University Library



A gateway to Melbourne's research publications

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Grozev, Nikolay

Title:
Brokering techniques for managing three-tier applications in distributed cloud computing environments

Date:
2015

Persistent Link:
https://hdl.handle.net/11343/58928

Brokering Techniques for Managing Three-Tier Applications in Distributed Cloud Computing Environments

Nikolay Grozev

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

October 2015

Department of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE, AUSTRALIA

Produced on archival quality paper.

Copyright © 2015 Nikolay Grozev

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the author except as permitted by law.

Brokering Techniques for Managing Three-Tier Applications in Distributed Cloud Computing Environments

Nikolay Grozev
Supervisor: Prof. Rajkumar Buyya

Abstract

Cloud computing is a model of acquiring and using preconfigured IT resources on demand. Cloud providers build and maintain large data centres and lease their resources to customers in a *pay-as-you-go* manner. This enables organisations to focus on their core lines of business instead of building and managing in-house infrastructure. Such in-house IT facilities can often be either under or over utilised given dynamic and unpredictable workloads. The cloud model resolves this problem by allowing organisations to flexibly resize/scale their rented infrastructure in response to the demand. The confluence of these incentives has caused the recent widespread adoption of cloud services.

However, cloud adoption has introduced challenges in terms of service unavailability, regulatory compliance, low network latency to end users, and vendor lock-in. These factors are of special importance for large scale interactive web-facing applications, which observe unpredictable workload spikes and need to serve users worldwide with low latency. The utilisation of multiple cloud sites (i.e. a Multi-Cloud) has emerged as a promising solution. Multi-Cloud approaches also facilitate cost reduction by taking advantage of the diverging prices in different cloud sites.

The 3-Tier architectural model is the de-facto standard approach to build interