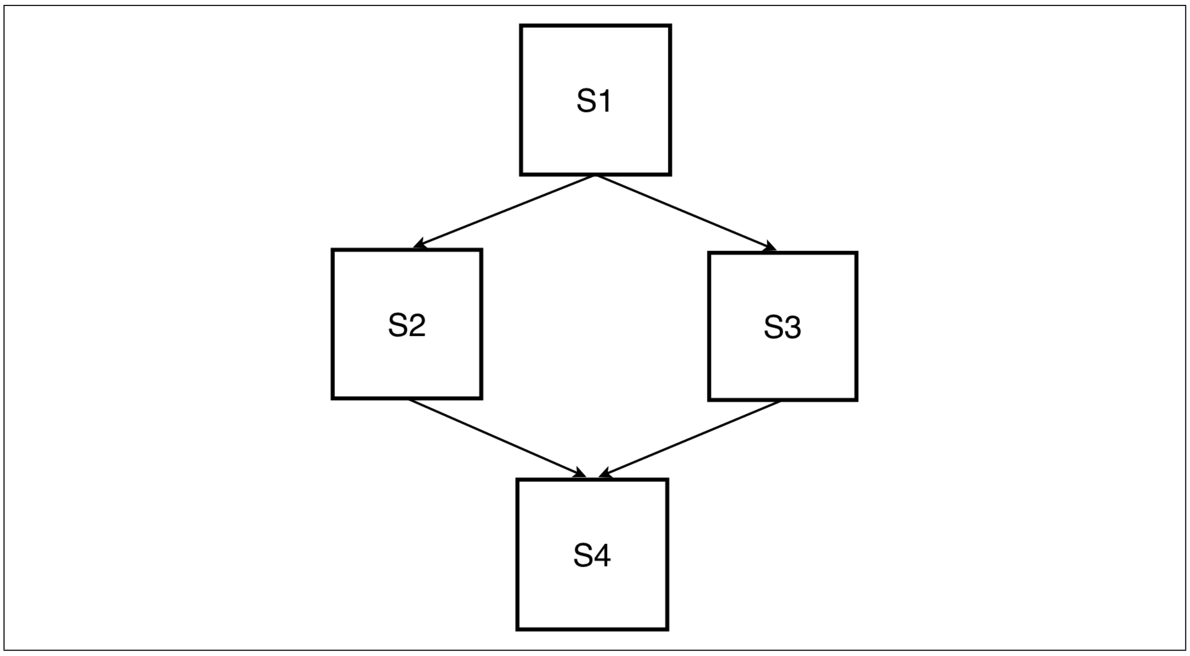


Figure 4.1: Sample Workflow Application

Matrix	Element	Clearance	Location
SecS	S1	0	0
	S2	1	1
	S3	1	1
	S4	1	0
SecC	C0		0
	C1		1
SecD	$d_{s1,s2}$		0
	$d_{s1,s3}$		0
	$d_{s2,s4}$		1
	$d_{s3,s4}$		1

Table 4.5: The Security Levels for Services, Data and Clouds

4.4 An Illustrative Example

The workflow, shown in Figure 4.1, is used to demonstrate the algorithm. More complicated workflows are evaluated in section 4.6.

Workflow and cloud cost information is shown in Tables 4.1, 4.2, 4.3, 4.4. Additionally, the security levels of the services, data and clouds are also required, and these are given in Table 4.5.

Before assigning the services to clouds, the security of the workflow is checked. From Table 4.5, every service and data meet the security rules NRU and NWD.

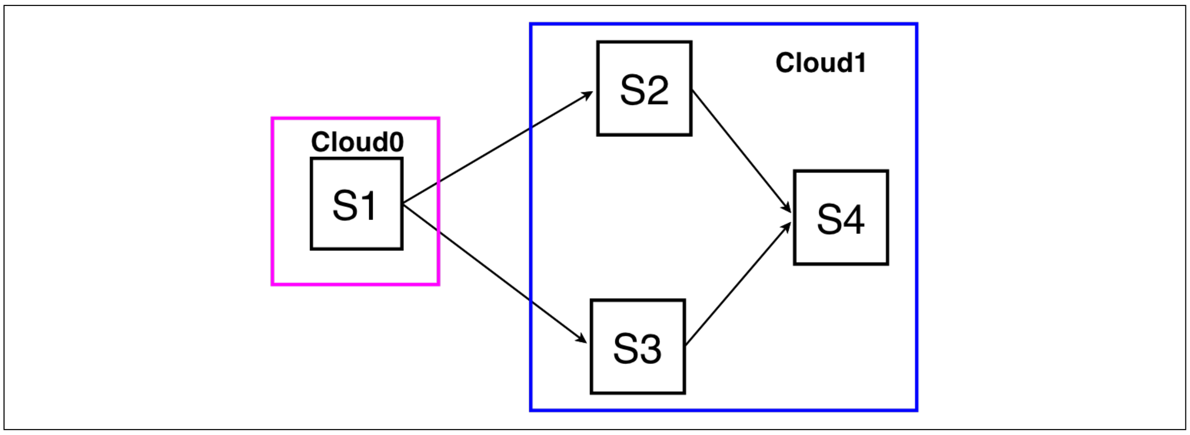


Figure 4.2: Pre-Deployment Workflow System

Next, the workflow has to be pre-assigned by following the function COD and security rule CS. The initial deployment is shown in Figure 4.2 and the cost is 1045.

To improve on the initial deployment, the "Re-Deployment Method" is applied. When $detect(s1)$ is applied, the value of $\sum_{s_h \in SETP(s_i)} REC(s_1) = 530$ and $MIN(SETP(s1)) = 300$ ($SETP(s1)$ includes $s1$ and $s3$, and " Cloud1" is the one to minimise the cost). In addition, $\sum_{s_h \in SETC(s_i)} REC(s_1) = 845$, while $MIN(SETC(s1)) = 550$ ($s1$, $s2$ and $s3$ are selected for $SETC(s1)$, and "Cloud1" is also the best choice). In this example, both case1 and case2 are satisfied. However, the former case has a cheaper MIN value, therefore assigning services in $SETP(s1)$ to "Cloud1".

Next, after applying the *detect* method to "s2"and "s2" belonging case4, that are assigned to "Cloud1" after $replace(s2)$ is invoked. Similarly, service "s4" is also allocated to "Cloud1" after the application of the "Re-Deployment Method". Consequently, all services are assigned to "Cloud1", and the cost is 750.

4.5 Complexity Analysis

In the proposed algorithm, we split the workflow security check and pre-planned deployment into two parts, because this can make the algorithm easier to understand. However, they can be combined in one step. This makes calculations more efficient and results in complexity $O(|E| \times |C|)$, where E is the set of data dependency edges and C is the set of clouds. The *detect* and *replace* functions compare the cost of a service and its immediate predecessor services in a pre-planned situation with the cost of deploying these services in one cloud. However, the complexity of the algorithm is also affected by the structure of the workflow. If the workflow is linear, the complexity in the worst case becomes $O(|E + S| \times |C|)$. Conversely, for a star-shaped workflow the best case complexity is $O(|E| \times |C|)$.

4.6 Experimental Results and Discussion

4.6.1 Randomly Generated Workflows

In this case, the workflow is represented by a distance matrix D in which if $d_{i,j}$ is greater than 0 that means service s_i sends data to service s_j. The value represents the size of data that is transferred between them. We create a $n \times n$ matrix with "-1" as the initial value. Next, the values of the matrix are assigned randomly, however we make sure that workflows are connected and acyclic.

In the experiment, we consider five different clouds that are assigned security levels from 0 to 4 randomly and stored in SecC vector. The other required matrices are generated in a similar way.

The experiments are implemented in the Java language and run on a 4-Cores machine with 2 GHz Intel Core i7 processor, 8G RAM and OS X Mavericks.

Figure 4.3a denotes the execution time of the NCF algorithm for different workflows with different numbers of blocks, as shown on the x-axis from 2 to 30. In all cases the time taken was less than 1 millisecond. However, the execution time does not grow linearly with the number of blocks because the time complexity depends also on the structure of the workflow. In our experiment we can control the number of clouds, services and edges but the structure of the workflow is randomly generated. Note also that due to huge differences in the execution time of the NCF and other algorithms, we present in the Figures ratio alg/NCF rather than absolute execution time.

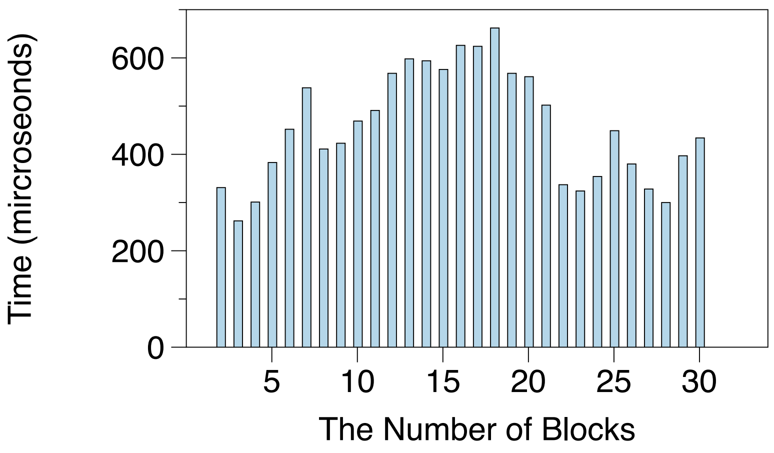
The time cost of each column is the mean value of ten different structures of workflows. For example, we calculate the time cost of a workflow with five blocks by mean of the time cost of the algorithm which is applied to ten different 5 blocks workflows. As Figure 4.3b shows, the time cost of the ACO significantly increases when the workflow has more than 10 blocks. Figure 4.3b compares the ACO and NCF algorithms. The x-axis shows the number of blocks in random workflows, which is from 2 to 12 blocks, and the y-axis is the ratio of execution time of ACO to the execution time of NCF. A sudden surge can be noted in the run time of the ACO algorithm when workflows have more than 10 blocks

The relative monetary cost of both algorithms is displayed in Figure 4.3c. In most cases NCF is very close to the optimum with the worst case result we found being 25% higher than reported by ACO. The reason for this gap is that NCF can only improve the result by adjusting the deployment of the immediate predecessor services of the current planning service. However, further away services are not considered by our algorithm.

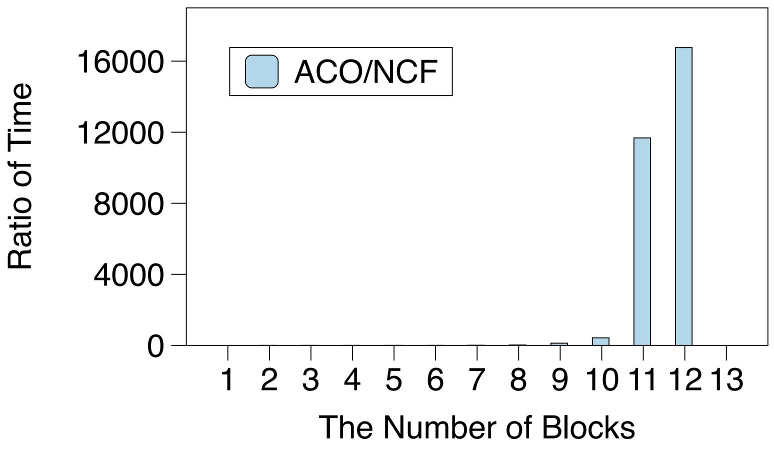
To generate results for more complex workflows by using ACO is very costly in time; therefore we also compare our algorithm with GA and Greedy Algorithm (GR). Both are popular methods used to deploy large workflow systems on federated clouds as proposed in [39]. As Figure 4.4a shows, the deployment options generated by our algorithm are closer to the optimal options. For a 30-block workflow, NCF generates a solution that is 20% less costly than that of the GA and 36% less costly than the GR Besides this, NCF is thousands of times faster than GA to generate a deployment option, and only 13 times slower than GR in the worst case, as shown in Figure 4.4b, 4.4c. These experimental results are generated by testing more than 750 workflows, between 5 blocks and 30 blocks.

4.6.2 Workflows from a Real Scientific Application

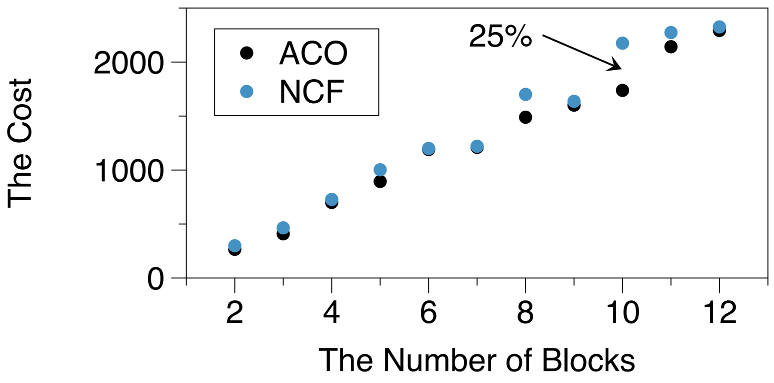
To verify our algorithm on a real workflow, we used one from the Cloud e-Genome project[31] (Figure 4.5). The project's overall goal is to facilitate the adoption of genetic testing in clinical practice on a population wide scale. To realise this goal, Cloud e-Genome uses workflow modelling to program the whole exome sequencing pipeline, cloud computing to run the workflows on a large scale and provenance of workflow enactment to achieve reproducibility. All these aspects are supported by the e-Science Central platform (e-Science Central) [81] used to develop the pipeline



(a) Execution Time of NCF

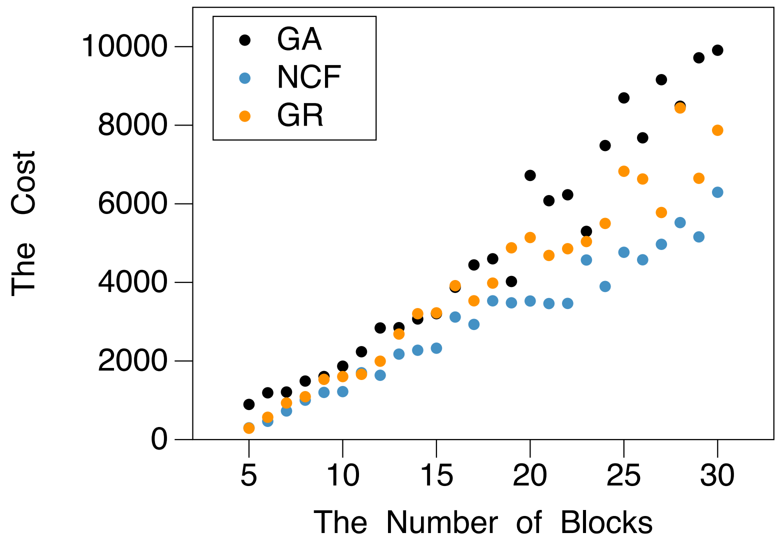


(b) Time Ratio between ACO and NCF

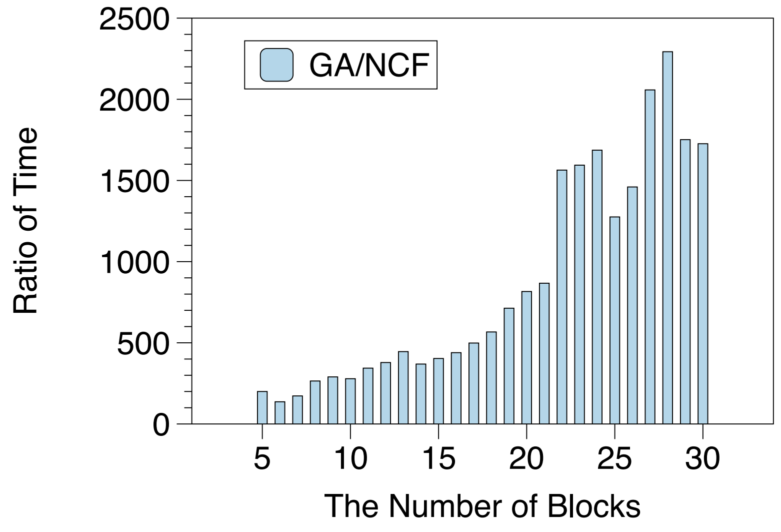


(c) The Cost of ACO and NCF

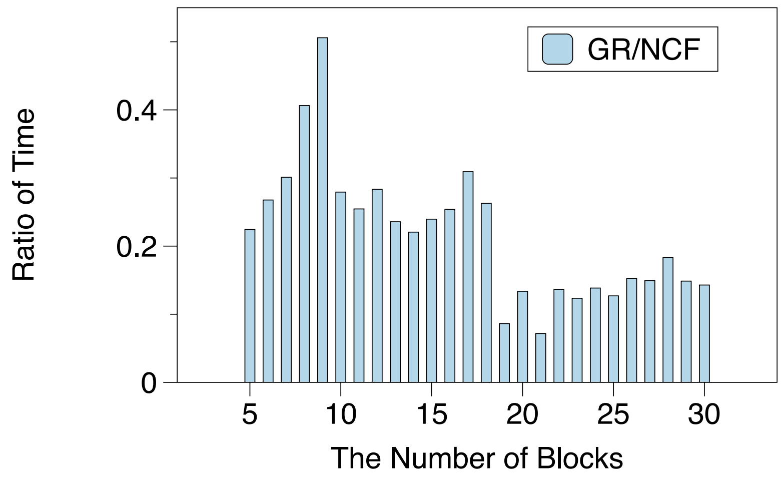
Figure 4.3: Results of ACO and NCF



(a) The Cost of NCF, GA and Greedy



(b) Time Ratio between GA and NCF



(c) The Cost of Greedy and NCF

Figure 4.4: Results of GA, GR and NCF

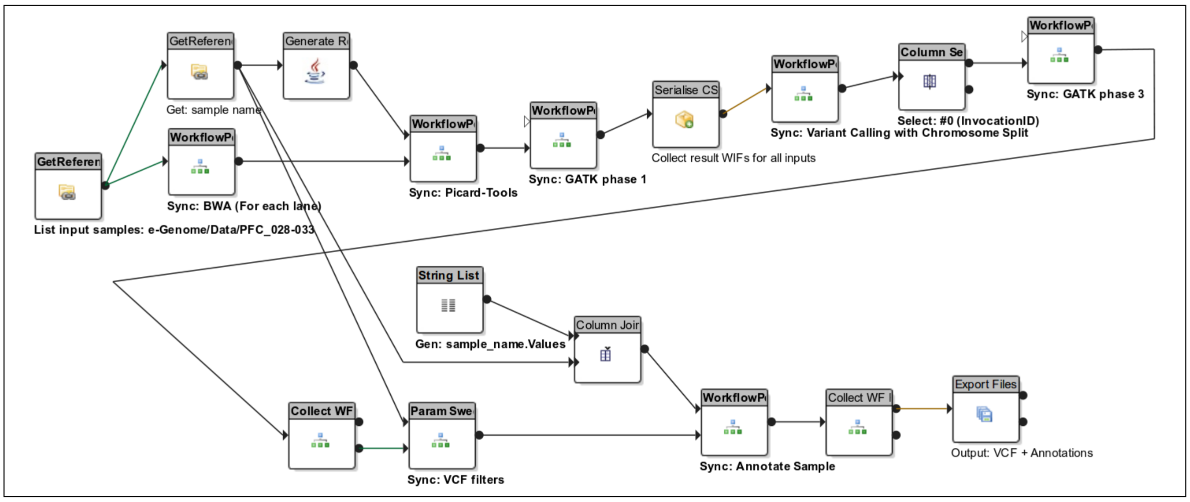


Figure 4.5: e-Genome Workflow

Although security aspects are not a central focus of this pilot project, guaranteeing that human genome data can be securely processed on the cloud is a key issue. Therefore, we selected a workflow from Cloud e-Genome and modelled its security requirements by assigning security levels as shown in Tables 4.6 and 4.7. The services “Data In” and “Data Out” are e-Science Central storage services which are not represented by blocks. Therefore they are not included in Figure 4.5

Using logs collected by e-Science Central, we are able to determine the size of data transferred between workflow blocks and execution times. Table 4.6 includes the execution times in hours, while a value of 0 means that the times was less than 1 minute. Table 4.7 shows the data sizes in GB, where 0 means that the size was less than 1 MB. From this data we calculated the cost of running the workflow in clouds offered by two major cloud providers.

The pricing of the clouds is shown in Table 4.8. We chose the same type of Virtual Machines in both clouds and also refer to public and private cloud setup offered by both providers; the private cloud setup is more secure yet more expensive. In the same table we present also data transfer costs between the clouds.

From these inputs, services $S1$ and $S5$ are assigned to Azure public cloud, and others are located in Azure private cloud. The total cost is 84.319 dollars.

Service	Name	Clarence	Location	Execution Time(hours)
Data In	S1	0	0	0
List Samples	S2	0	0	0
Get Samples Info	S3	0	0	0
Generate MetaData	S4	0	0	0
Align Sequences	S5	0	0	6.42
Clean Sequences	S6	0	0	4.05
Recalibareate Sequences	S7	0	0	11.45
Serialise CSV	S8	0	0	0
Export File	S9	0	0	0
Detect Variants	S10	1	1	4.46
Column Select	S11	0	0	0
CSVExport	S12	0	0	0
Recalibareate Variants	S13	1	1	1.425
Filter Variants	S14	1	1	0.5
Generate String List	S15	0	0	0
Column Join	S16	0	0	0
Annotate Sample	S17	1	1	0.86
Export File	S18	1	1	0
Data Out	S19	1	1	0

Table 4.6: Services Representation and Security and Execution Time

4.7 Conclusion

In this chapter, we have described a novel, efficient scalable algorithm to automatically deploy complex workflows over a federated cloud while meeting security requirements and minimising execution cost. The main contribution is to redesign the exact algorithm presented in [143] to enable it to be applied in real world scenarios by reducing the time it takes to generate a low-cost, secure partitioning option.

The algorithm was tested on randomly generated workflows and real world scientific workflows. Comparing with other methods, the time complexity of our algorithm has a significant advantage, and the cost is very close to the cost of the optimal deployments and less than the cost of deployments that generated from widely used algorithms.

In next chapter, we will introduce a framework which provides a generic way to deploy scientific workflow over federated cloud to meet security requirements and minimise the cost. Furthermore, cloud availability change is also considered, and the framework will been evaluated through both simulation tool and realistic cloud based workflow platforms.

Service	Location	Size (GB)
$S_{1,2}$	0	0
$S_{1,3}$	0	0
$S_{1,5}$	0	24.99
$S_{2,3}$	0	0
$S_{2,5}$	0	0
$S_{2,15}$	0	0
$S_{2,17}$	0	0
$S_{3,4}$	0	0
$S_{4,6}$	0	0
$S_{5,6}$	0	18.63
$S_{6,7}$	1	15.56
$S_{7,8}$	0	0
$S_{7,10}$	1	6.87
$S_{8,9}$	0	0
$S_{8,10}$	0	0
$S_{9,19}$	0	0
$S_{10,11}$	0	0
$S_{10,13}$	1	0.074
$S_{11,12}$	0	0
$S_{11,13}$	0	0
$S_{12,19}$	0	0
$S_{13,14}$	1	0.09
$S_{14,17}$	1	0.055
$S_{15,16}$	0	0
$S_{16,17}$	0	0.11
$S_{17,18}$	1	0
$S_{18,19}$	1	0.11

Table 4.7: Data Security

Cloud	Pu1	Pr1	Pu2	Pr2
Security	0	1	0	1
CPU	1.68(/hour)	3.41(/hour)	1.40(/hour)	3.23(/hour)
EC2	0	0.08	0.02	0.07
EC(private)	0.08	0	0.07	0.12
Azure	0.06	0.11	0	0.1
Azure(private)	0.11	0.16	0.1	0

Table 4.8: Cloud Cost (U.S. Dollar per GB)

5

A Framework for Dynamically Deploying Workflow over Federated Cloud to Meet Security Requirements and Minimising the Cost

Contents

Summary

In the previous Chapter, we have proposed a scalable algorithm to partition a complex workflow over federated clouds. However, the algorithm considered only one type of security model and it lacks of solutions to handle the cloud availability change in the federation. In Chapter 3 we have introduced a method to dynamically handle the clouds fail, but this method cannot be applied to a complex workflow application, and also do not consider the situation of new clouds becoming available. Therefore, in this chapter we propose a generic framework to partition a scientific workflow over federated clouds to meet security requirements while minimising the monetary cost. To this end, the framework needs to support more general workflow and security models, and the ability to addres other non-functional requirements such as reliability and performance is reqired. Furthermore, our framework also provides a dynamic rescheduling method to deal with cloud availability changes during workflow execution.

5.1 Introduction

Previous chapters have focussed on exploiting clouds that differ in security and price allowing workflow applications to optimise their deployment over a set of clouds. Furthermore, Chapter 3 introduced a tool which can partition a workflow application over federated clouds and interact with a method to handle the failure of clouds during the workflow execution. However, the method is not very scalable for complex workflow applications and a generic framework is lacking which would allow users to quickly adapt their own workflow application and security models to optimise deployment. In addition, we aim to support dynamic federations in which clouds can join and leave at any time. Therefore, a method is needed to handle cloud availability changes during workflow execution.

In paper [89] Jrad et al. proposed a cloud broker to schedule larger scientific workflows over federated clouds to match the QoS and cost requirements. However, the static scheduling algorithms involved are unable to manage changes in cloud availability.

In this chapter, we propose DoFCF (Deploy on Federated Cloud Framework), a framework for deploying workflows over federated clouds that meets security requirements and optimises cost. The proposed framework can be efficiently adapted to suit different structures of scientific workflows and different security models. The framework also adopts a general optimisation process to minimise the cost for executing a workflow application over different types of cloud.

The proposed framework provides a set of solutions for workflow scheduling, including where to deploy tasks, when to start each service and how to handle the cloud environment changes. The deployment of the workflow over federated clouds is based on a set of specific security requirements and minimisation procedures. Additionally, DoFCF offers a solution to dynamically reschedule a running workflow to new clouds, in order to complete the execution of an unfinished workflow or save on costs.

To this end we list the following core contributions:

- A framework for modelling, quantifying and guiding the deployment of workflows over a federated cloud to meet security requirements while minimising the cost. The framework also considers the situation of a change in the available clouds during workflow execution.
- Presentation of an analysis of the existing state-of-the-art algorithms for optimising deployment. In addition, we compare and adapt those algorithms to our framework to achieve rapid exploration for a possible deployment solution, while handling cloud resource changes.
- Implementation of DoFCF as a Cloud Service Broker [28] to deal with scientific workflow deployment over federated clouds, and dynamically handle the available clouds change.
- Evaluation of the implementation on CloudSim [32], which is a Cloud simulator, and e-Science Central [81] (e-SC), a real scientific workflow based cloud platform.

The rest of this chapter is organised as follows. In Section 5.2 the basic models of the framework are discussed, including General Security model, Cost model, Cost optimisation problem and Handling Cloud change problem. Next, the state-of-the-art optimisation algorithms are analysed, in order to provide a general suggestion in applying algorithms to do the deployment. In Section 5.4, we evaluate the framework by using CloudSim, and also develop a tool to schedule the workflows over a set of e-SC instances to meet security requirements.

5.2 Basic Modelling Constructs

In this section, we present models of security requirements for deploying a workflow application over federated clouds. In addition, a general cost model - extending the cost model in previous chapter is used to calculate the monetary cost of deployment. Moreover, we present an optimisation model that can guarantee the deployment solution meets security constraints as well as minimising cost. Finally, a dynamic cost model is used to help reschedule the running workflow when the available clouds change.

5.2.1 General Security Model

An application's security can be improved using two approaches: firstly, byrefining the design and implementation of the application; secondly, by deploying the application over more trustworthy resources, such as shifting the application to a higher security server. In this chapter, we propose to increase the security of a workflow by adopting the latter approach. To achieve the enhancement in workflow security, we present two functions which are used to provide a concrete representation of different types of security requirements:

func1 is to embed constraints for d and s. According to the security model that will be used in the following sections, either d or s should be deployed on a cloud where can meet its security requirements.

We assume that each $o \in \mathcal{O}$ has its own security constraints and must be deployed on a cloud $c \in C$ which can meet those constraints. Therefore, $func1$ can be defined as:

$$func1 : \mathcal{O} \times C \rightarrow$$

$$\begin{cases} true, & \text{if } o \in \mathcal{O} \text{ can be deployed onto } c \in C \ (o^c) \\ false, & \text{Otherwise} \end{cases}$$

func2 represents the constraints for the whole workflow deployment $\mathcal{W}$. For example, the Common Vulnerability Scoring System (CVSS) [114] can generate consistent scores to accurately represent the impact of vulnerabilities.

In this function, Λ represents the possible deployment solutions for $\mathcal{W}$ over federated clouds C, and λ is one of the the deployments noting that $\{o_1^{c_i}, ..., o_n^{c_m}\}$ and H is one of the security requirements. Therefore function $func2$ can be defined as:

$$func2 : \Lambda \rightarrow$$

$$\begin{cases} true, & \text{if } \lambda \in \Lambda \text{ can meet the security constraint } H \\ false, & \text{Otherwise} \end{cases}$$

5.2.2 Cost Model

In the previous chapter, we have introduced a specific cost model for algorithm NCF to calculate the cost of deploying a workflow over federated clouds. However, in this section we propose a generic cost model which supports calculating the cost of deploying the most general scientific workflow over federated clouds.

Therefore, a set of cost functions is defined as follows:

- The first function is the data storage cost:

$$SCOST(s_i^c) = \sum_{d_{i,j} \in OUT} d_{i,j} \times T_{i,j} \times Store_c \tag{5.1}$$

  Where s_i^c means that service s_i is deployed on cloud c. OUT is a set of data dependencies, representing the data that are generated by s_i and transferred to its immediate successor s_j which is not deployed on c (note that if all immediate successors of s_i are on c, then $OUT = \emptyset$). $d_{i,j}$ represents the amount of data which is generated by s_i and consumed by s_j. $T_{i,j}$ denotes the storage time of data $d_{i,j}$, which is the required time starting from the generation of data until the completion of workflow execution. Finally, $Store_c$ is the cost of storing 1GB of data for one hour on cloud c.

In this model, we make an assumption that the data remains stored only on the source cloud to avoid double-accounting for the cost. The reason for storing the outputs of a service even after the generated data has been sent to another cloud is to be able to deal with a failure of the destination cloud. In this case, the stored data provides a way to resume the computation on another cloud without the need to restart the whole workflow execution. This process can be adapted to handle the cloud change problem.

- The second function, $CCOST$, is used to estimate the communication cost for transferring data between different services.

$$CCOST(s_j^c) = \sum_{d_{i,j} \in IN} d_{i,j} \times Com_{c',c} \tag{5.2}$$

This is the cost of the data transferred from the immediate predecessors of service s_j (denoted as IN). $Com_{c',c}$ represents the unit cost of transferring 1GB of data from cloud c' to c. However, if two services are deployed on the same cloud, the cost is zero, i.e. $\forall c' = c : Com_{c',c} = 0$.

- Finally, $ECOST(s_j^c)$ indicates the execution cost of service s_j on c. It is defined as:

$$ECOST(s_j^c) = T_j^c \times Exec_c \tag{5.3}$$

Where T_j^c is the execution time of s_j on cloud c, and $Exec_c$ represents the cost of using compute resources on c for one hour.

Based on the three cost functions, we can formulate the $COST(\lambda)$ function to define the total cost of a workflow deployment over a set of clouds:

$$\begin{aligned} COST(\lambda) = \sum_{s_i^c \in \lambda} (SCOST(s_i^c) + \\ CCOST(s_i^c) + ECOST(s_i^c)) \end{aligned} \tag{5.4}$$

5.2.3 Deployment Optimisation

As mentioned previously, we propose a model for optimising monetary cost as well as meeting the security requirements. Therefore, the optimisation problem is to find a deployment $\lambda \in \Lambda$ with two constraints: 1) the deployment λ must meet the security requirements by belonging to either $func1$ or $func2$. 2) the value of $COST(\lambda)$ should be minimised to obtain deployments with a low cost of execution. We express this problem as:

$$\begin{aligned}
&\textbf{minimise}\,(COST(\lambda)) \\
&\textbf{subject to}\,\forall\, o^c \in \lambda:\ func1(o^c) := true \text{ OR} \\
&\qquad func2(\lambda) := true \\
&\qquad \exists\, \lambda \in \Lambda
\end{aligned}$$

In the following, we use $func1$ as an example to prove that the optimisation problem is a NP-completeness problem.

Theorem: The optimisation is a NP-completeness problem.

Proof: we first verify that the problem of deploying a workflow over a set of clouds to meet security requirements is a NP problem (noting $\exists\, \lambda \in \sum(\Lambda, \mathcal{W})$, where $\mathcal{W}$ represents the security requirements).

The NP-completeness of optimising the cost can be illustrated as follows: we start by transforming PARTITION [65] (one of six core NP-complete problem) to our problem. Let the instance of PARTITION be a finite Set $A = (a_1...a_m)$ and a weight $w(a_i)$. We want to have two disjoint subsets A_1 and A_2 $A_1, A_2 \subseteq A$, where $A_1 \cup A_2 = A$ and $A_1 \cap A_2 = \emptyset$, such that $\sum_{a \in A1} w(a) = \sum_{a \in A2} w(a)$.

In order to reduce our problem to a PARTITION problem, we assume that a workflow has m numbers of $\mathcal{O}$, and two clouds are available for deployment. Further, we do not consider the security issue, which means any o can be deployed over any of the two clouds. Therefore, we can have two sets of deployments $C_1 = (o_1....o_m)$ and $C_2 = (o_1....o_m)$ over the two available clouds

Regarding our problem, we need to have two disjoint subsets C_1' and C_2', where $C_1' \cup$

$C'_2 = \mathcal{O}$ and $C'_1 \cap C'_2 = \emptyset$. This match the conditions of the PARTITION problem. Furthermore, $w(o)$ represents the cost of deploying o onto the cloud, so $\sum_{o \in C'_1} w(o)$ is the cost of set C'_1. However, the PARTITION problem is to find two disjoint sets with the same weight, which has the same complexity as our problem that is trying to find two disjoint sets while minimising the total cost, noting $min(\sum_{o \in C1'} w(o) + \sum_{o \in C2'} w(o))$.

5.2.4 Dynamic Cost Model

The dynamics of Cloud resources may affect workflow execution. Situations arise when individual nodes may fail during execution, or in some extreme cases the whole cloud is unreachable for several hours. As a consequence of such failures, workflow applications may not be executed to completion. Furthermore, new clouds, which may be attractive because they are cheaper or more secure etc., and they may become available during the execution of workflow applications. Therefore, to deal with the dynamism of cloud resources, we develop a new cost model that dynamically calculates the cost of deploying uncompleted services over the currently available clouds.

We assume a set *Selected* is composed of the services that need to be rescheduled, including unfinished services as well as the services that have been completely processed, and their outputs are the inputs of unfinished services, but the outputs have not been stored because of the failure of the clouds. The details are illustrated in Section 5.3.4.

Input is a set of data which has already ben generated from the processed services and is required for services in *Selected*, and stored in the available clouds. Based on the definition, we can have the initial cost for setting up a new deployment, which is the cost of storing the input data of *Selected*, can be defined as:

$$ICOST(Selected) = \sum_{d_{i,j} \in Input} d_{i,j} \times T_{i,j} \times Store_c \tag{5.5}$$

In addition, Λ' is the possible deployments of the services in *Selected* over the available clouds C. Consequently, the cost of the new deployment can be defined as:

$$DCOST(\lambda') = COST(\lambda') + ICOST(Selected) \tag{5.6}$$

Where $\lambda' \in \Lambda'$ represents one of the possible deployments for the services in *Selected*.

5.3 Deployment Optimisation Algorithms

To demonstrate how to adapt the security model to DoFCF, we still use the security model which have been explored in previous chapter. In this section, we investigate and analyse some state-of-the-art optimisation algorithms and then provide some suggestions for adapting these algorithms to our framework for deploying workflow applications over federated clouds.

5.3.1 *Branch and Bound Algorithm*

As discussed above, we need to find a $\lambda \in \Lambda$ which minimises the deployment cost. The most common approach is B&B (branch and bound algorithm) [124]. Generally, this type of algorithm ranks all of the secure deployment solutions and then chooses the cheapest one. However, we have proven that our problem is a NP-hardness problem, and therefore it is very difficult to design an algorithm using B&B to find an optimised deployment solution in polynomial time.

Although this method gives the optimal solution and guarantees that the result is the cheapest deployment, it is not very scalable. In chapter 3, we demonstrated that when the number of services is increased to 12, a version of B&B that we implemented needed approximately 15 minutes to generate a solution. For larger workflow sizes, the execution time grows exponentially. Thus, these types of algorithms are not considered in our framework.

5.3.2 *Genetic Algorithm*

A Genetic Algorithm (GA) can efficiently find a solution to a problem in a large space of candidate solutions [117]. It is a search heuristic that mimics the process of natural selection to find an optimal solution, yet in our case the heuristic function will not constantly produce the optimal (or cheapest) solution. Moreover, the design or method of application of GA can also have a significant impact on the quality of the solution[77].

In the following, we extend and adapt GA to our framework to find an acceptable solution in polynomial time.

5.3.2.1 Security Candidate List

In this chapter, we are aiming to find an optimised solution that meets security requirements while minimising the monetary cost. Therefore, this can be considered as a bi-objective optimisation problem. As mentioned in Section 3.2.1 of chapter 3, each object of the workflow has its security requirements for deploying over clouds. Therefore, we firstly list the satisfied clouds for each object of the workflow in "Candidate List". The pseudocode in Algorithm 4 shows how to create the "Candidate List" for a given workflow by following the security rules.

Algorithm 4 Candidate List

```
$C$ set of Clouds;
$SArray$ –Security Candidate List
for $o$ in $\mathcal{O}$ do
    for $c$ in $C$ do
        if $o$ is secured deployed on $c$ then
            $SArray[o] \leftarrow c$
        end if
    end for
end for
```

The security requirements for each object can be hard constraints (noting it must be met), and the valid clouds that meet these constraints will be maintained in the "Candidate List". Consequently, our problem is reduced to a single objective optimisation which is minimising the monetary cost of the deployment.

5.3.2.2 Elitist Prevention and Diversity Maintenance

The basic GA can be adapted to generate a deployment solution for the problem discussed above. However, to generate an efficient solution, the two primary factors of selection pressure and population diversity have to be considered carefully [147]:

- Selection pressure. A highly effective search must have a search criterion (the fitness function) and a selection pressure that makes the individuals with higher fitness have a higher chance to be selected for reproduction, mutation, and survival.

- Population diversity. Having a variety of individuals to be compared allows the algorithm to fruitfully explore the search space, e.g. if we only explore a subset of the clouds for deploying a given workflow, it will significantly reduce the search space but may miss some efficient solutions.

Unfortunately, these two factors are inversely related. Increasing selection pressure must lead to a loss of population diversity; likewise, maintaining the population diversity will reduce selection pressure. In other words, the GA must be carefully designed to balance the effects on the population of diversity and selection pressure. In order to solve this trade-off, we apply the elitism method[22] summarised in Algorithm 5, to increase the selection pressure, and at the same time control diversity dynamically.

The purpose of the elitist method is to avoid destroying superior individuals in crossover and mutation. Thus, once a solution is confirmed as elitist, it should be directly inherited by the new generation of the population.

Algorithm 5 Elitist Prevent

```
$s$, the size of elitist list; $elist$-list of elitist individuals; $pop$-list of all individuals
if $elist$ is empty then
    ▷ ASCsort sort the $pop$ as ascending order
    ASCsort ($pop$)
    ▷ copy the first s number of solutions to $elist$
    $elist \leftarrow$ from $pop[0]$ to $pop[s-1]$
else
    $pop \leftarrow$ combine $elist$ and $pop$
    ASCsort ($pop$)
    delete $s$ numbers of $pop$ in tail
    $elist \leftarrow$ from $pop[0]$ to $pop[s-1]$
end if
```

Generally, diversity can be measured at three levels: gene-level, chromosome-level, and population-level, with reference to different problems[51]. The challenges, i.e. the large search range and indeterminate chromosome length, a diversity measurement algorithm with high complexity will strongly affect the efficiency of GA. As a result, in our algorithm we applied a simpler diversity measurement at population level.

In contrast, the low diversity of a population usually means that the search reaches a local extreme, which significantly affects the solution. In order to solve this problem, we dynamically control the mutation rate to influence the generation of the new chromosome. Algorithm 6 shows how the diversity protection works.

Firstly, we remove duplications in *pop*, and count the total number (sr) of unique solutions. Based on that we can have density d of the population, which is represented by the ratio of duplicated solutions. Next, the current mutation rate will be increased if the density is greater than the predefined diversity threshold and the current mutation rate is less than the maximum mutation rate. If the current mutation rate is greater than the maximum mutation rate, it will be decreased.

Algorithm 6 Diversity Protection

```
$pop$-a list of all individuals; $size$-size of $pop$; $threshold$-predefined threshold of diversity; $rate$-the current mutation rate; $Minrate$-minimum mutation rate; $Maxrate$-maximum mutation rate.
▷ function $removeDup$ removes the duplications of $pop$
and then copy it to $rpop$
$rpop \leftarrow removeDup(pop)$
$sr \leftarrow$ size of $rpop$
$d \leftarrow 1 - \frac{sr}{size}$
if $d > threshold$ AND $rate < Maxrate$ then
    increase $rate$
else if $rate > Minrate$ then
    decrease $rate$
end if
```

5.3.2.3 Adapting to the Framework

The adaptation of a genetic algorithm (AdaGA) is divided into five phases: coding, generating a candidate list, initialising individuals, selection, crossover and mutation. In general, all possible combinations should be coded into a single string or list, and those codings must represent all solutions. So we coded our deployment solution as a vector $[s_1^i, s_2^j....s_n^k]$, where s_i^j means that service s_i is deployed on cloud c_j. In order to reduce the possibility of generating an insecure solution, we chose the clouds from "Candidate List", assigned them to the corresponding objects, and then coded them as above. However, the application of these operations will not be able to avoid producing a few new coded solutions which may not meet the security requirements.

In such cases, rule SIC is used to verify the security of those solutions. If an insecure code is detected, the algorithm will generate a new one to replace it.

After generating the initialised individuals, selection, crossover and mutation operations are applied on the individuals to produce new generations. During this process, Algorithm 5 is used to prevent elitist individuals, and two methods are applied to perform selection: one is a fitness function that can transfer the fitness of a coding into a numeric representation to select superior solutions. The other is a diversity analysis that does not affect the selection results, but influence the crossover and mutation rate. Both methods are defined as follows:

- Fitness Function: as discussed above, our problem is reduced to finding a solution that minimises the cost by iterating through several generations. Therefore, the Fitness Function can be defined in the same way as in equation 5.4, i.e: $F(X) = COST(\lambda)$.

- Selection strategy: we use roulette wheel selection. The solutions of a population are placed on the roulette wheel based on fitness [72] [73]. The size of each segment in the roulette wheel is inversely proportional to the fitness of each solution. For instance, the probability of selecting solution j for a generation is $P_j = \frac{F(X_j)}{\sum_{i=1}^{M} F(X_i)}$, where M is the population size of the generation to which the individual j belongs. Now we spin the wheel M times to generate M random numbers between 0 and 1.

Crossover is a process of taking part of the features from two chromosomes and combining them to generate a new chromosome. The new child chromosome inherits the features from its parents, so it has a higher possibility of being a good solution. However, the crossover operation occurs with a defined probability P for each pair of chromosomes. The technique that we used in the crossover is one-point crossover (replacing string from position fifth to eighth, as shown in Table 5.1)

After applying crossover, some new chromosomes are produced and the others are directly copied.

In order to enhance the search range, mutation is added to the algorithm. This is implemented by randomly selecting chromosomes in the current generation and changing

$$\begin{array}{lcl}[3,2,3,1,\underline{4,1,2,2}] & \rightarrow & [3,2,3,1,\underline{1,1,3,2}] \\ [1,2,3,2,\underline{1,1,3,2}] & \rightarrow & [1,2,3,2,\underline{4,1,2,2}]\end{array}$$

Table 5.1: example of crossover

them to new secure chromosomes. Further, this operation only happens in very low probability, for example 1%. A few chromosomes of the current generation are selected randomly. The three processes, i.e. selection, crossover and mutation, are repeated until the results reach convergence.

In Section 5.4 the evaluation of our algorithm will be presented and compared with HUGA (Hybrid Utility-based Genetic Algorithm) [89] which was used to optimise the deployment of workflow over a federated cloud, considering QoS requirements. However, we adapt it into our framework.

5.3.3 Greedy Algorithm

The greedy algorithm is an iterative algorithm that incrementally finds better solutions. Unlike the Genetic algorithms that need to finish executing before returning a solution, the greedy algorithm generates a valid and improved solution in each iteration. This is a desirable characteristic for systems where the parameters change frequently and the available time for calculating an improved deployment varies significantly.

Therefore, we employ NCF algorithm [145] which is an extended version of greedy algorithm, to adapt it to our framework. The NCF algorithm has been introduced in the chapter 4, so the details are not described again.

5.3.4 Adaptive Rescheduling Algorithm

In this section, we introduce a heuristic algorithm which has been adapted to the DoFCF framework and dynamically generated a new solution for handling cloud availability changes.

The generic adaptive rescheduling algorithm for handling the change of cloud availability works as follows: when a change in cloud availability is detected, the planner estimates the monetary cost for each service based on the available clouds and information (e.g. execution time) of each service. For our case, the estimation heuristic algorithm can be applied to generate a new deployment solution for the services in Selected over available clouds to meet the security requirements as well as minimising the monetary cost. Therefore, services are distributed based on the new solution and then executed. In addition, the execution is monitored until the workflow has been fully executed. Alongside this, a new deployment must be generated in polynomial time, because the time is accounted for in the makespan of the workflow execution, which may bring extra cost. In extreme cases, if it takes considerable time, the available clouds may change again.

We designed and developed a dynamic rescheduling algorithm to be adapted to our framework, which is summarised in Algorithms 7 and 8. In this algorithm, we assume that a workflow $\mathcal{W}$ is executed with the deployment DEP_0. The "Cloud monitor" is used to monitor the workflow execution status and cloud status. Once the monitor has detected any changes in cloud status, function $REDEPLOYSERVICES$ will be invoked to find the services which require redeployment. These actions are summarised in Algorithm 8. Firstly, the availability of input data for services in set UP is checked, which means that all inputs data must be stored in the available clouds. Otherwise, the elements in set UP will be copied to the *Selected* set and added to the services which have been completely executed but their outputs are not available to *Selected*. Then, the function tracks back and repeats the same action until the available inputs are found. In this case, *Selected* will include the services in UP as well as the added services. Otherwise, *Selected* is equal to UP.

Based on the selected services and the available clouds, a new deployment DEP_1 can be generated by applying a NCF algorithm. If the cloud change is caused by the failure of some clouds, DEP_1 is used to deploy the services in *Selected* directly. Otherwise, if the change is caused by a new cloud becoming available, then the cost of DEP_1 and DEP_0 will be compared (only considering the unfinished services). If DEP_1 is cheaper, the workflow will be scheduled to a new deployment solution, otherwise the execution status and cloud status are monitored until the workflow is completely executed.

Algorithm 7 Dynamic rescheduling

```
UP- set of unprocessed services; C-set of all available clouds; $DEP_0$- deployment
set initial $UP$ = all services of $\mathcal{W}$
while $\mathcal{W}$ not finished do
    ▷ C is updated via the communication with
    $CloudMonitor$
    update C
    ▷ UP is updated via removing the finished services
    update UP
    if cloud status has changed in C then
        Selected=$REDEPLOYSERVICES(UP, C)$
        $DEP_1 = NCF(Selected, C)$
        if clouds have failed in C then
            ▷ submit current execution information to
            generate a new deployment.
            if $DEP_1$ is not found then
                ▷ not valid deployment
                break;
            else
                submit $DEP_1$
                $DEP_0 \leftarrow DEP_1$
            end if
            ▷ new cloud resources become available
        else
            ▷ if the new deployment is cheaper than the
            present one.
            if $DEP_0.cost > DEP_1.cost$ then
                submit $DEP_1$
                $DEP_0 \leftarrow DEP_1$
            else
                do nothing
            end if
        end if
    end if
end while
```

Algorithm 8 Redeploy Services

```
$DEP_0$ the original deployment; Selected-set of services for redeploying; $parents(s_i)$-
set of $s_i$'s parents services
function REDEPLOYSERVICES($UP$,$C$)
    $Selected \leftarrow UP$;
    for $s_i \in UP$ do
        $FINDNODES(parents(s_i), C, Selected)$
    end for
    return $Selected$
end function
function FindNodes($parents(s_i)$,$C$,$Selected$)
    if $parents(s_i) == \emptyset$ then
        return
    else
        for $s_j \in parents(s_i)$ do
            if $s_j \notin UP$ then
                ▷ $DEP_0(s_j)$ indicates the cloud on which
                $s_j$ was deployed
                if $DEP_0(s_j) \in C$ then
                else
                    add $s_j$ to $Selected$ if not included
                    $FINDNODES(parents(s_j), C, Selected)$
                end if
            end if
        end for
    end if
end function
```

5.4 Experiments and Evaluation

To evaluate the performance of our proposed framework, we conducted a series of simulation experiments on a number of real scientific workflow applications. In addition, we applied our framework to dynamically deploy a scientific workflow over a set of e-Science Centrals instances on multiple clouds. The experimental setting, and results are presented in the following sections.

5.4.1 Cloud Service Broker

For the purpose of evaluation, we developed a Cloud Service Broker to identify suitable cloud service providers. The Broker performs cloud exchange and negotiates with each available cloud to allocate resources that meet the users' specific requirements. Figure 5.1 shows the architecture of the Cloud Service Broker for workflow application along with other components as illustrated below:

The *Client* can be a platform for workflow management, such as e-Science Central or Pegasus[46], which allows users to describe and submit their workflow application through platform components. The **Workflow Engine** delivers the workflow tasks (or services in this chapter) to the underlying Cloud Service Broker, including execution requirements, task descriptions, and the desired security requirements.

*T*he Cloud Service Broker enables the functions of resource allocation, workflow scheduling and software deployment. Our framework includes a **Planner** component that performs a matching process to select the target clouds for deployment, based on the information passed from **Global Cloud Market**. Further, the workflow tasks are assigned by the **Scheduler**, and the **Data Manager** maintains the data transfer during workflow execution. The planned tasks are distributed to the underlying cloud providers via **Deployment APIs**. These APIs can also be used to interact with the underlying clouds to monitor workflow execution and cloud availability.

A *Federated cloud* is a cloud resource pool that provisions computation and storage resources, as well as specific non-functional capabilities (referring to different security levels in this chapter) and functional capabilities such as the execution environment of each task.

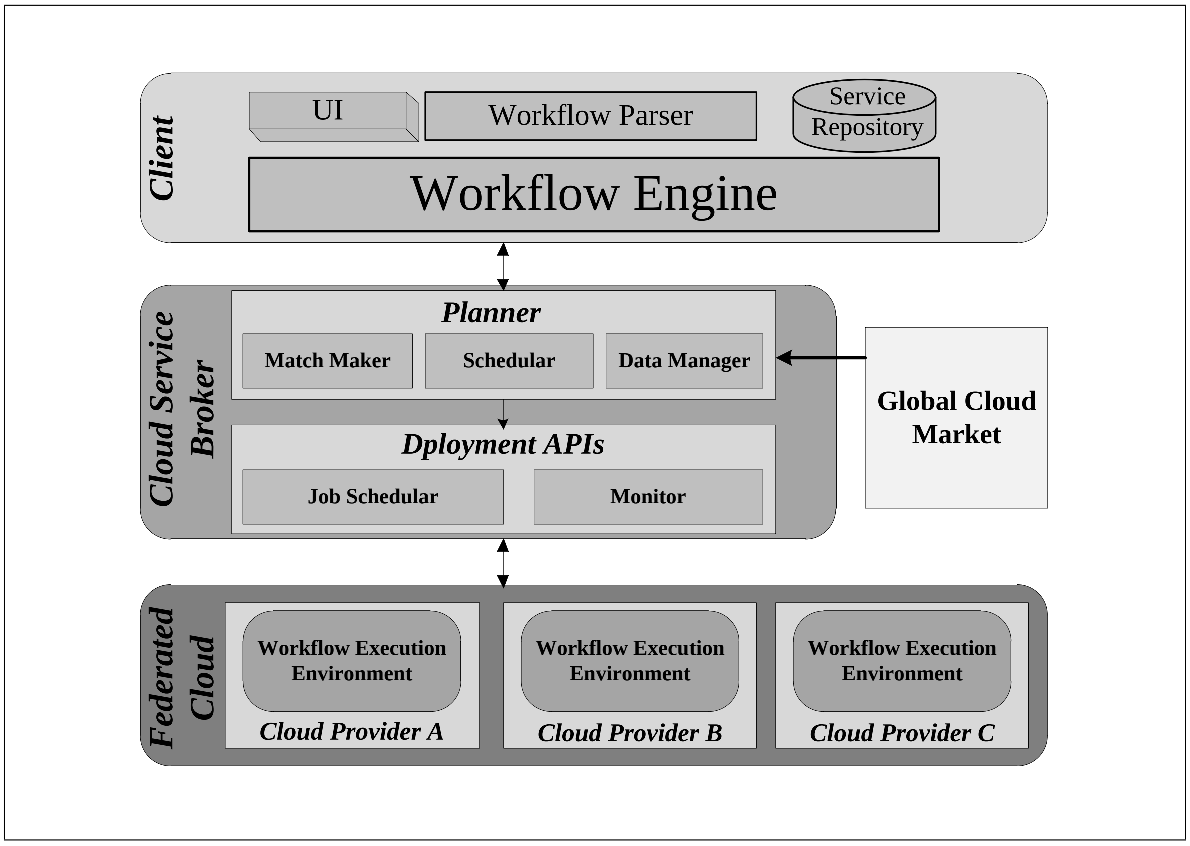


Figure 5.1: Architecture of Cloud Service Broker for Scientific Workflow

5.4.2 Simulation Environment

In this work, the experiments must be repeatable in order to easily compare and analyse different types of algorithms. Therefore, to ensure repeatability, we evaluated our framework by using CloudSim to investigate the deployment and scheduling of workflows in federated cloud environments.

5.4.2.1 Experiment Setup

In the evaluation of our framework, we consider four common types of workflow applications: CyberShake (earthquake risk characterisation), Montage (generation of sky mosaics), LIGO (detection of gravitational waves) and Epigenomics (bioinformatics).[1] The full characterisation of these workflow applications can be found eslewhere [90], however, we only consider the execution time, and the input and output data of each service. Table 5.2 lists the four workflow types with different numbers of tasks:

[1]The XML description files of the workflows are available via the Pegasus project: https://confluence.pegasus.isi.edu/display/pegasus/ WorkflowGenerator

Workflow	Medium	Large	Very large
CyberShake	30	100	1000
Montage	25	100	1000
LIGO	30	100	1000
Epigenomics	24	100	995

Table 5.2: Number of Tasks of each Workflow for Each of the Three Scales

Type	Location	Exec (/hour)	Store (/hour/GB)	In (/GB)	Out (/GB)
C1	0	0.40	0.10	0	0.02
C2	2	2.20	0.60	0.03	0.01
C3	1	1.23	0.30	0.14	0.07
C4	2	3.70	0.60	0.10	0.05
C5	3	4.50	0.90	0.14	0.05
C6	4	5.5	1.30	0.14	0.13

Table 5.3: Cloud Pricing and Security Levels

medium, large and very large.

The data privacy information for the workflows is unavailable to be used for assigning the security levels of each object. Therefore, we randomly generated the security levels for each object in these workflows.

Six virtual machines (VMs) have been created, representing workflow execution environments, in six different data centres to represent six types of cloud (with different security levels). Additionally, each VM can run several services at the same time. Table 5.3 details the numbers of clouds with location security levels, computation cost, storage cost and communications cost, where In and Out represent the cost of incoming and outgoing data respectively.

The experiment results, presented below, show the average values of observing 1000 executions of each algorithm for each type of workflow. For each of the 1000 repetitions, the same random number generation seeds for each execution, which guarantees each algorithm is exactly running over the same infrastructure. The observed outputs metrics are execution time normalised with the corresponding value obtained from the NCF algorithm.

5.4.2.2 Monetary Cost Evaluation

As discussed earlier, the total cost includes execution cost, running time storage cost and communication cost. The pricing of each cloud listed in Table 5.3 shows that a more secure cloud is generally more expensive. Our cost calculation does not consider additional costs like license and VM image costs which are charged by some cloud providers.

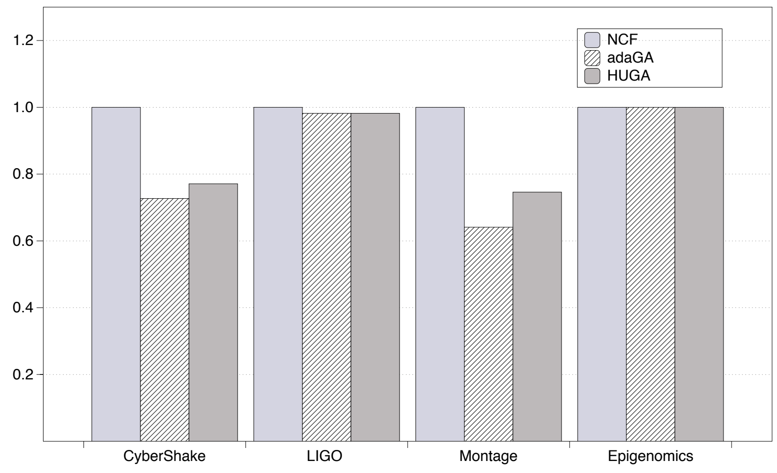
Figures 5.2a, 5.2b and 5.2c depict the normalised cost for the four types and three sizes of workflow applications mentioned previously using three algorithms: NCF, HUGA, and adaGA. Each figure represents cost calculations for a specific workflow size to show the variations in cost according to size.

The results show that the algorithm AdaGA can always generate the cheapest deployment solution. For example, in a case with medium size of the "Montage" workflow, the solution generated by AdaGA can save up with 35% compared to the NCF solution.

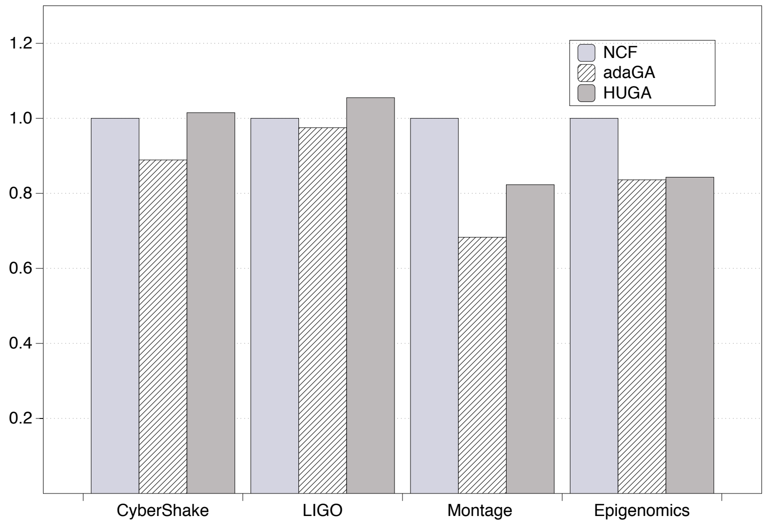
The types of workflow significantly affect the solutions generated by NCF. The cost of deployment solutions which are generated by the three algorithms, applying to the medium size of LIGO and Epigenomics workflow (see Figure 5.2a) very close. However, for the other two types of workflow, the solutions generated by NCF are much more costly than those of the AdaGA. Furthermore, the differences are reduced with the increase in workflow size. This is because NCF is not influenced by the search space (larger workflow indicates more deployment solutions). In addition, the search space significantly affects the results generated by AdaGA and HUGA. However, the Elitist Prevent and Diversity Protection methods were used in AdaGA to avoid the algorithm visiting less desirable solutions.

5.4.2.3 Time Complexity Evaluation

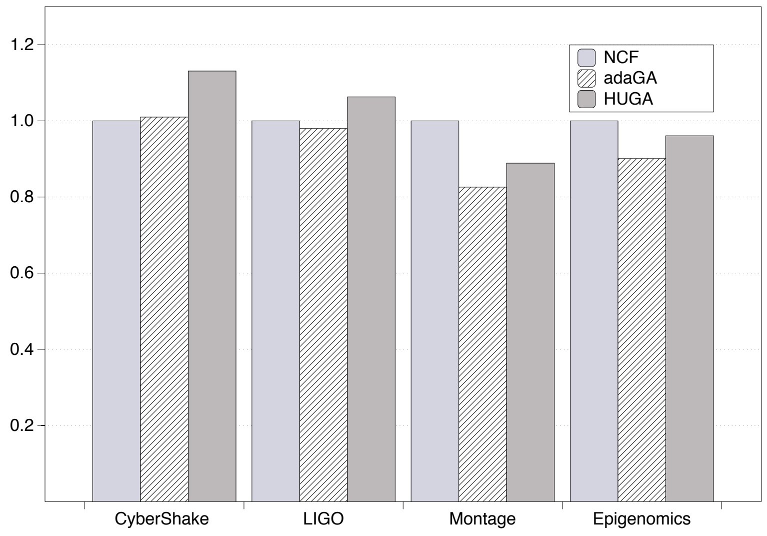
In order to evaluate the time complexity of each algorithm, we measured the time consumed by each to find the optimised deployment for the four types of workflow on the given clouds. According to our evaluation, Figure 5.3 shows that algorithm NCF is significantly faster than the other algorithms. Further, AdaGA has better performance than HUGA with medium size workflows. The reason for this is that the



(a) Cost for Medium Size Workflows



(b) Cost for Large Size Workflows



(c) Cost for Very Large Size Workflows

Figure 5.2: Cost for Different Workflows

search will be terminated if no better solution has been found after repeating the pre-defined generations. Nevertheless, as workflow size increases, AdaGA consumes more time than HUGA to find the deployment solution. However, the deployment solution for a very large size of the "CyberShake" workflow can be generated by AdaGA in less than one minute. Consequently, by considering cost savings, AdaGA can be the better choice.

Figure 5.3 also indicates that the time complexity of each algorithm not only depends on the number of o (data and services) and the available clouds, but also on the structure of the target workflows and the security levels for each o and cloud.

The time complexity of each algorithm is formalised as follows:

HUGA algorithm can be split into three phases in order to analyse the time complexity: selection, crossover and mutation. Therefore, the time complexity of the selection phase is $O(|P| \times |G| \times |\mathcal{O}|)$ where P is the size of population and G is the generations numbers7-8. For crossover and mutation phases, we need to operate on the whole chromosome, and so the complexity of both is $O(|P| \times |G|)$. Thus the time complexity of HUGA is $O(|P| \times |G| \times |\mathcal{O}|)$.

Comparing *AdaGA* with *HuGA*, AdaGA contains an additional Elitist phase for each generation which slightly increases the time complexity. In each Elitist phase, the algorithm observes the chromosomes for each population and saves the best one. Thus, the complexity of this phase is $O(|P| \times |G| \times |\mathcal{O}|)$ and the time complexity of *AdaGa* is $O(|P| \times |G| \times |\mathcal{O}|)$ as well.

In the *NCF* algorithm, we have already analysed the time complexity in previous chapter. It is significantly impacted by the structure of the workflow. If the workflow is linear, the complexity in the worst case becomes $O(|\mathcal{O}| \times |C|)$. Conversely, for a star-shaped workflow, the best case complexity is $O(|D| \times |C|)$.

5.4.2.4 Cloud Availability Change Evaluation

In this part of the experiments, we simulated the change in cloud availability by pre-defining the times when each cloud was available. To do this we set the start time and termination time for each cloud before starting the simulation. A *Cloud Monitor* was

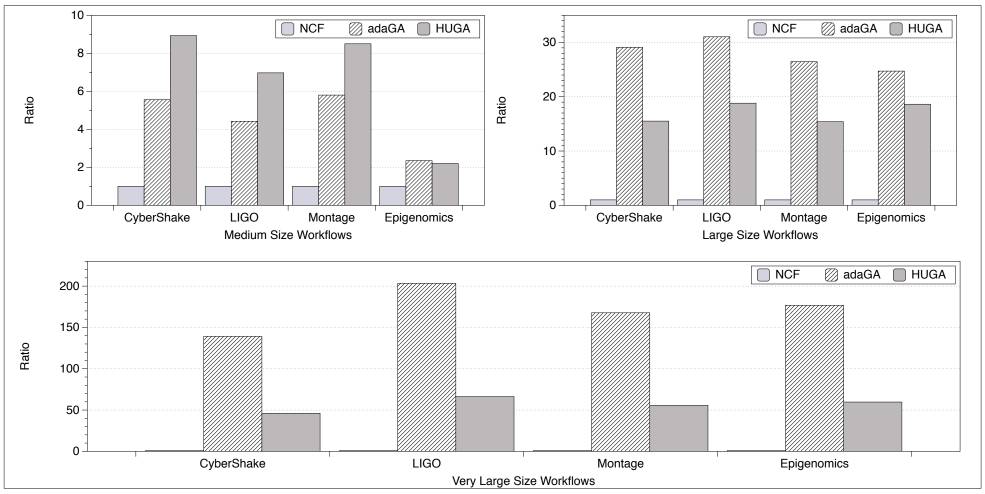


Figure 5.3: The execution of each algorithm

implemented to monitor cloud status, i.e. detect changes in cloud environments. If a changed status is detected, a notification will sent to the *broker*. Thus, the broker can reschedule the running workflow to the available clouds based on the cost of the new deployment. This was evaluated for two types of change: *Clouds fail* and *Availability of new clouds*.

Cloud fail was simulated by setting the terminal time for the randomly selected clouds as uniformly distributed between 0 and the *makespan*. In another words, during the workflow execution, the number of cloud fail is randomly set from 0 to 6. Furthermore, we performed 1000 simulations, each with different cloud fail settings, and recorded each execution statue, including how many clouds fail, the makespan, completion of workflow execution. Consequently, we can have the average of the cost and the makespan based on the recording.

In this evaluation, we used "Epigenomic " workflow with medium, large and very large size. As an example of the setting for the medium workflow size, we randomly selected the clouds and set their termination time as uniformly distributed over $[0, makespan]$ where the *makespan* is the execution time of the selected workflow running as the deployment generated by NCF. For the clouds that were not selected, the terminate time was set as infinite.

Thus, we can have three types of executions: (1) Success: in this type, the workflow is completely executed without rescheduling (noted as Success in Table 5.4). The reason is that the selected clouds finished executing the assigned tasks before their terminated time, therefore failures do not affect task execution. (2) SBR: a type in which the workflow is successfully executed by rescheduling to the currently available clouds after a cloud has become unavailable. (3) Fail: the workflow execution cannot be completed as no alternative deployments were available for it after a cloud failure – this could be because of cloud unavailability or because the available clouds do not meet the security requirements.

Table 5.4 depicts the results of cloud failure. These demonstrate that the failures occur more frequently with the increasing sizes of workflow and execution time. We normalise the results by using the ratio of value of the outputs of the SBR executions with that from the Success executions, recording $SBR/Success$. In the case of very large workflow, we only have to pay an extra 2% money and 33% time to avoid re-running the workflow from the beginning when a failure happens during workflow execution.

Workflows	Execution status (%)	Cost	Time
Medium	Success:74	1	1
	SBR:16	1.14	2.36
	Fail:10	none	none
Large	Success: 70	1	1
	SBR: 20	1.1	1.15
	Fail:10	none	none
Very Large	Success: 54	1	1
	SBR: 22	1.02	1.33
	Fail: 24	none	none

Table 5.4: Experiment results for cloud failure

To simulate *Availability of new clouds*, the workflow has to be already running over a set of clouds with a deployment. Thus, we pre-set clouds C6 and C5 as available for the initial deployment. Furthermore, we randomly selected clouds and set the start time of them as uniformly distributed over $[0, \infty]$. We generated 1000 settings and repeated the executions of the rescheduled workflow to new clouds based on the monetary cost savings. This resulted in the execution types shown in Table 5.5: (1) *Success* represents no new clouds becoming available or the new clouds not offering cheaper deployment than the currently running one. (2) *SBR* denotes the running workflow can be rescheduled to the new available clouds to save execution costs.

Table 5.5 shows that the fluctuations of the saving cost are very significant, depending on the types of the workflow and available clouds. (In Table 5.5, we used the same normalisation rule as Table 5.4). Therefore, when new clouds become available, users should participate in making the decision to use a new deployment based on the estimated cost and makespan saving.

Moreover, Table 5.5 shows that the makespan increases , while the monetary cost goes down. The reason is that when a running task is shifted to a new cloud which is cheaper, the running task has to be killed, and then be redeployed and re-executed in the new cloud. If the new cloud is faster than the current host cloud, the makespan might be reduced. However, in this work we do not consider the performance of the cloudsïijŇand assume that clouds' performance are the same.

Workflows	Execution status (%)	Cost	Time
Medium	Success:92 SBR:8	1 0.98	1 1.46
Large	Success: 88 SBR: 12	1 0.90	1 1.21
Very Large	Success: 70 SBR: 30	1 0.99	1 1.33

Table 5.5: Experimental results for new clouds becoming available

5.4.3 Realistic System Evaluation

To evaluate our algorithm in conditions closer to a real-world scenorio use, we applied it to schedule scientific workflows in e-Science Central(e-SC). We used the e-SC APIs to create a Cloud Services Broker that can orchestrate invocations of a single workflow partitioned over a number of e-SC instances.

5.4.3.1 Design and Setup

According to the architecture of the cloud services broker, shown in Figure 5.1, our tool consists of three components: Client, Cloud Services Broker and Federated Cloud.

The *Client* includes a user interface (UI) which allows users to create workflows for the e-Science Central workflow engine. The description of the created workflow can then be passed to the Broker.

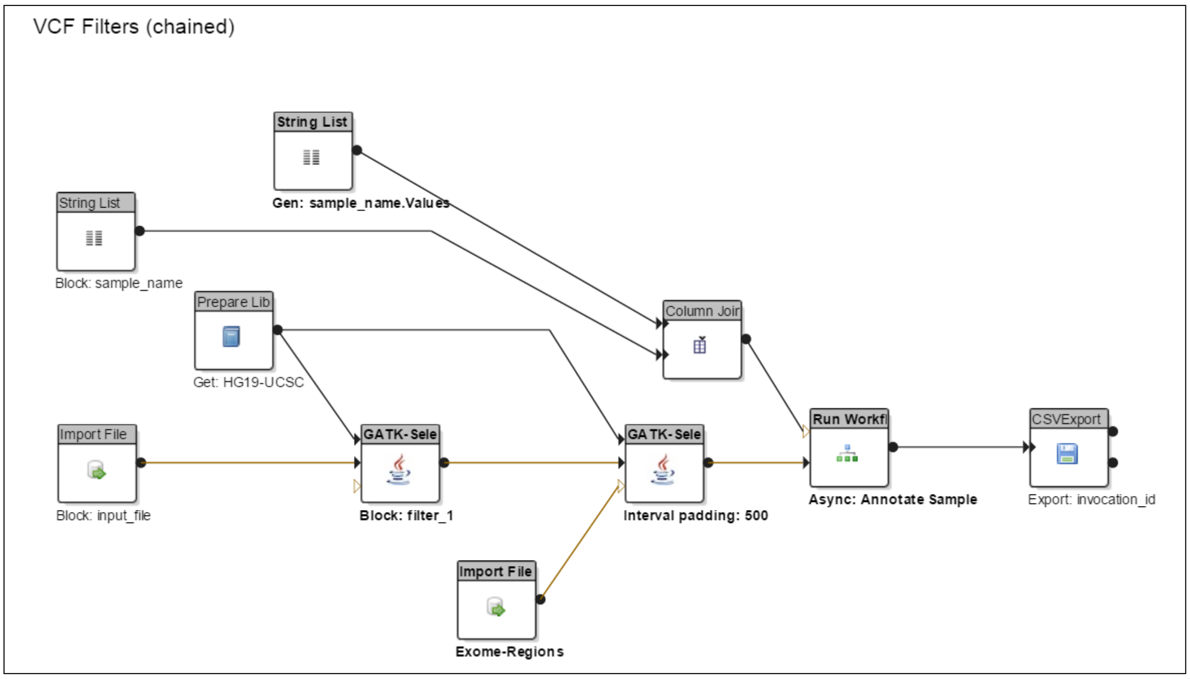


Figure 5.4: A selected workflow from the cloud e-Genome project

Cloud Services Broker is the core part of our tool and includes a planner to assign the workflow to a federated cloud using the algorithms discussed earlier. e-SC APIs are used to dispatch tasks to corresponding clouds and monitor the execution. A Failure Generator is used to simulate failures by turning on or shutting down e-SC instances.

Federated Cloud is a set of e-SC instances which can interact with the broker and other e-SC instances through e-SC APIs, and process the tasks which are scheduled.

To evaluate our tool we selected one of the workflows used in the cloud e-Genome project [31] (as shown in Figure 5.4).

The workflow was implemented to process an exome sequenced by using e-SC deployed on Microsoft Azure cloud.

While in the e-Genome project security aspects are not a primary concern, guaranteeing that human genomic data can be securely processed on the cloud is very important. Therefore, we modelled the security requirements of the selected e-Genome workflow by assigning security levels as shown in Tables 5.6 and 5.7. Note that the data size transferred among blocks and the execution time of each block are real values taken from logs collected by e-SC. Table 5.6 shows data sizes in GB, where 0 denotes less than 1 MB of data. The pricing of Clouds C1, C2 and C3 in Table 5.3 was applied to

Service	Name	Clearance	Location	Time(/h)
Sample_Name	S1	1	0	1
ImportFile	S2	1	0	1.5
Sample_Vaule Info	S3	1	0	3
*HG*19	S4	1	0	0.1
Filter	S5	2	0	10
Exome − Regions	S6	1	0	7
Intervalpadding	S7	0	0	20
ColumnJoin	S8	2	0	0.1
AnnotateSample	S9	2	0	5
Export	S10	1	0	0.3

Table 5.6: Services representation and security and execution time

Data	Location	Size (GB)
$S_{1,8}$	1	0
$S_{2,5}$	0	1.1
$S_{3,8}$	2	0.01
$S_{4,5}$	0	0.005
$S_{4,7}$	0	0.005
$S_{5,7}$	0	6.2
$S_{6,7}$	0	10.3
$S_{7,9}$	1	3.6
$S_{8,9}$	0	0
$S_{9,10}$	0	0.05

Table 5.7: Data security

calculate the deployment cost.

To simulate this environment we set up three virtual machines, each running a single instance of e-SC system. VM1 was hosted on a personal PC and represented the private cloud. Two other VMs were hosted in our University virtualised environment and played the role of public cloud providers C2 and C3.

Our evaluations included three steps: the first step tests the static deployment algorithm. We kept all three e-SC instance running and applied adaGA to make the deployment plan. The second step shows how to handle a cloud failing, by shutting down one of the running e-SC instances when the workflow was running. The setting of this step is similar to that of CloudSim. Finally, we tested the availability of a new cloud by deploying the given workflow on two clouds, and then turning on a new instance which offers price advantage. Also, for the purpose of the experiment we reduced the execution time of the given workflow to about 30 seconds by scaling down the amount of input data shown in Table 5.7 by a factor of 6000.

5.4.3.2 Results and Analysis

Based on the presented experiment setup, all of the three steps of the deployments are illustrated in Table 5.8. Precisely, “Static” refers to first step, and “Cloud fail” and “New Cloud” correspond to steps two and three respectively.

Static, shown in Table 5.8, represents the cheapest solution which was generated using the adaGA algorithm by deploying the workflow over cloud C1, C2 and C3 to meet the security requirements. Services S_1 S_7 and S_8 were deployed on C2 and others were allocated on C1.

For the Cloud Fail, we used the deployment of *Static* as the initial deployment (INI). However, cloud C1 failed (shown as blue in Table 5.8) when service S_9 was ready to execute.

The available clouds are C2 and C3, and the inputs of S_9 are stored in C2 (the outputs of S_7 and S_8), therefore, S_9 can be rescheduled to C3 to continue the execution (indicated in "Cloud fail", SBR). If S_7 is deployed on C1 with the same failure, S_2, S_4, S_5, S_6 and S_7 should be re-executed in C3. Thus S_9 and S_{10} can be completed in C3.

In the third step, C2 and C3 were available for the initial deployment (see Table 5.8 New Cloud, INI). After the workflow was executed for one second , C1 became available. S_1 and S_4 were completely executed on C2 and C3 respectively, but S_2, S_6 and S_3 were still running. Based on the status information, a cheaper deployment solution (see New Cloud SBR in Table 5.8) became available, which required termination of S_2 and S_6, and then re-running them on C1, as shown in green in Table 5.8.

Service	Static	Cloud Fail		New Cloud	
		INI	SBR	INI	SBR
S_1	C2	C2	C2	C2	C2
S_2	C1	C1	C1	C3	C1
S_3	C2	C2	C2	C2	C2
S_4	C1	C1	C1	C3	C3
S_5	C1	C1	C1	C3	C1
S_6	C1	C1	C1	C3	C1
S_7	C2	C2	C2	C2	C2
S_8	C2	C2	C2	C2	C2
S_9	C1	C1	C3	C3	C1
S_{10}	C1	C1	C3	C3	C1

Table 5.8: Two deployments

Table 5.9 shows average values of the cost and makespan of each deployment by repeating the executions 10 times. Where SBRF represents the situation of handling C1 fail (see Table 5.8 Cloud Fail SBR). Similarly, ININ and SBRN are the experimental results of new cloud available. Where the makespan of SBRN is approximate one second more than others (it will take one hour more by using the original inputs). It because of the C1 has to re-execute S_2 and S_6 from the beginning.

Deployment	Time (seconds)	Cost
Static	28.93	64.44
SBRF	28.94	67.23
ININ	28.92	84.62
SBRN	29.90	65.17

Table 5.9: The cost of different deployments

5.5 Conclusion

In this chapter, we have presented a framework to improve the security of workflow applications while they are distributed on a federated cloud. The cost model was designed to optimise the cost of each deployment option. Furthermore, we added a unique dynamic rescheduling method to handle changes in cloud availability to our framework. This supports execution continuing on other clouds when a cloud fail, and can save the cost when new clouds become available. Additionally, we designed and implemented three algorithms for static deployment planning. These algorithms have been applied to different types of workflows to demonstrate different cases, and then their performance was discussed and analysed. We evaluated the performance of our framework by conducting a series of experiments on various types of real scientific workflows. The experiments have been performed using a simulation environment as well as on a real workflow management system. The results show that our framework is suitable for deploying and handling the failures of universal scientific workflows over federated clouds. Moreover, we provide some suggestions for deployment optimisation through analysing state-of-the-art optimisation algorithms. Reliability is a key consideration for users moving their applications to the cloud. Although our framework can continue workflow execution when clouds fail, extra cost for rescheduling is incurred in terms of both time and money. This has been discussed with reference to the experimental results. Therefore, in the next chapter a method is proposed which can improve the reliability of executing scientific workflows over a federated clouds while guaranteeing security requirements are met and also the monetary costs are minimised.

6

Cost Effective, Reliable and Secure Workflow Deployment over Federated Clouds

Contents

Summary

In the previous chapter, we proposed a framework for workflow deployment over a federated cloud, while meeting the security requirements and minimising the monetary cost. However, cloud federation offers applications a variety of clouds, which differ in terms of reliability, as well as security. This chapter presents a novel algorithm to deploy workflow applications on federated clouds. Firstly, we introduce an entropy-based method to quantify the most reliable workflow deployments. Secondly, we apply an extension of the Bell-LaPadula Multi-Level security model to address application security requirements. Finally, we optimise deployment in terms of its entropy and also its monetary cost, taking into account the cost of computing power, data storage and inter-cloud communication.

6.1 Introduction

Previous chapters have focussed on exploiting clouds that differ in security and price, allowing applications to optimise their deployment over a set of clouds, both initially, and after a failure. In this chapter we focus further on reliability.

Despite the fact that the SLA (service-level agreement) for clouds are usually at the level of 99.95% availability for the compute service, cloud providers are not always able to achieve this. In practice, this level of availability means that the service can only be offline for about 20 minutes in a month, or for only about 250 minutes per year. However, in early 2011 several high-profile technical companies were landed in trouble when Amazon's EC2 service suffered an outage [2] which lasted for almost 11 hours. In a month this gives only 98.47% of availability and in a year only 99.87%. Similarly, an outage at the GoDaddy cloud provider took down millions of web sites.[1]

With all the variety and uncertainty involved, application developers who decide to host their systems in the cloud face the issue of which cloud to choose to get the best operational conditions in terms of price, reliability and security. And the decision becomes even more complicated if their application consists of a number of distributed

[1] K. Finley, "Godaddy outage takes down millions of sites, anonymous member claims responsibility," http://techcrunch.com/2012 /09/10/godaddy-outage-takes-down-millions-of-sites.

components, each with slightly different requirements. In such contexts, a combination of different cloud providers might be best.

As has been described, cloud federation offers the ability to distribute a single application on two or more cloud platforms, so that the application can benefit from the advantages of each [116]. The key challenge in attempting to exploit this opportunity is how to find the best distribution (or deployment) of application components, which can yield the greatest benefits.

Previous research has addressed scheduling with other objectives such as security, reliability and performance. The authors of paper [52] minimise the makespan and failure probability by combining both objectives into a single cost function to form a new matching and scheduling algorithm. In paper[54], the authors distinguish the reliability maximisation and execution time minimisation mappings and then provide a method for converting workflow scheduling heuristics on heterogeneous clusters into heuristics that take reliability into account. The work in [55] presents two algorithms to address the trade-off between makespan and reliability. One is based on a dynamic level scheduling algorithm and the other is a version of a genetic algorithm. A framework was introduced in [58] has been demonstrated to address four objectives: makespan, economic cost, energy consumption and reliability. These works consider reliability and multi-objective optimisation, especially [58] which was related to single cloud. However, our work uses a model-based technique on federated clouds where the monetary cost also plays a crucial role, and none of the previously mentioned algorithms have been considered of this.

In this chapter we tackle this problem and propose an algorithm to deploy workflow-based applications over federated clouds in order to exploit the strengths of each cloud. Our algorithm schedules an application structured as a workflow such that requirements for each block (services and data) against security and reliability are met whilst the cost of execution is minimised.

The algorithm considers the security model we have discussed in chapter 3, and adapts it to the new multi-criteria requirements. The security model is based on the Bell-LaPadula [21] Multi-Level Security model [36] and was designed to partition a workflow over a federated cloud to meet certain security requirements. This security model is used to address confidentiality, i.e. the absence of unauthorised disclosure of information [17].

The basis for the algorithm is a new method to quantify the most reliable workflow deployments which applies Shannon's information theory [128]. Using reliability information for each application component and the underlying cloud platform, the method calculates entropy of a workflow deployment. This value is then used as a constraint in the optimisation problem. We argue that using entropy can reduce the overall risk of workflow failures caused by a small number of components being deployed on less reliable clouds. Reliability is understood to mean the readiness for correct service; for example, the amount of time a device or service is actually operating as the percentage of total time it should be operating.

Furthermore, as there may be a number of deployments that meet our security and reliability requirements, we search for the option that minimises the monetary cost. Finding the optimal deployment is, however, an NP-hard problem [65], thus we need an approximate algorithm to solve it. The two most common approaches are: (1) to linearise the problem by assigning weights to the criteria and then optimise the weighted sum [96], (2) to optimise one criterion and keep the others constrained within predefined thresholds. In the first approach the difficulty is not only in defining the weights properly but also in the limitation of the simple, linear model which may not be able to accurately represent the complexity of the problem. Hence in this work we use the second approach.

To handle this multi-criteria and NP-hard problem we generate a valid initial solution and then apply a set of refinement methods to approach the optimum. At the same time we want to guarantee that the time complexity of the algorithm is polynomial.

The rest of the chapter is structured as follows. We first introduce the notation and models used to represent the reliability and security requirements and to calculate the monetary cost of deployment. Next, in Section III we show the optimisation problem as described by the models. Then a scheduling algorithm to search efficiently for a suitable deployment option is presented. In Section V we discuss the evaluation of our work. Finally, future work is outlined and conclusions are drawn.

6.2 Problem Description

With the increasing availability of public and private cloud resources it is easy to deploy instances of the same service in multiple places. We have observed this tendency with our e-Science Central data analysis system [81] which, depending on the use case, has been deployed in a variety of locations including private clouds at universities in Spain and Brazil and public cloud resources such as Amazon AWS and Microsoft Azure. Each of these clouds has its own advantages and thus our focus in this chapter is on how a single workflow application might be deployed over a federated cloud. By a federated cloud we consider in this chapter a set of workflow execution environments (such a e-Science Central) running in different clouds, which we can manage and use to run applications. Our goal is to partition a workflow application in such a way that it can benefit from the "best" combination of these environments.

Below we introduce the notation used through the rest of the chapter and define the three concepts that form the basis of our algorithm: the measure of reliability, the security rules and the cost model.

6.2.1 Reliability

The target clouds consist of a set of PMs (physical machines), and can be geographically distributed. A single physical machine is able to contain a set of VMs (virtual machines). Each VM can run 0 or more instances of WP (workflow execution platform). Lastly, a WP can run a number of services/workflow tasks (to put it simply, we use service in the rest of chapter) concurrently. We are not considering fault tolerance [159] as a means for improving reliability in this chapter.

6.2.1.1 Reliability of Computing

By reliability we mean the readiness for correct service, for example, the amount of time a device or service is actually operating as the percentage of total time it should be operating. Therefore, reliability can be defined as: $REL = \frac{\text{Fault-Free Time}}{\text{Total Time}}$.

We assume that a service itself is implemented fault-free [125] and it is executed completely and reliably on a workflow execution platform, iff the physical machine, VM and WP which the service is running on are fault-free during the service execution time.

Let t be a random variable which represents a point in time when a failure happens, while f denotes the probability density function of t. We assume that failures are occurring randomly in time and can be modelled by an exponential distribution. Therefore, the exponential probability density function of WP is $f_{wp}(t) = \lambda_{wp} e^{-\lambda_{wp} t}$, where λ_{wp} is the failure rate of workflow platform wp. Then, following [125], we can define the reliability function of wp as:

$$R_{wp}(t) = 1 - \int_0^t f_{wp}(t)dt = e^{-\lambda_{wp} t} \tag{6.1}$$

We assume that the execution time of service s_i is t_i and T_{wp_i} is the time since wp_i is available but just before s_i is launched. Thus we can have the following corollary.

Corollary. The reliability of execution of s_i in wp_i is $R_i(t_i) = R_{wp_i}(t_i + T_{wp_i})$.

Proof. From the discussion above, reliability $R_{vm_i}(t)$ of vm_i at time t depends on the reliability of the VM instance itself, $R'_{vm_i}(t)$, and the reliability of its host machine $R_{pm_i}(t+T_{pm_i})$, where T_{pm_i} is the time since pm_i has been available but just before vm_i is started. We can denote that as: $R_{vm_i}(t) = R'_{vm_i}(t) \cdot R_{pm_i}(t + T_{pm_i})$. Similarly, the reliability of wp_i is $R_{wp_i}(t) = R'_{wp_i}(t) \cdot R_{vm_i}(t+T_{vm_i})$, where T_{vm_i} is the time since vm_i has been available but just before wp_i is started. Consequently, we can state that the reliability of running service s_i in the Cloud is $R_i(t) = R'_i(t) \cdot R_{wp_i}(t + T_{wp_i})$. However, earlier we assume that service s_i is implemented fault-free and so $R_i(t) = R_{wp_i}(t+T_{wp_i})$.

6.2.1.2 Measure of Workflow Reliability

We assume that workflow w consists of n services $s_1, \ldots, s_n$, and the reliability of each service is R_i. Moreover, the communication between related WPs and services are fault-free. Therefore, the possibility that w is failure-free is:

$$R_P(\phi) = \prod_{i=1}^{n} R_i = e^{\sum_{i=1}^{n} -\lambda_{wp_i}(t_i+T_{wp_i})} \tag{6.2}$$

where ϕ is a deployment of w, λ_{wp_i} is the failure rate of the workflow platform which is used to deploy s_i (we assume that λ_{wp_i} already includes the failure rate of the underlying layers), and $t_i + T_{wp_i}$ is the execution time of the WP since it was started and until s_i finished. R_P (power method thereafter) is the most common way to measure the reliability of the execution of a workflow application [78] and [52].

However, in this work we propose an *entropy-based* method to measure workflow reliability. Entropy [128] is a widely used approach to capture the degree of dispersal or concentration of values of a random variable. For a discrete random variable X with probabilities $p(x_i)$ it is defined as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \tag{6.3}$$

We deem $p(x_i)$ as reliability of service s_i and use the entropy value as a measure of workflow reliability. Thus, the reliability of workflow w can be calculated as:

$$R_E(\phi) = -\sum_{i=1}^{n} R_i \log R_i \tag{6.4}$$

In Appendix A, we illustrate that entropy method has advantages over the traditional power method. Furthermore, we propose a constraint on the entropy value which can not only guarantee the reliability of a workflow deployment, but also reduces the risk that a workflow includes a service with relatively low reliability. If we assume that the required reliability of deploying workflow w is $\mathcal{R}$, then using the power method we need $R_P(\phi) \geq \mathcal{R}$. However, the corresponding entropy constraint is: $R_E(\phi) \leq -R_M \log \mathcal{R}$, where $R_M = MAX(R_i)$ is the maximum reliability of the workflow services.

6.2.2 Security Model

A security model is needed to determine whether a deployment of services and data to a set of clouds meets the organisation's security requirements. In this chapter we apply the same security model in chapter 3 as an hard constraint.

6.3 The Challenges

6.3.1 *Multi-Objective Optimisation Problem*

For a given ϕ which is a secure deployment of workflow w over federated clouds WP, we propose to optimise two parameters: 1) minimise the value of entropy $H(\phi)$ which results in a distribution that maximises the reliability of the workflow, 2) minimise the value of $COST(\phi)$ to obtain deployments with low cost of execution. We express this problem as:

$$\begin{aligned}
&\textbf{minimise}\,(COST(\phi),(H(\phi))\\
&\textbf{subject to}\,\exists\,\phi\in\sum(\Phi,W)\\
&\qquad\qquad\Phi = D\times S\times WP\times L
\end{aligned}$$

Finding a solution which optimises both $COST(\phi)$ and $H(\phi)$ is challenging. Note that if we simplify our problem to only one objective optimisation, e.g. finding deployment ϕ which minimises cost, it can be directly mapped to the *skewed* graph partitioning [80] which is a variant of classic NP-complete graph partitioning [65].

In the skewed graph partitioning each vertex i has a desire to be in set k denoted by $d_k(i)$. Given that, the problem tries to minimise the cut edges (flow of data between sets) and maximise the desires. If $w(e_{ij})$ is the weight associated to the edge between vertices i and j and $s(i)$ is the set to which the vertex i is assigned, the problem tries to find mapping s which minimises objective function:

$$\textbf{minimise}\sum_{e_{ij}}\begin{cases} w(e_{ij}), & \text{if } s(i)\neq s(j)\\ 0, & \text{otherwise}\end{cases} - \sum_{i=1}^{n} d_{s(i)}(i)$$

In our case vertices denote workflow services, weights associated with edges represent the cost of data transfer between services, set $s(i) = wp_i$ is the workflow platform assigned to execute service s_i and, finally, desire $d_{s(i)}(i)$ is the negative value of the execution and data storage cost associated with running service s_i on platform wp_i.

6.3.2 Trade-off Problem

To have a better understanding of the trade-off between entropy and the monetary cost of workflow deployments we ran two sets of experiments. First we used the NCF (Not Cheapest First) algorithm from our previous work [145] to optimise deployment by cost. NCF is an approximate greedy algorithm which can quickly find deployments of workflows with security constraints. Then we implemented another greedy algorithm to optimise deployment by its entropy value. Finally, we ran these two algorithms to deploy 300 different randomly generated workflows which included from 2 to 30 services. Figure 6.1 shows the result.

Triangles represent the deployments which minimise the monetary cost, whereas circles are deployments with minimal entropy. Colours are related to axes with blue showing the monetary cost of a deployment and red showing its entropy. Clearly, the cost optimised deployments are cheaper than the entropy optimal ones. However, they have also greater entropy value. Conversely, the entropy optimal deployments have lower entropy but result in more expensive deployments. Also, as may be seen, the gap between cost optimal and entropy optimal deployments increases with the increasing number of services in a workflow, which clearly indicates the trade-off between these values.

6.4 Scheduling Algorithm

The scheduling algorithm we propose can optimise the deployment of a workflow over a set of clouds. It takes into account user requirements against multiple criteria, namely security, reliability and cost. The optimisation part of the algorithm is an extension of the multiple-choice knapsack problem (MCKP) [93].

Overall, our algorithm is executed following the three steps: 1) setting a boundary on one of the two objectives, 2) searching for a deployment which minimises the other objective while the first objective is within the boundary set and 3) traversing the available options to optimise the deployment found in step 2.

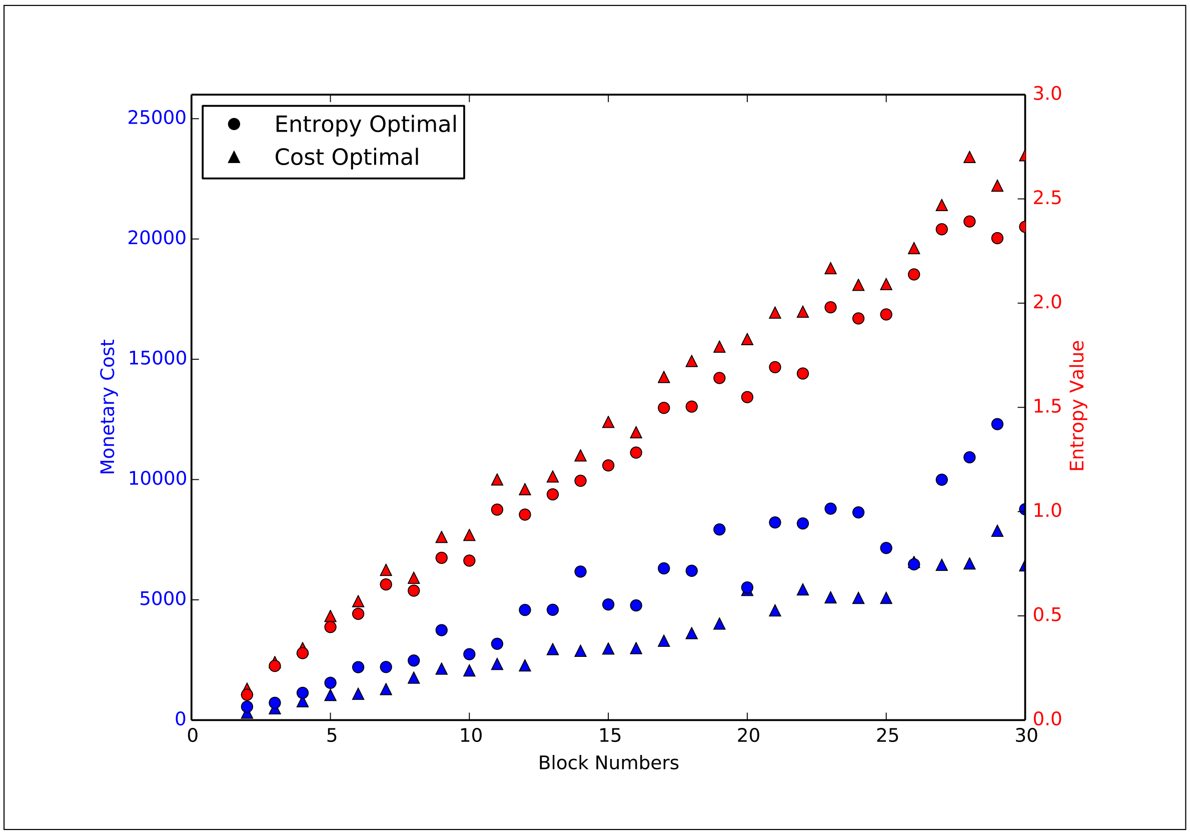


Figure 6.1: Cost and entropy optimal workflow deployments.

In the first step we set a bound $\mathcal{C}$ on entropy such that:

$$\begin{aligned} R_E(\phi) &\leq -nR_{max} \log \sqrt[n]{\frac{R_P(\phi^L) + R_P(\phi^M)}{2}} \\ &= -R_{max} \log \frac{R_P(\phi^L) + R_P(\phi^M)}{2} = \mathcal{C} \end{aligned} \tag{6.5}$$

where ϕ^M is the deployment which has the maximum reliability rate (see equation 6.2), and ϕ^L is the lower bound deployment of monetary cost without considering the reliability constraint. Therefore, $R_P(\phi^L)$ and $R_P(\phi^M)$ are the values of the power method, where R_{max} is the service which has the maximum reliability rate, deploying over federated clouds and also meet the security requirements. The n denotes that the target workflow includes n number of services. We could also make a bound on cost, however, we need to guarantee that the reliability rate is acceptable. We also do not set the boundary on security because all candidate solutions we generated meet the security requirements W.

As mentioned above, our optimisation is an extension of MCKP. MCKP is used to optimise a set of decisions $SET = \{set_1, \ldots, set_n\}$ within a defined constraint. In our scenario, SET represents the set of deployments Φ, where set_i is a mapping of each workflow object e onto a cloud platform wp, $set_i = \{e_1^{wp_i}, \ldots, e_m^{wp_j}\}$. Our algorithm aims to find the optimised set_i.

To realise this goal, firstly, we generate the ϕ^L (lower bound) and ϕ^M (most reliable) deployments that both satisfy security requirements. Then, we set $\mathcal{C}$ as the reliability constraints and invoke a set of optimisation mechanisms to improve ϕ^M. Thus, our algorithm consists of two phases: initial deployment and deployment optimisation.

6.4.1 Initial Deployment

The goal of the initial deployment is to generate quickly, in linear time, a solution which can be a seed to find the optimal deployment. First, we produce the cost optimised deployment ϕ^L by applying the NCF algorithm from our previous work [158]. That algorithm works as follows: (1) a greedy-based function determines a deployment which accounts for the costs related to running on a cloud, with each service considered in isolation; (2) the algorithm then takes into account direct links between services and redeploys services such that the influence of the data transfer costs is minimised. In both steps services' security requirements are guaranteed.

Given ϕ^L generated by NCF, we compare its entropy value with constraint $\mathcal{C}$. If $R_E(\phi^L) < \mathcal{C}$, ϕ^L will be the optimal solution returned by our algorithm. Otherwise, we apply a greedy algorithm to ïňĄnd the most reliable solution ϕ^M using the power method. The algorithm takes a list of the workïňĆow services and assigns each service to the most reliable cloud that meets the security constraints of that service. Once all services are assigned, ϕ^M becomes the seed for the next phase.

Note that ϕ^M is optimal, and can be generated by a greedy algorithm because calculating the reliability of a workflow using the power method has the optimal substructure property if we assume fault-free connections between services; by maximising the reliability of a single service we obtain the optimal solution if the reliability of remaining services is also maximised. For the proof please see Appendix B.

6.4.2 Deployment Optimisation

When ϕ^L is not a valid deployment, we use ϕ^M as the seed to find available options. We limit the search to find t deployment options and store the result in descending order of the cost. The optimal solution is the last on the list; the pseudocode is shown in Algorithm 9.

Algorithm 9 Deployment Optimisation

```
$\mathcal{W}$ – workflow; $\phi^M$ – the power method optimal deployment; $\mathcal{C}$ – entropy constraint;
$t$ – maximum number of deployments; $Cloud$ – valid clouds for each service; $M$ –
valid deployments;
$M[0] \leftarrow \phi^M$
$\phi \leftarrow \phi^M$
for $i \in 1...t$ do
                                        ▷ generate a new deployment from $\phi$
    $\phi^{NEW} \leftarrow CHANGE(\phi, Cloud)$;
    $\phi \leftarrow \phi^{NEW}$
    $M[i] \leftarrow \phi^{NEW}$
end for
sort $M$ by cost return $M[t]$
```

Finding t valid options was the key challenge in designing the algorithm due to the huge search space. To solve it, we observe the contribution of each cloud for each service, also taking into account its predecessors. To calculate the cost of deploying a service on a specific cloud we use the *COD* function. *COD* is calculated by adding the computing cost of service s_i to the transmission cost and storage cost of data sent from all of its immediate predecessor services that are not in the same cloud.

$$\begin{aligned} COD(s_i^p) =& SCOST(s_i^p) + CCOST(s_i^p) \\ &+ ECOST(s_i^p) \end{aligned}$$

Each s_i may have more than one valid cloud; therefore, the ranking of the valid clouds can be created by ascending order of *COD* value. This is described by the ranking algorithm (Algorithm 10).

As Algorithm 11 shows, the ranking is used to find an alternative cloud for s_i, which is chosen randomly by using the *Benford* function.

Algorithm 10 Rank

```
$\mathcal{W}$ – workflow;
function RANK($\phi$)
                                        ▷ Topsort returns a topological order of $\mathcal{W}$
    for $s_i \in topsort(\mathcal{W})$ do
        for $p \in Cloud[s_i]$ do
            $Tmp[p] \leftarrow COD(s_i^p)$
        end for
        sort $Tmp$ in the ascending order
        ▷ update the order of the clouds in $Cloud[s_i]$ by following the corresponding order of cache
        $Cloud[s_i] \leftarrow Tmp$
    end for
end function
```

Algorithm 11 Change

```
function CHANGE($\phi, Cloud$)
      ▷ Sort the cloud list by COD in the ascending order and update the Cloud list for the Benford function
    $Cloud \leftarrow RANK(\phi)$
    while true do
        Randomly choose a service $s_i$
        $\phi^{NEW} \Leftarrow Cloud[Benford(s_i)]$; ▷ Apply $Benford$ function to select a replace the cloud for $s_i$
        if $\phi^{NEW} \notin M$ and is secure and $H(\phi^{NEW}) \leq \mathcal{C}$ then return $\phi^{NEW}$
        end if
    end while
end function
```

We used randomisation because the optimal option is not always composed of the clouds which minimise the *COD*. By applying the function, the clouds at the head of the ranking have a higher chance of being selected but we also avoid the situation that some clouds are never chosen.

The *Benford* function is a transformation of Benford's law [86], which is defined as:

$$Benford(s_i) = \lfloor \frac{1}{10^b - 1} \rfloor$$
$$b = random(\log_{10}(1 + \frac{1}{len(Cloud[s_i])}), log_{10}2)$$

6.5 Results of the Experiments and Evaluation

In order to evaluate our algorithm we set up two experiments. First, we conducted a series of simulation experiments on a number of real scientific workflow applications. In the second experiment we applied our algorithm over federated clouds to deploy a scientific workflow from one of the projects that we have been involved in. Note that workflows in this work are defined as directed acyclic graphs where vertices represent tasks and arcs between them represent data dependencies.

6.5.1 Simulation Environment

In this work, the experiments are required to be repeatable in order to compare and analyse different types of algorithms. Therefore, to evaluate our algorithm we combined together two simulation environments WorkflowSim and DynamicCloudSim. WorkflowSim can simulate the execution of workflows of the Pegasus workflow management system [48], whereas DynamicCloudSim operates for the cloud simulation instability and task failures. This combination can more realistically simulate workflow execution in dynamic cloud environments.

6.5.1.1 Experimental Setup

To evaluate our algorithm we consider four workflow applications available in the Pegasus project and in WorkflowSim: CyberShake (earthquake risk characterisation),

Workflow app.	Medium	Large	Very large
CyberShake	30	100	1000
Montage	25	100	1000
LIGO	30	100	1000
Epigenomics	24	100	997

Table 6.1: Number of tasks in each Pegasus workflow for the three different sizes.

VM	Loc.	Exec (/hour)	Store (/hour/GB)	In (/GB)	Out (/GB)	Start-up Time (hour)
C1	0	0.40	0.10	0	0.02	5.0
C2	2	2.20	0.60	0.03	0.01	3.0
C3	1	1.23	0.30	0.14	0.07	4.5
C4	2	3.70	0.60	0.10	0.05	2.5
C5	3	4.50	0.90	0.14	0.05	1.5
C6	4	5.50	1.30	0.14	0.13	0.5

Table 6.2: Cloud Pricing and Security Levels

Montage (generation of sky mosaics), LIGO (detection of gravitational waves), and Epigenomics (bioinformatics).[2] For each of these applications we selected three sizes: medium, large and very large, which determine the number of tasks within the workflows (Table 6.1). The full characterisation of these workflow applications can be found in [90]. In our simulation, however, we only need to consider the execution time, input and output data, and security level of each service. As the data privacy information for the workflows is unavailable, we randomly generated the security levels for the services.

To represent a federation of workflow platforms we created six VMs in six different data centres each with a random security level in range 0–4. Note that more secure clouds are usually more expensive. Table 6.2 shows details of the platform setup with the cloud security level (*Location*) and cost of computation, storage, and incoming and outgoing communication. Additionally, *Start-up Time* indicates a randomly generated time when each WP has become available.

We compare our algorithm EMCK (extended multiple-choice knapsack) with DCA [121] and BDLS [52] which represent two existing and widely used multi-criteria scheduling algorithms. DCA and EMCK are both extensions of MCKP. However, DCA is focused on a single computing resource, and does not take into account the cost of data transfer and storage, so we adapted DCA to our security model and

[2]The XML description files of the workflows are available via the Pegasus project pages at 'https://confluence.pegasus.isi.edu/display/ pegasus/WorkflowGenerator'.

called the extended version EDCA. BDLS is a list scheduling algorithm that schedules services according to a priority list of service–resource pairs. For the purpose of the experiment, we had to extend and adapt BDLS as follows: (1) we applied the ranking mechanism similar to Algorithm 10 to build a cost list; (2) we added another list with entropy constraint for each service. As a result the services using extended BDLS (EBDLS) were assigned to the cloud which minimised the cost and also met the entropy requirement.

The experimental results presented below are the average values of 1000 deployments generated by each algorithm. The observed outputs are normalised against the results of the EMCK algorithm.

6.5.1.2 Monetary Cost and Time Complexity Evaluation

Figure 6.2, 6.3 and 6.4 depict the normalised monetary costs for all twelve application settings with four types of workflows in three sizes as mentioned previously. There are five different deployment solutions: EMCK, EBDLS, EDCA, lowerBound and mReliable. EMCK, EBDLS and EDCA represent the deployment solutions generated by the corresponding algorithms, whereas lowerBound (λ^L) and mReliable (λ^M) are the cost optimised and most reliable deployments, respectively.

The results show that EMCK almost always generates the cheapest deployment when compared with the other two algorithms. As for the case of large CyberShake workflow, it even produced deployments which were cheaper than our lowerBound result. The reason is that NCF, the algorithm used to generate the lowerBound solutions, is a heuristic-based algorithm [129] that can generate deployments based on cost and security yet it cannot guarantee the cheapest solution.

The performance of EBDLS is significantly influenced by the type and size of the workflow. For small and medium size workflows the cost of deployment solutions generated by EMCK and EBDLS is very close. However, for large CyberShare and Epigenomics workflows EMCK can save nearly 10% of cost when compared with EBDLS. Furthermore, for the very large Montage workflow EMCK yields 18% cost savings.

Figure 6.5 shows the normalised time consumption of generating a deployment solution

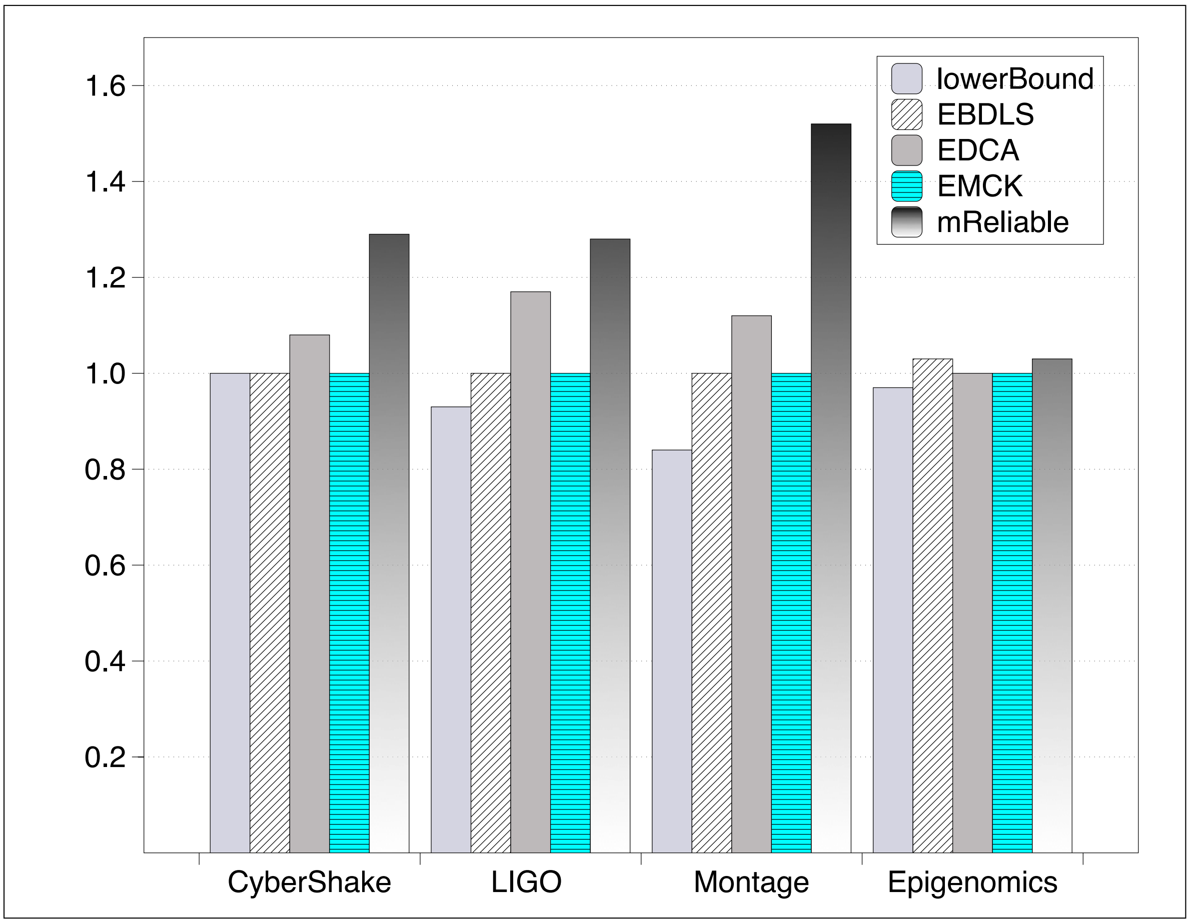


Figure 6.2: Cost for the medium size workflows.

for different algorithms. EBDLS is hundreds times faster than EMCK and with the increasing size of the workflow the gap is also increasing. Similarly, EDCA is ten times faster. Although our algorithm is much slower than the two others, the time to generate a deployment, even for very large workflows, is less than one minute, which is acceptable for our use.

6.5.1.3 Reliability evaluation

In order to simulate failures during workflow execution we combined WorkflowSim with DynamicCloudSim. DynamicCloudSim introduces a basic failure model which simulates the task fail during task execution. Whenever a task is assigned to a VM and its execution time is computed, DynamicCloudSim determines whether the task is bound to succeed or fail. This decision is based on the average rate of failure specified by the user [26].

Our experiment was set up as follows. Firstly, we set the initial reliability rate of each

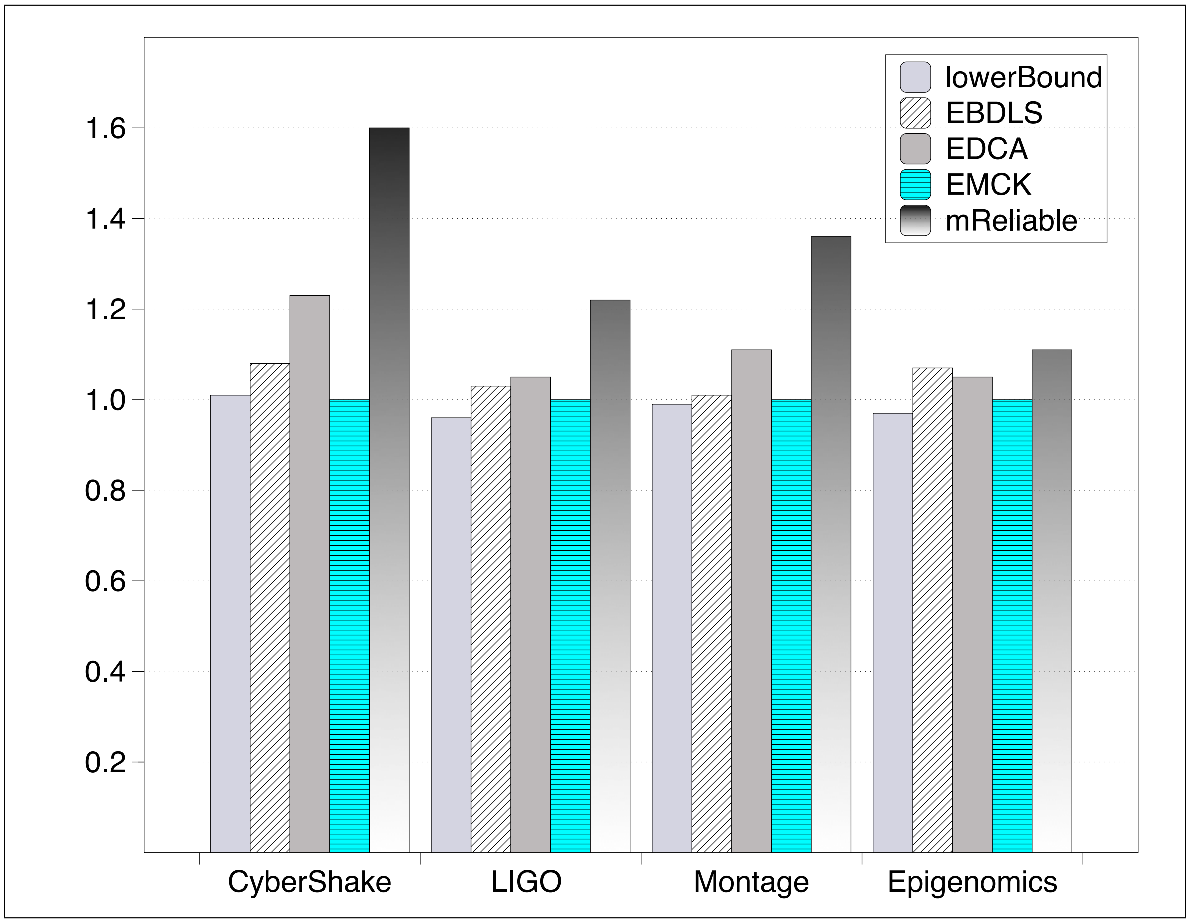


Figure 6.3: Cost for the large size workflows.

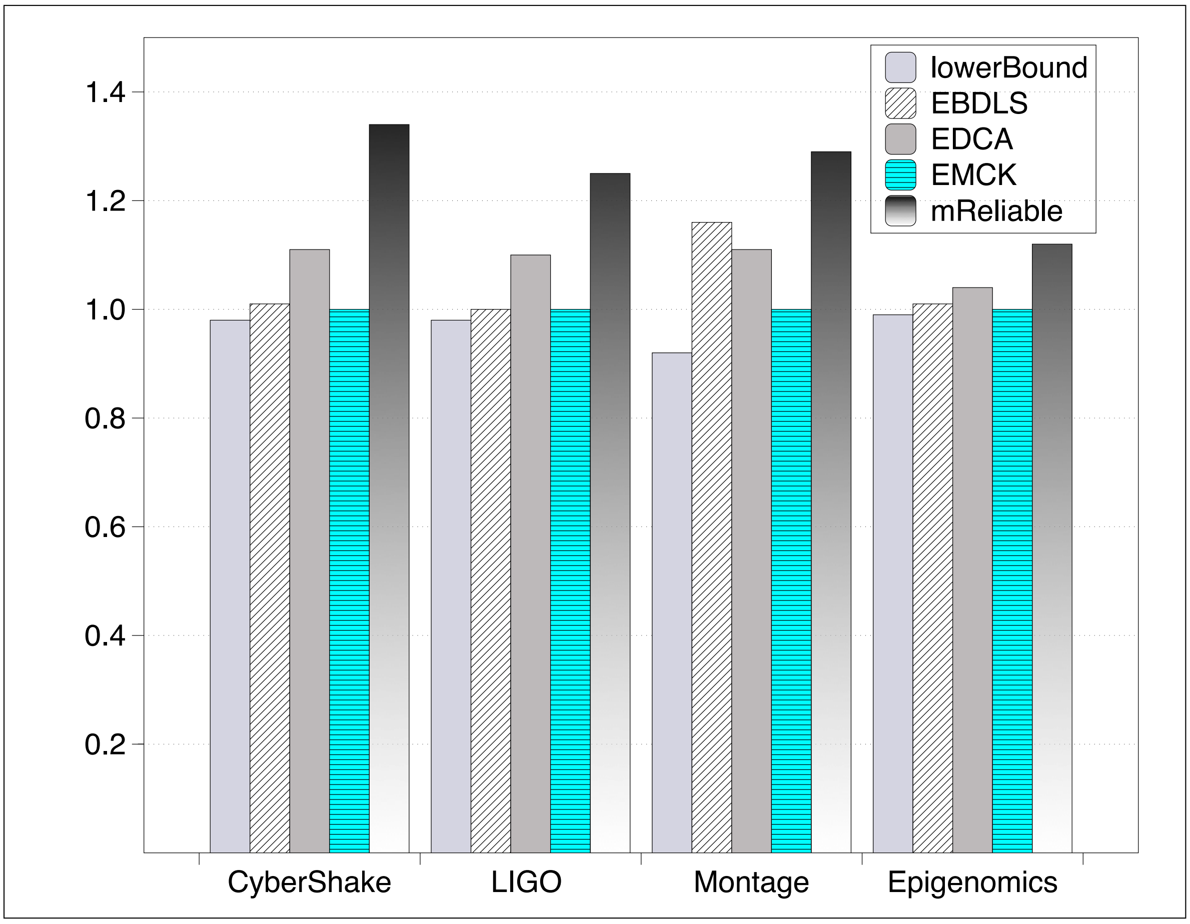


Figure 6.4: Cost for the very large size workflows.

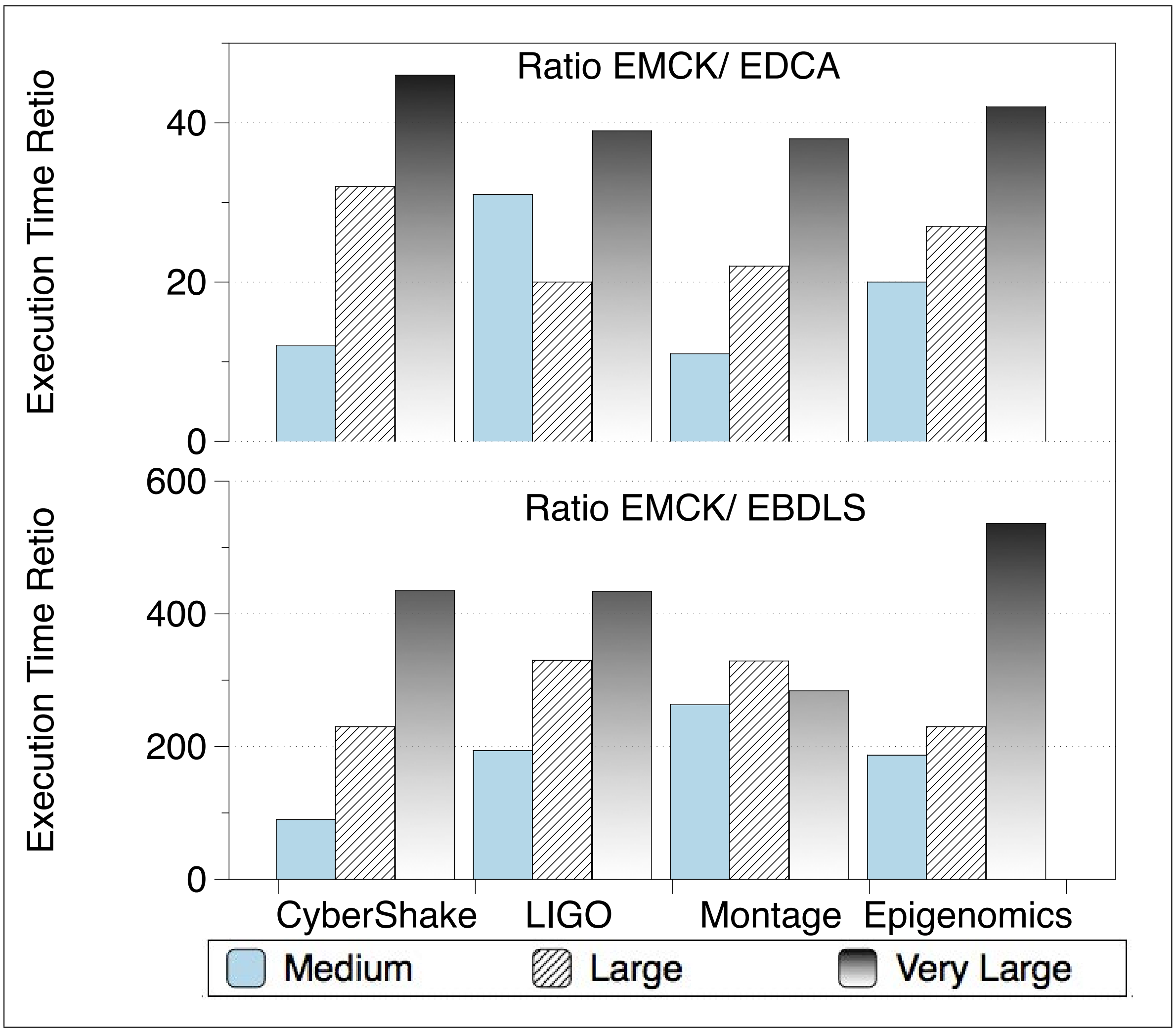


Figure 6.5: Time ratio of EMCK to the other two algorithms

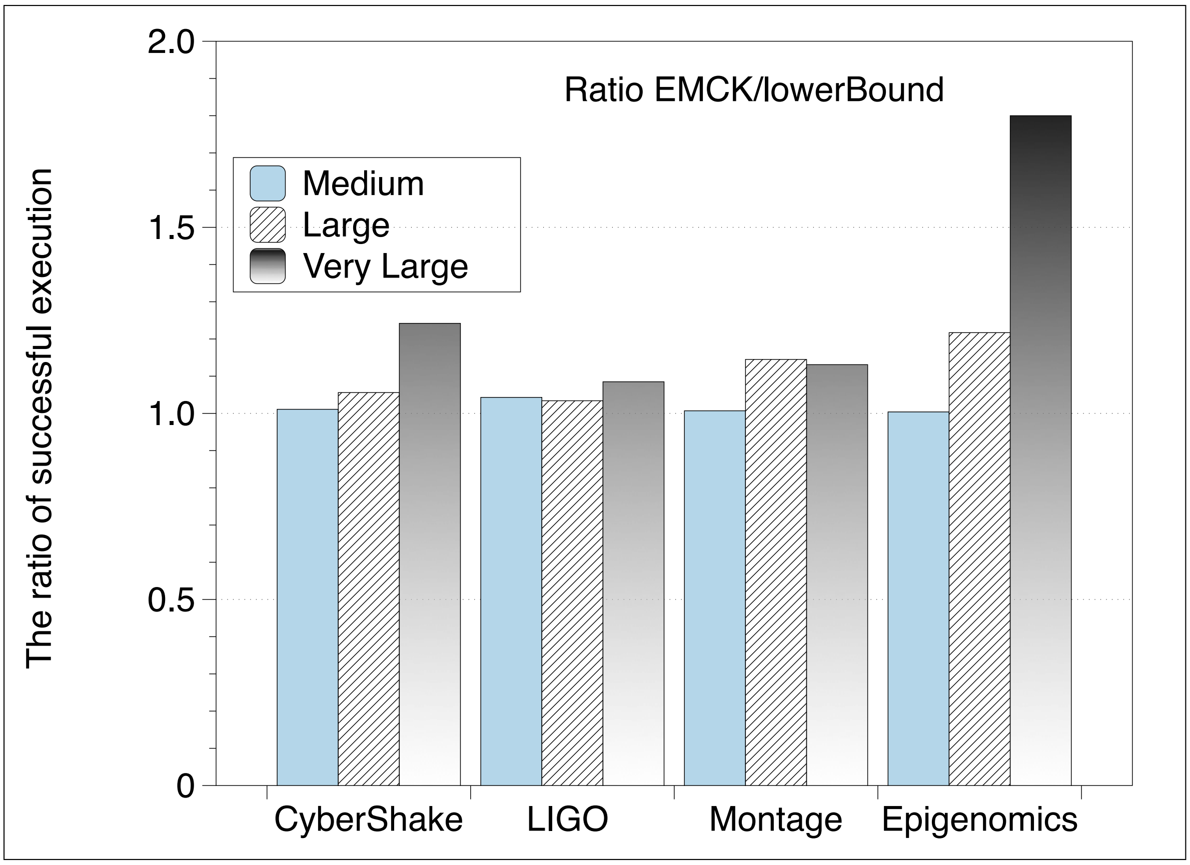


Figure 6.6: Comparing successful executions between two types of deployments of workflows for different size

cloud to 99.95% (The authors in [66] observed that a large proportion of fail events occur within 2 days in Google Cloud environment). Next, the VMs were initialised with security and cost parameters as shown in Table 6.2. The table also includes the start-up time of each VM, which is the time when a platform was started and is needed to calculate its reliability value at the moment when a workflow deployment occurs. Regarding the start-up time, the reliability rate of service s_i (or task in DynamicCloudSim) was $e^{-0.0005 \cdot t}$, where t was the start-up time of the VM assigned to execute s_i plus the execution time of s_i.

In this experiment we ran 1000 executions of the Pegasus workflows deployed using EMCK and NCF which generates cost optimised deployments ("lowerBound"). The results are presented in Figure 6.6 as the ratio of successful executions between these two algorithms. As shown, deployments generated by EMCK are always more reliable than the cost optimised ones. Furthermore, as workflows with more tasks suffer from more probability of failure, the ratio is more in favour of EMCK with the increasing size of the workflows. For example, for the very large size of the "Epigenomics" workflow, EMCK avoided 80% of failures when compared with deployments generated by NCF.

Importantly, workflow reliability is also influenced by the structure of the workflow. The "LIGO" workflow includes mostly short running services executed in parallel, whereas "Epigenomics" consists of chains of long running tasks. As a result, "Epigenomics" has much more chance to fail and so for this workflow the benefits of using EMCK are the most prominent.

6.5.2 Realistic System

To evaluate our algorithm in conditions closer to a production use we applied it to schedule scientific workflows in e-SC.

6.5.2.1 Tool Design

Figure 6.7 shows the architecture of our deployment tool. It consists of four core components: *Planner* assigns workflow partitions to *Federated Clouds* using the algorithm discussed above. The *Federated Clouds* consists of a set of e-Science Central instances

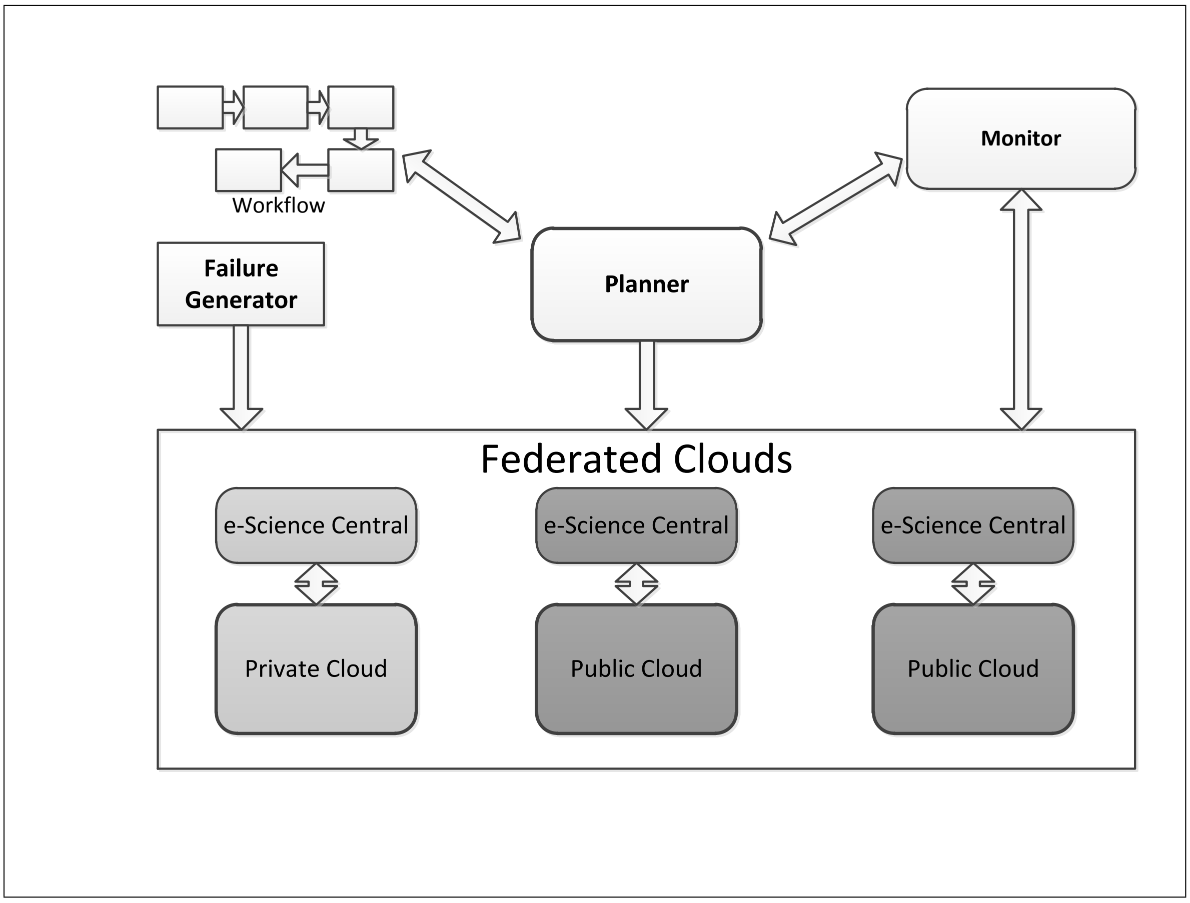


Figure 6.7: The Architecture of the Deployment Tool

running in different clouds. *Monitor* observes the status of each instance, detects failures, and provides the information about available instances to the *Planner*. Finally, *Failure Generator* is used to simulate failures by shutting down the e-SC instances with a predefined probability.

6.5.2.2 Experiment Setup

To verify our algorithm we selected one of the workflows used in the Cloud e-Genome project (the same workflow as previous chapter).

Guaranteeing that human genomic data can be securely processed on the cloud is very important. Therefore, we modelled the security requirements of a selected Cloud e-Genome workflow by assigning security levels as shown in Tables 6.3 and 6.4; note that the size of data transferred between blocks and the execution time of each block are actual values taken from logs collected by e-SC. Table 6.4 shows data sizes in GB, where 0 denotes less than 1 MB of data. The pricing shown in Table 6.5 is collected

Service	Name	Clearance	Location	Time(/h)
$Sample\_Name$	S1	1	0	1
$ImportFile$	S2	1	0	1.5
$Sample\_Vaule$ Info	S3	1	0	3
$HG19$	S4	1	0	0.1
$Filter$	S5	2	0	10
$Exome - Regions$	S6	1	0	7
$Intervalpadding$	S7	0	0	20
$ColumnJoin$	S8	2	0	0.1
$AnnotateSample$	S9	2	0	5
$Export$	S10	1	0	0.3

Table 6.3: Services representation and security and execution time

Data	Location	Size (GB)
$S_{1,8}$	1	0
$S_{2,5}$	0	1.1
$S_{3,8}$	2	0.01
$S_{4,5}$	0	0.005
$S_{4,7}$	0	0.005
$S_{5,7}$	0	6.2
$S_{6,7}$	0	10.3
$S_{7,9}$	1	3.6
$S_{8,9}$	0	0
$S_{9,10}$	0	0.05

Table 6.4: Data security and size.

Cloud	Pr1	Pu1	Pu2
Security	2	1	0
CPU	3.41(/h)	2.40(/h)	1.28(/h)
Pr1	0	0.1	0.11
Pu1	0.13	0	0.09
Pu2	0.07	0.02	0

Table 6.5: Basic attributes of the three clouds used in the experiment: security level, cost of computing resources, cost of data transfer between clouds (e.g. $Pr1 \rightarrow Pu1 = 0.1$)

Service	lowerBound	EMCK
S_1	Pr1	Pu1
S_2	Pu2	Pu2
S_3	Pr1	Pr1
S_4	Pu2	Pu2
S_5	Pu2	Pu1
S_6	Pu2	Pu2
S_7	Pu2	Pu2
S_8	Pr1	Pr1
S_9	Pu2	Pu2
S_{10}	Pu2	Pr1

Table 6.6: Two deployments

from two major cloud providers and is based on the equivalent VM configurations. Also one of the providers offered a private cloud service and we include its pricing in the table.

To simulate this environment, we set up three virtual machines and each of which runs a single instance of the e-SC system. VM1 was hosted on a personal PC and represented the private cloud. Two other VMs were hosted in our University virtualised environment and played the role of public cloud providers *Pu1* and *Pu2*.

As previously, to test our algorithm the platform's start-up time must be defined. We set $1.7h$, $2.4h$ and $3.5h$ for VM1, VM2 and VM3, and their initial reliability as 0.95, 0.93 and 0.90, respectively. Our working hypothesis was that: failure rates had to be unrealistically high in this experiment; demonstrating workflows fail frequently, as it was aimed. If not, we would need to run the longitudinal experiments for a prohibitively long time. For the purpose of the experiment, we also reduced the execution time of the given workflow by 30 seconds by scaling down the amount of input data shown in Table 6.3 by a factor of 6000.

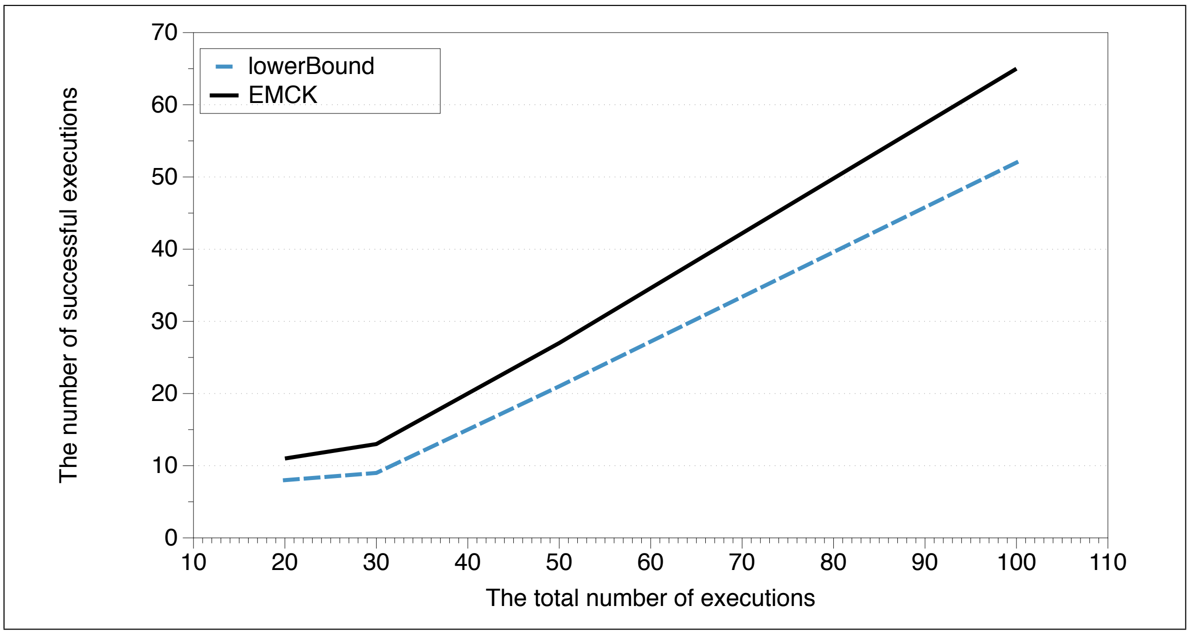


Figure 6.8: The execution results of two deployments

6.5.2.3 Results and Discussion

Based on the presented experiment setup we generated two deployments: “lowerBound” produced by the NCF algorithm and one generated by EMCK (Table 6.6). The cost of the workflow execution was 69.832 for the “lowerBound” and 128.897 for EMCK. Although this is nearly twice as costly as the cost optimised solution, the deployment produced by our algorithm improved reliability by about 25% (Figure 6.8).

One may challenge that the failure rate for both deployments were too high (about 40%). This is, however, the result of the high initial failure rate set for each VM. If the initial rate of each VM was set as the realistic value, there are lack of failures during the execution of the given workflow. This is not only because of the size of workflow is too small, but also the start-up time of each VM is just a few hours. However, the experiment in which we simulate more realistic reliability conditions were presented in the previous section, whereas in this experiment we show that our algorithm can be adapted to the real system and can generate correct deployments.

6.6 Conclusion

Driven by the popularity of moving to cloud, increasing numbers of workflow based applications are hosted in cloud. However, the variety and dynamics of cloud environments with security, reliability and price, lead users to struggle the users struggle in choosing a best cloud for deploying their applications. In this chapter, we presented a new algorithm to address the problem of deploying workflow applications over federated clouds meeting the reliability, security and monetary cost requirements. We have shown the trade-off between reliability and cost, and identified that the optimisation problem is NP-hardness problem. Our algorithm guarantees that the security and reliability constraints are met while optimising the monetary cost. The algorithm has been evaluated using realistic scientific workflows on both simulated environment and a real world cloud based platform. Experimental results show that our solutions can guarantee the security and reduce failures by 25% while generating the cheapest solution compared with other algorithms.

7 Conclusion

Contents

Summary

In this chapter, we firstly summarise the research work presented in this thesis. We then outline the contributions, and discuss open research problems in the field that could inform future work.

7.1 Thesis Summary

In this thesis, we have explored the algorithms and framework for partitioning a scientific workflow over federated clouds to meet different non-functional requirements. In this work, we design and implement a set of algorithms to allow a cloud workflow broker to distribute a scientific workflow over a federated cloud, in order to meet non-functional requirements, such as security, reliability and monetary cost.

Chapter 2 first gave an overview of scientific workflow deployment on clouds. Next, the state-of-the-art methods were classified into six categories: Financial Cost Driven, Security Driven, Reliability Driven, Performance Driven, Data Driven and Multi-Objective optimisation. In addition, the diversity of cloud environments offers the possibility of distributing a single workflow application to a set of clouds to satisfy the requirements of each component of the workflow. However, we showed that there is scarcity of research that has considered planning and deploying a workflow over federated clouds to meet different non-functional requirements. To bridge this gap, in this thesis we proposed a set of algorithms and a framework addressing these problems.

Chapter 3 described an implementation that adapts Watson's multi security level partition method [143] to distribute a workflow application over a federated cloud. The cloud federation is represented as a set of e-Science Central cloud based platform instances which are deployed over different clouds from different cloud providers. Moreover, a dynamic exception handling mechanism was designed and adapted to the tool to handle cloud failure when the workflow is running over a federated cloud. To this end, the mechanism is triggered when clouds fail, and it automatically selects the best way to repartition the workflow, whilst still meeting security requirements. This avoids the need for developers to code ad-hoc solutions to address cloud failure, or the alternative of simply accepting that an application will fail when a cloud fails.

Chapter 4 proposed an algorithm which can be used to partition a large size workflow over federated clouds to meet the security requirements while minimising monetary cost. Partitioning a workflow over a federated cloud is a NP-hard problem, therefore requiring an algorithm or a method that can efficiently generate a good-enough deployment solution. Thus, a new cost model was developed to calculate the monetary cost of workflow execution over a set of clouds. Additionally, the multi-level security model [143] was extended and adapted to the cost model. To generate a secure and cost optimised deployment solution, NCF (not cost first) algorithm was developed to efficiently partition a large size scientific workflow over a set of clouds to meet security requirements and also minimise the financial cost.

Chapter 5 proposed DoFCF (Deploy on Federated Cloud Framework) to dynamically deploy scientific workflow over a federated cloud to meet security requirements while minimising the monetary cost. As a framework, DoFCF can be easily adapted to any security model as well as other non-functional requirements models to partition scientific workflow over a set of clouds, while minimising the financial cost. In addition, a cloud federation can be dynamic, with the availability of clouds may change frequently. To this end, DoFCF also considers how to handle changes in cloud availability.

Chapter 6 different cloud providers offer various clouds with different levels of security, reliability and monetary cost. At the same, time user applications have become more and more complex. Often, they consist of a diverse collection of software components, and need to handle variable workloads, thereby posing different requirements on the infrastructure. Therefore, many organisations are considering using a combination of different clouds to satisfy these needs. This raises, however, a non-trivial issue of how to select the best combination of clouds to meet the application requirements. Therefore, a novel algorithm was developed to address the problem of deploying workflow applications over federated clouds to meet the reliability, security and monetary cost requirements. We have shown the trade-off between reliability and cost, and the optimisation problem is NP-hard. Our algorithm guarantees that the security and reliability constraints are met, while optimising the monetary cost.

7.1.1 Contributions on Partitioning Workflow Over Federated Clouds

In order to partition workflows over a federated clouds, cloud broker is required to distribute different partitions to different clouds. The broker can interact with the workflow execution environments, including computing resources and dependencies, therefore distributing tasks to destination resources, executing tasks in a predefined order, and monitoring the execution. Moreover, handling exceptions is an essential function of a cloud broker. In chapter 3 these problems were taken into account to illustrate a tool which can dynamically partition workflow application over federated clouds. In addition, we applied a security model to guarantee the security of workflow deployment in a cloud federation.

Usually, workflow applications are complex consisting of very large numbers of task. We need a smart algorithm to rapidly partition complex workflow applications. Thus in Chapter 4 we extended the security model and develop a novel cost model, based on this we develop an adaptive heuristic algorithm to partition workflow applications to federated clouds while meeting security requirements and minimising the monetary cost. However, the generated solutions can be sub-optimal solution.

In addition, the cloud federation is very dynamic and a cloud may leave the federation without notification, and a new cloud may join the federation when the workflow is running. In order to adapt to and benefit from this dynamism, we propose an new algorithm in Chapter 5 to dynamically schedule the running workflow over federated clouds when the cloud availability change.

Last but not least, we take more attributes into account in cloud federation, considering security, reliability and monetary cost in Chapter 6. We first develop an entropy based model to measure the reliability of workflow execution. A new multi-objectives optimisation algorithm was designed and implemented to partition workflows over a federated clouds to meet security and reliability requirements, while minimising monetary cost.

7.2 Future Research Directions

Here we motivate a number of areas of future research, which are inspired by the work of the present PhD project.

7.2.1 *A Generic Cloud Broker*

We implemented a cloud broker to evaluate our algorithms and framework, but that was based on the eScience Central API and can only execute the workflows over the eScience Central platform. Future work could address the design of a generic cloud Broker that can automatically provision the required execution environments (without installing the whole platform) when existing resources cannot meet the requirements. Currently, a major issue here is the lack of standard cloud APIs.

7.2.2 *Efficient Data Transmission Between Clouds*

Existing techniques mainly focus on gathering relevant data on a shared disk file system at one data centre before starting execution. However, this can be very time consuming. New data staging methods are therefore desirable; for example, caching can increase the efficiency of data transmission during scientific workflow execution.

7.2.3 *The Truthfulness of Cloud Providers*

All of the deployment methods or algorithms are based on trusting the accuracy of the information received about the status of resources. The cloud federation may contain a set of selfish cloud providers, hiding or misrepresenting their resource information. For example, the cloud providers might exaggerate the speed of their resources so as to charge higher prices fro the users of their resources. Game theory is a mathematical method used to analyse the decisions of agents in a problem modelled as a game [118], which could be explored for verifying the truthfulness of the cloud providers.

8
Appendix

Contents

8.1 Appendix A

Why entropy?

In the following, we first illustrate the mathematical representation of the entropy and power methods, and then explain why our entropy method is suitable and has more advantages than power.

We assume that $REL = \sqrt[n]{R_P(w)}$ is the expected reliability of each service. Thus, we aim to find a deployment ϕ which meets the requirement of $R_P(\phi) \geq REL^n$.

The power method can be defined as follows:

$$\begin{aligned} R_P(\phi) = \prod_{i=1}^{n} R_i \geq REL^n \\ \sqrt[n]{\prod_{i=1}^{n} R_i} = \prod_{i=1}^{n} R_i^{\frac{1}{n}} \geq REL \\ \sum_{i=1}^{n} \log R_i^{\frac{1}{n}} \geq \log REL \end{aligned} \tag{8.1}$$

where R_i is the reliability of service $s_i \in \phi$.

Note that the left hand side of Inequality 8.1 shows that the power method is a version of geometric mean.

Regarding the Entropy method, we can have the following assumption:

$$R_E(\phi') = -\sum_{i=1}^{n} R'_i \log R'_i \leq -nREL \log REL \tag{8.2}$$

where R'_i is the reliability of service $s_i \in \phi'$. Given Inequality 8.2, we show that R_E is weighted geometric mean.

Theorem 1. Entropy is a version of weighted geometric mean.

Proof. We assume $s_i \in w$ and $|\phi'| = n$, noting ϕ' includes n services. If R'_i is the reliability of service s_i, we can measure the reliability of ϕ' using the entropy method as shown by Inequality 8.2:

$$\frac{1}{n} \sum_{i \in n} R'_i \log R'_i \geq REL \log REL \tag{8.3}$$

And the left hand side of (8.3) can be transformed as:

$$\begin{aligned} \frac{1}{n}\sum_{i=1}^{n} R'_i \log R'_i &= \sum_{i=1}^{n} \log R_i'^{\frac{R'_i}{n}} \\ &= \log \prod_{i=1}^{n} R_i'^{\frac{R'_i}{n}} \end{aligned} \tag{8.4}$$

Q.E.D

The reason why entropy can avoid the distributions with huge differences in REL between services when compared to R_P is because Theorem 1 shows that R_E is a *weighted* geometric mean, whereas R_P is a geometric mean. *Compared with geometric mean, weighted geometric mean better reflects a situation when a shortage in one element limits the result and cannot be compensated by other elements* [87]. Therefore, in our case weighted geometric mean can convey more information about service reliability than simple geometric mean can. Importantly, the characteristic of R_P that the lower value has the higher reliability is still carried on with R_E.

However, from Inequality 8.2, we cannot guarantee $R_P(\phi') \geq REL^n$. In other words, Equation 8.1 and 8.2 are not necessary and sufficient condition. To overcome this issue, we assume $-R_E(\phi') = \log \prod_{i=1}^{n} R_i'^{\frac{R'_i}{n}} \geq X$, where X is an unknown auxiliary value. We, therefore, can have:

$$\log \prod_{i=1}^{n} R_i'^{R'_i} \geq nX \tag{8.5}$$

Let us set $R_M = MAX(R'_i)$ as the highest R'_i. Thus, the Inequality 8.5 can be transformed as:

$$\log \prod_{i=1}^{n} R_i'^{R'_i} \geq \log \prod_{i=1}^{n} R_i'^{R_M} = nX \tag{8.6}$$

We combine the condition $R_P(\phi') = \prod_{i=1}^{n} R'_i \geq REL^n$ with Inequality 8.6, as follows:

$$\begin{gathered} nX = R_M \log \prod_{i=1}^{n} R'_i \geq R_M \log REL^n \\ X \geq R_M \log REL \end{gathered} \tag{8.7}$$

Consequently, the constraint on entropy is:

$$\begin{gathered} \log \prod_{i=1}^{n} {R'_i}^{\frac{R'_i}{n}} \geq X \geq R_M \log REL \\ R_E(\phi') \leq -nR_M \log REL \end{gathered} \tag{8.8}$$

Mathematically, we have discussed that the method based on entropy has the advantage of reflecting a situation when one or a few elements limit the result which cannot be compensated by other elements. Therefore, by the application of this method, *not only can we quantitatively measure the reliability of deploying a workflow over federated clouds, but it also has better performance by avoiding the situation in which a deployment that meets the overall reliability requirement includes a service with significantly low reliability.* We illustrate this with the following experiment.

Let us consider a workflow composed of 6 services with randomly assigned reliability $R_i \in [0.9, 0.99]$. And let us assume that expected reliability of each service is $REL = 0.95$. We then randomly generate 1000 of such workflows and use entropy and the arithmetic, power and harmonic mean ($\frac{n}{\sum_{i=1}^{n} \frac{1}{R_i}}$) to measure the overall workflow reliability; note that the harmonic mean combines characteristic of both the average (arithmetic mean) and power method. The reliability constraint of each method is: $6 \cdot 0.95 \cdot log 0.95$; 0.95; 0.95^6; $\frac{6}{\sum_{i=1}^{6} \frac{1}{0.95}}$ for entropy, arithmetic mean, power and harmonic mean, respectively.

In this experiment, we aim to demonstrate that entropy can better avoid situations in which a deployment meets the reliability requirements, yet includes a service with significantly low reliability. Therefore, we apply the same constraint for each method, instead of Inequality 8.8 (actually, the constraint of Inequality 8.8 is more strict, thus the set of deployments that would match Inequality 8.8 is a subset of the one used in this experiment). Figure 8.1 shows that for power and both means nearly half of

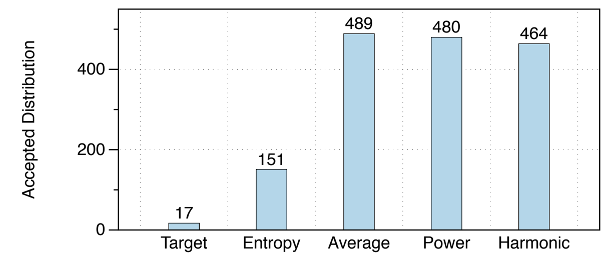


Figure 8.1: Measure Rate of each Method

the distributions meet the reliability constraint and so are acceptable. However, only for 17 workflows (marked as "Target" in Figure 8.1) reliability R_i of each service was greater than or equal to 0.95. As shown, the number of workflows accepted using the entropy method was much lower than for the other methods and much closer to the target.

8.2 Appendix B

Theorem 2. Calculating the reliability of a workflow using the power method has the optimal substructure property if we assume fault-free connections between services. Note that the problem is said to have the optimal substructure property if an optimal solution can be constructed from optimal solutions of its subproblems [41]. We use induction to prove this theorem.

Base case. Let us consider a workflow with n services and take a single service s1 and assign it to the platform which satisïňĄes the security constraints of s1 and which has the maximum reliability. Clearly, such assignment is the most reliable solution for this workflow.

Induction step. Now, let us consider a workflow with n services and assume that services $\{s_1, \ldots, s_k\}$ have been assigned such that $\prod_{i=1}^{k} s_i^{p_i}$ is maximal and each platform p_i satisfies security constraints of service s_i assigned to it. We then take s_{k+1} and assign it to the platform which satisfies its security constraints and has the highest reliability M. Since the connections between the services are fault-free, then $\prod_{i=1}^{k+1} s_i^{p_i} = \prod_{i=1}^{k} s_i^{p_i} \cdot M$ is the maximum reliability of the deployment of services $\{s_1, \ldots, s_k, s_{k+1}\}$.

Conclusions. By the principle of induction we showed that the optimal solution to calculate reliability can be constructed from optimal solutions to two sub-problems: finding the optimal solution to deploy the first k workflow services and finding the optimal solution to deploy service $k + 1$. Q.E.D.

Bibliography

[1] OpenNebula, url = http://www.opennebula.org, year = 2015.

[2] Summary of the amazon ec2 and amazon rds service disruption in the us east region, 2011.

[3] Eucalyptus home page, 2015.

[4] Java cog kit, 2015.

[5] Microsoft azure pricing, 2015.

[6] Montage galactic plane, 2015.

[7] Nimbus home page, 2015.

[8] C. S Alliance. Top threats to cloud computing v 1.0, 2010.

[9] I Altintas, C Berkley, E Jaeger, M Jones, B Ludascher, and S Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424, June 2004.

[10] R Ananthakrishnan, K Chard, I. T Foster, and S Tuecke. Globus platform-as-a-service for collaborative science applications. *Concurrency and Computation: Practice and Experience*, 27(2):290–305, 2015.

[11] M Armbrust, A Fox, R Griffith, A. D Joseph, R Katz, A Konwinski, G Lee, D Patterson, A Rabkin, I Stoica, and M Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[12] M Armbrust, A Fox, R Griffith, A. D Joseph, R Katz, A Konwinski, G Lee, D. A Patterson, A Rabkin, I Stoica, and M Zaharia. Above the clouds: A berkeley view of cloud computing, 2009.

[13] I Assayad, A Girault, and H Kalla. Tradeoff exploration between reliability, power consumption, and execution time for embedded systems. *International Journal on Software Tools for Technology Transfer*, 15(3):229–245, 2013.

[14] M Assis, L Bittencourt, and R Tolosana-Calasanz. Cloud federation: Characterisation and conceptual model. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 585–590, Dec 2014.

[15] V Atluri, S. A Chun, and P Mazzoleni. A chinese wall security model for decentralized workflow systems. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, pages 48–57, New York, NY, USA, 2001. ACM.

[16] C Augonnet, J Clet-Ortega, S Thibault, and R Namyst. Data-aware task scheduling on multi-accelerator based platforms. In *Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems*, ICPADS '10, pages 291–298, Washington, DC, USA, 2010. IEEE Computer Society.

[17] A Avizienis, J.-C Laprie, B Randell, and C Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, Jan 2004.

[18] S Azarnoosh, M Rynge, G Juve, E Deelman, M Niec, M Malawski, and R Da Silva. Introducing precip: An api for managing repeatable experiments in the cloud. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 19–26, Dec 2013.

[19] A Barker, J Weissman, and J van Hemert. Reducing data transfer in service-oriented architectures: The circulate approach. *Services Computing, IEEE Transactions on*, 5(3):437–449, Third 2012.

[20] A Barker and R Mann. Flexible service composition. In M Klusch, M Rovatsos, and T Payne, editors, *Cooperative Information Agents X*, volume 4149 of *Lecture Notes in Computer Science*, pages 446–460. Springer Berlin Heidelberg, 2006.

[21] D. E Bell and L. J LaPadula. Secure Computer Systems: Mathematical Foundations. Technical report, MITRE Corporation, March 1973.

[22] D Bhandari, C Murthy, and S. K Pal. Genetic algorithm with elitist model and its convergence. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(06):731–747, 1996.

[23] N Bhensook and T Senivongse. An assessment of security requirements compliance of cloud providers. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 520–525, 2012.

[24] L. F Bittencourt and E. R. M Madeira. A performance-oriented adaptive scheduler for dependent tasks on grids. (November 2007):1029–1049, 2008.

[25] L. F Bittencourt and E. R. M Madeira. HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds. *Journal of Internet Services and Applications*, 2(3):207–227, August 2011.

[26] M Bux and U Leser. Dynamiccloudsim: Simulating heterogeneity in computational clouds. *Future Generation Computer Systems*, 46(0):85 – 99, 2015.

[27] R Buyya, R Ranjan, and R. N Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'10, pages 13–31, Berlin, Heidelberg, 2010. Springer-Verlag.

[28] R Buyya, R Ranjan, and R. N Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'10, pages 13–31, Berlin, Heidelberg, 2010. Springer-Verlag.

[29] E.-K Byun, Y.-S Kee, J.-S Kim, and S Maeng. Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8):1011 – 1026, 2011.

[30] Z Cai, X Li, L Chen, and J Gupta. Bi-direction adjust heuristic for workflow scheduling in clouds. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pages 94–101, Dec 2013.

[31] J Cala, Y. X Xu, E. A Wijaya, and P Missier. From scripted HPC-based NGS pipelines to workflows on the cloud. In *Procs. C4Bio workshop, co-located with the 2014 CCGrid conference*, Chicago, IL, 2014. IEEE.

[32] R Calheiros and R Buyya. Meeting deadlines of scientific workflows in public clouds with tasks replication. *Parallel and Distributed Systems, IEEE Transactions on*, 25(7):1787–1796, July 2014.

[33] S Callahan, J Freire, J Freire, E Santos, C Scheidegger, C Silva, and H Vo. Managing the evolution of dataflows with vistrails. In *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*, pages 71–71, 2006.

[34] S Campana, D Rebatto, and A Sciaba'. Experience with the glite workload management system in atlas monte carlo production on lcg. *Journal of Physics: Conference Series*, 119(5):052009, 2008.

[35] F Cao and M Zhu. Distributed workflow mapping algorithm for maximized reliability under end-to-end delay constraint. *The Journal of Supercomputing*, 66(3):1462–1488, 2013.

[36] C.E.Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13, 1981.

[37] L Chen, V Tan, F Xu, A Biller, P Groth, S Miles, J Ibbotson, M Luck, and L Moreau. A proof of concept: Provenance in a service oriented architecture. In *In Proceedings of the UK OST e-Science Second All Hands Meeting 2005 (AHM05*, 2005.

[38] W Chen, Y Lee, A Fekete, and A Zomaya. Adaptive multiple-workflow scheduling with task rearrangement. *The Journal of Supercomputing*, 71(4):1297–1317, 2015.

[39] W Chen and E Deelman. Integration of workflow partitioning and resource provisioning. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 764–768, May 2012.

[40] W Chen and E Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *Proceedings of the 2012 IEEE 8th International Conference on E-Science (e-Science)*, E-SCIENCE '12, pages 1–8, Washington, DC, USA, 2012. IEEE Computer Society.

[41] T. M Cormen, C. E Leiserson, R. L Rivest, and C Stein. *Introduction to Algorithms, Third Edition.* MIT Press, 2009.

[42] K Deb, A Pratap, S Agarwal, and T Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Apr 2002.

[43] K Deb, A Pratap, S Agarwal, and T Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Apr 2002.

[44] E Deelman, G Singh, M Livny, B Berriman, and J Good. The cost of doing science on the cloud: The montage example. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–12, Nov 2008.

[45] E Deelman, D Gannon, M Shields, and I Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528 – 540, 2009.

[46] E Deelman, D Gannon, M Shields, and I Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[47] E Deelman, G Singh, M hui Su, J Blythe, Y Gil, C Kesselman, G Mehta, K Vahi, G. B Berriman, J Good, A Laity, J. C Jacob, and D. S Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *SCIENTIFIC PROGRAMMING JOURNAL*, 13:219–237, 2005.

[48] E Deelman, K Vahi, G Juve, M Rynge, S Callaghan, P. J Maechling, R Mayani, W Chen, R Ferreira da Silva, M Livny, and K Wenger. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2014.

[49] K Deng, K Ren, J Song, D Yuan, Y Xiang, and J Chen. A clustering based coscheduling strategy for efficient scientific workflow execution in cloud computing. *Concurrency and Computation: Practice and Experience*, 25(18):2523–2539, 2013.

[50] E Deza and M. M Deza. *Encyclopedia of Distances.* Springer, Berlin, Heidelberg, 2009.

[51] P. A Diaz-Gomez and D. F Hougen. Initial population for genetic algorithms: A metric approach. In *GEM*, pages 43–49, 2007.

[52] A Dogan and F Ozguner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):308–323, Mar 2002.

[53] J. J Dongarra, E Jeannot, E Saule, and Z Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '07, pages 280–288, New York, NY, USA, 2007. ACM.

[54] J. J Dongarra, E Jeannot, E Saule, and Z Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '07, pages 280–288, New York, NY, USA, 2007. ACM.

[55] A Doğan and F Özgüner. Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems*. *Comput. J.*, 48(3):300–314, May 2005.

[56] J. J Durillo, H. M Fard, and R Prodan. MOHEFT : A Multi-Objective List-based Method for Work fl ow Scheduling. pages 185–192, 2012.

[57] J. J Durillo and R Prodan. Workflow Scheduling on Federated Clouds. pages 318–329, 2014.

[58] H. M Fard, R Prodan, and T Fahringer. Multi-objective list scheduling of workflow applications in distributed computing infrastructures. *Journal of Parallel and Distributed Computing*, 74(3):2152 – 2165, 2014.

[59] H Fard, R Prodan, and T Fahringer. A truthful dynamic workflow scheduling mechanism for commercial multicloud environments. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1203–1212, June 2013.

[60] S Farokhi, F Jrad, I Brandic, and A Streit. HS4MC - hierarchical sla-based service selection for multi-cloud environments. In *CLOSER 2014 - Proceedings of the 4th International Conference on Cloud Computing and Services Science, Barcelona, Spain, April 3-5, 2014.*, pages 722–734, 2014.

[61] T Ferrandiz and V Marangozova. Managing scheduling and replication in the lhc grid. In *Grid Middleware and Services*, pages 65–77. Springer US, 2008.

[62] K Finley. Godaddy outage takes down millions of sites, anonymous member claims responsibility, year = 2012, url = http://techcrunch.com/2012/09/10/godaddy-outage-takes-down-millions-of-sites/, urldate = 2012.

[63] I Foster, Y Zhao, I Raicu, and S Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, Nov 2008.

[64] D. K G. von Laszewski, M. Hategan. Java cog kit workflow, in: Workflows for e-science. 119(5):143–166, 2007.

[65] M. R Garey and D. S Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[66] P Garraghan, P Townend, and J Xu. An empirical failure-analysis of a large-scale cloud computing environment. In *High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on*, pages 113–120, Jan 2014.

[67] Y Gil. Workflow composition: Semantic representations for flexible automation. In I Taylor, E Deelman, D Gannon, and M Shields, editors, *Workflows for e-Science*, pages 244–257. Springer London, 2007.

[68] Y Gil, V Ratnakar, E Deelman, G Mehta, and J Kim. Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In *Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence - Volume 2*, IAAI'07, pages 1767–1774. AAAI Press, 2007.

[69] Y Gil, V Ratnakar, E Deelman, M Spraragen, and J Kim. Wings for pegasus: A semantic approach to creating very large scientific workflows. In *IN THE EIGHTEENTH CONFERENCE ON INNOVATIVE APPLICATIONS OF ARTIFICIAL INTELLIGENCE*, pages 10–11, 2006.

[70] E Goettelmann, W Fdhila, and C Godart. Partitioning and cloud deployment of composite web services under security constraints. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pages 193–200, March 2013.

[71] I Goiri, J Guitart, and J Torres. Characterizing cloud federation for enhancing providers' profit. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 123–130, July 2010.

[72] D. E Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[73] D. E Goldberg and K Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[74] Y Gu and Q Wu. Maximizing workflow throughput for streaming applications in distributed environments. In *Computer Communications and Networks (ICCN), 2010 Proceedings of 19th International Conference on*, pages 1–6, Aug 2010.

[75] Y Gu, Q Wu, X Liu, and D Yu. Improving throughput and reliability of distributed scientific workflows for streaming data processing. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 347–354, Sept 2011.

[76] L Guo, Z He, S Zhao, N Zhang, J Wang, and C Jiang. Multi-objective optimization for data placement strategy in cloud computing. In C Liu, L Wang, and A Yang, editors, *Information Computing and Applications*, volume 308 of *Communications in Computer and Information Science*, pages 119–126. Springer Berlin Heidelberg, 2012.

[77] D Gupta and S Ghafir. An overview of methods maintaining diversity in genetic algorithms. *International Journal of Emerging Technology and Advanced Engineering*, 2(5):56–60, 2012.

[78] M Hakem and F Butelle. Reliability and scheduling on systems subject to failures. In *Parallel Processing, 2007. ICPP 2007. International Conference on*, pages 38–38, Sept 2007.

[79] M Hakem and F Butelle. Reliability and scheduling on systems subject to failures. In *Proceedings of the 2007 International Conference on Parallel Processing*, ICPP '07, pages 38–, Washington, DC, USA, 2007. IEEE Computer Society.

[80] B Hendrickson, R Leland, and R. V Driessche. Skewed graph partitioning. In *Proceedings of 8th SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[81] W. P Hiden H, Woodman S and C J. Developing cloud applications using the e-science central platform. *Royal Society of London. Philosophical Transactions A. Mathematical, Physical and Engineering Sciences*, 371:20120085, 2013.

[82] M HoseinyFarahabady, Y. C Lee, and A Zomaya. Pareto-optimal cloud bursting. *Parallel and Distributed Systems, IEEE Transactions on*, 25(10):2670–2682, Oct 2014.

[83] A Inc. *Amazon Elastic Compute Cloud (Amazon EC2).* Amazon Inc., http://aws.amazon.com/ec2.

[84] G Inc. *Google App Engine.* Google Inc., http://code.google.com/appengine.

[85] S Inc. *Salesforce.com.* Salesforce Inc, salesforce.com.

[86] D Jang, J. U Kang, A Kruckman, J Kudo, and S. J Miller. Chains of distributions, hierarchical bayesian models and benfordâĂŹs law. Technical report, 2008.

[87] M. D Jenness, J. and P Beier. Corridor designer evaluation tools.

[88] F Jrad, J Tao, I Brandic, and A Streit. SLA enactment for large-scale healthcare workflows on multi-cloud. *Future Generation Comp. Syst.*, 43-44:135–148, 2015.

[89] F Jrad, J Tao, I Brandic, and A Streit. {SLA} enactment for large-scale healthcare workflows on multi-cloud. *Future Generation Computer Systems*, 43âĂŞ44(0):135 – 148, 2015.

[90] G Juve, A Chervenak, E Deelman, S Bharathi, G Mehta, and K Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682 – 692, 2013. Special Section: Recent Developments in High Performance Computing and Security.

[91] G Juve and E Deelman. Wrangler: Virtual cluster provisioning for the cloud. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 277–278, New York, NY, USA, 2011. ACM.

[92] K Keahey and T Freeman. Contextualization: Providing one-click virtual clusters. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 301–308, Dec 2008.

[93] H Kellerer, U Pferschy, and D Pisinger. *Knapsack Problems.* Springer Verlag, Berlin, Germany, 2004.

[94] A KertÃl'sz, G Sipos, and P Kacsuk. Brokering multi-grid workflows in the p-grade portal. In W Lehner, N Meyer, A Streit, and C Stewart, editors, *Euro-Par 2006: Parallel Processing*, volume 4375 of *Lecture Notes in Computer Science*, pages 138–149. Springer Berlin Heidelberg, 2007.

[95] A KertÃl'sz and P Student. Brokering solutions for grid middlewares. In *in Pre-proc. of 1st Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, 2005.

[96] Y Klein and G Langholz. Multi-criteria scheduling optimization using fuzzy logic. In *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, volume 1, pages 445–450 vol.1, Oct 1998.

[97] H Kllapi, E Sitaridi, M. M Tsangaris, and Y Ioannidis. Schedule optimization for data processing flows on the cloud. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 289–300, New York, NY, USA, 2011. ACM.

[98] S Lamparter, A Ankolekar, R Studer, and S Grimm. Preference-based selection of highly configurable web services. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 1013–1022, New York, NY, USA, 2007. ACM.

[99] K Lee, N. W Paton, R Sakellariou, and A. A. A Fernandes. Utility functions for adaptively executing concurrent workflows. *Concurrency and Computation: Practice and Experience*, 23(6):646–666, 2011.

[100] Y. C Lee, H Han, A. Y Zomaya, and M Yousif. Resource-efficient workflow scheduling in clouds. *Knowledge-Based Systems*, 80(0):153 – 162, 2015. 25th anniversary of Knowledge-Based Systems.

[101] Y. C Lee, R Subrata, and A. Y Zomaya. On the performance of a dual-objective optimization model for workflow applications on grid platforms. *IEEE Trans. Parallel Distrib. Syst.*, 20(9):1273–1284, September 2009.

[102] Y. C Lee and A. Y Zomaya. Rescheduling for reliable job completion with the support of clouds. *Future Generation Computer Systems*, 26(8):1192 – 1199, 2010.

[103] H Li. *Introducing Windows Azure.* Apress, Berkely, CA, USA, 2009.

[104] B Liu, R. K Madduri, B Sotomayor, K Chard, L Lacinski, U. J Dave, J Li, C Liu, and I. T Foster. Cloud-based bioinformatics workflow platform for large-scale next-generation sequencing analyses. *Journal of Biomedical Informatics*, 49(0):119 – 133, 2014.

[105] G Liu, Y Zeng, D Li, and Y Chen. Schedule length and reliability-oriented multi-objective scheduling for distributed computing. *Soft Computing*, pages 1–11, 2014.

[106] W Liu, S Peng, W Du, W Wang, and G. S Zeng. Security-aware intermediate data placement strategy in scientific cloud workflows. *Knowl. Inf. Syst.*, 41(2):423–447, November 2014.

[107] X Liu, Z Ni, Z Wu, D Yuan, J Chen, and Y Yang. A novel general framework for automatic and cost-effective handling of recoverable temporal violations in scientific workflow systems. *Journal of Systems and Software*, 84(3):492 – 509, 2011.

[108] J Mace, A van Moorsel, and P Watson. The case for dynamic security solutions in public cloud workflow deployments. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pages 111–116, June 2011.

[109] R. K Madduri, P Dave, D Sulakhe, L Lacinski, B Liu, and I. T Foster. Experiences in building a next-generation sequencing analysis service using galaxy, globus online and amazon web service. In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, XSEDE '13, pages 34:1–34:3, New York, NY, USA, 2013. ACM.

[110] M Malawski, G Juve, E Deelman, and J Nabrzyski. Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 22:1–22:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[111] M Malawski, G Juve, E Deelman, and J Nabrzyski. Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. *Future Generation Computer Systems*, 48(0):1 – 18, 2015.

[112] G Manno, W Smari, and L Spalazzi. Fcfa: A semantic-based federated cloud framework architecture. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 42–52, July 2012.

[113] M Mao and M Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12, Nov 2011.

[114] P Mell, K Scarfone, and S Romanosky. Common vulnerability scoring system. *Security Privacy, IEEE*, 4(6):85–89, Nov 2006.

[115] P. M Mell and T Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[116] M Mihailescu and Y. M Teo. Dynamic resource pricing on federated clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 513–517, May 2010.

[117] M Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, USA, 1998.

[118] N Nisan, T Roughgarden, E Tardos, and V. V Vazirani. *Algorithmic Game Theory.* Cambridge University Press, New York, NY, USA, 2007.

[119] OMG. Deployment and Configuration of Component-based Distributed Applications Specification, 2006.

[120] S Pandey, L Wu, S Guru, and R Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 400–407, April 2010.

[121] R Prodan and M Wieczorek. Bi-criteria scheduling of scientific grid workflows. *Automation Science and Engineering, IEEE Transactions on*, 7(2):364–376, April 2010.

[122] X Qin and H Jiang. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. *Parallel Computing*, 32(5âĂŞ6):331 – 356, 2006.

[123] L Ramakrishnan, C Koelbel, Y suk Kee, R Wolski, D Nurmi, D Gannon, G Obertelli, A YarKhan, A Mandal, T Huang, K Thyagaraja, and D Zagorodnov. Vgrads: enabling e-science workflows on grids and clouds with fault tolerance. In *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pages 1–12, Nov 2009.

[124] G Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications.* Springer-Verlag, Berlin, Heidelberg, 1994.

[125] E. L Rhys Lewis. *Introduction to Reliability Engineering.* Wiley, November 15, 1995.

[126] B Rochwerger, D Breitgand, E Levy, A Galis, K Nagin, I Llorente, R Montero, Y Wolfsthal, E Elmroth, J Caceres, M Ben-Yehuda, W Emmerich, and F Galan. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4:1–4:11, July 2009.

[127] A Saleh, A Sarhan, and A Hamed. A new grid scheduler with failure recovery and rescheduling mechanisms: Discussion and analysis. *Journal of Grid Computing*, 10(2):211–235, 2012.

[128] C. E Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, January 2001.

[129] B Shirazi, M Wang, and G Pathak. Analysis and evaluation of heuristic methods for static task scheduling. *Journal of Parallel and Distributed Computing*, 10(3):222 – 232, 1990.

[130] Y Simmhan, B Plale, D Gannon, and S Marru. Performance evaluation of the karma provenance framework for scientific workflows. In L Moreau and I Foster, editors, *Provenance and Annotation of Data*, volume 4145 of *Lecture Notes in Computer Science*, pages 222–236. Springer Berlin Heidelberg, 2006.

[131] H. K Sonia Yassa, Rachid Chelouah and B Granado. Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. *The Scientific World Journal*, 2013:13, Apr 2013.

[132] O Sonmez, N Yigitbasi, S Abrishami, A Iosup, and D Epema. Performance analysis of dynamic workflow scheduling in multicluster grids. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 49–60, New York, NY, USA, 2010. ACM.

[133] C Szabo and T Kroeger. Evolving multi-objective strategies for task allocation of scientific workflows on public clouds. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8, June 2012.

[134] I Taylor, M Shields, I Wang, and A Harrison. The Triana Workflow Environment: Architecture and Applications. In I Taylor, E Deelman, D Gannon, and M Shields, editors, *Workflows for e-Science*, pages 320–339. Springer, New York, Secaucus, NJ, USA, 2007.

[135] I. J Taylor, E Deelman, D. B Gannon, and M Shields. *Workflows for e-Science: Scientific Workflows for Grids.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[136] H Topcuoglu, S Hariri, and M.-Y Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, Mar 2002.

[137] B Varghese and A Barker. Are clouds ready to accelerate ad hoc financial simulations? In *Proceedings of the 2014 IEEE/ACM International Symposium on Big Data Computing*, BDC '14, pages 54–63, Washington, DC, USA, 2014. IEEE Computer Society.

[138] C Vecchiola, X Chu, and R Buyya. Aneka: A software platform for .net-based cloud computing. *CoRR*, abs/0907.4622, 2009.

[139] T Velte, A Velte, and R Elsenpeter. *Cloud Computing, A Practical Approach.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2010.

[140] D Villegas, N Bobroff, I Rodero, J Delgado, Y Liu, A Devarakonda, L Fong, S Masoud Sadjadi, and M Parashar. Cloud federation in a layered service model. *J. Comput. Syst. Sci.*, 78(5):1330–1344, September 2012.

[141] J.-S Vöckler, G Juve, E Deelman, M Rynge, and B Berriman. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2Nd International Workshop on Scientific Cloud Computing*, ScienceCloud '11, pages 15–24, New York, NY, USA, 2011. ACM.

[142] X Wang, C. S Yeo, R Buyya, and J Su. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. *Future Generation Computer Systems*, 27(8):1124 – 1134, 2011.

[143] P Watson. A multi-level security model for partitioning workflows over federated clouds. *Journal of Cloud Computing: Advances, Systems and Applications*, 1(1):15, 2012.

[144] P Watson and M Little. Multi-level security for deploying distributed applications on clouds, devices and things. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 380–385, Dec 2014.

[145] Z Wen, J Cala, and P Watson. A scalable method for partitioning workflows with security requirements over federated clouds. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 122–129, Dec 2014.

[146] Z Wen and P Watson. Dynamic exception handling for partitioned workflow on federated clouds. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 198–205, Dec 2013.

[147] D Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.

[148] D Williams, H Jamjoom, and H Weatherspoon. Plug into the supercloud. *Internet Computing, IEEE*, 17(2):28–34, March 2013.

[149] S Woodman, H Hiden, P Watson, and P Missier. Achieving reproducibility by combining provenance with service and workflow versioning. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, WORKS '11. ACM, 2011.

[150] T Xiaoyong, K Li, Z Zeng, and B Veeravalli. A novel security-driven scheduling algorithm for precedence-constrained tasks in heterogeneous distributed systems. *Computers, IEEE Transactions on*, 60(7):1017–1029, July 2011.

[151] F. S Y, S. S Y, T. F A, and T hoon Kim. Model of business service management with security approach 1, 2009.

[152] Z Yu and W Shi. An adaptive rescheduling strategy for grid workflow applications. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, March 2007.

[153] D Yuan, Y Yang, X Liu, and J Chen. A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, April 2010.

[154] L Zeng, B Veeravalli, and X Li. Saba: A security-aware and budget-aware workflow scheduling strategy in clouds. *Journal of Parallel and Distributed Computing*, 75(0):141 – 151, 2015.

[155] L Zeng, B Veeravalli, and A. Y Zomaya. An integrated task computation and data management scheduling strategy for workflow applications in cloud environments. *Journal of Network and Computer Applications*, 50(0):39 – 48, 2015.

[156] Y Zhang, C Koelbel, and K Cooper. Hybrid re-scheduling mechanisms for workflow applications on multi-cluster grid. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pages 116–123, May 2009.

[157] Y Zhao, Y Zhang, W Tian, R Xue, and C Lin. Designing and deploying a scientific computing cloud platform. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, pages 104–113, Sept 2012.

[158] F Zheng, H Zou, G Eisenhauer, K Schwan, M Wolf, J Dayal, T.-A Nguyen, J Cao, H Abbasi, S Klasky, N Podhorszki, and H Yu. Flexio: I/o middleware for location-flexible scientific data analytics. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 320–331, May 2013.

[159] Z Zheng and M Lyu. Selecting an optimal fault tolerance strategy for reliable service-oriented systems with local and global constraints. *Computers, IEEE Transactions on*, 64(1):219–232, Jan 2015.

[160] A Zhou and B He. Transformation-based monetary costoptimizations for workflows in the cloud. *Cloud Computing, IEEE Transactions on*, 2(1):85–98, Jan 2014.

[161] X Zhu, X Qin, and M Qiu. Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters. *Computers, IEEE Transactions on*, 60(6):800–812, June 2011.

[162] A. F Zorzo and R. J Stroud. A distributed object-oriented framework for dependable multiparty interactions. In *Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 435–446. ACM Press, 1999.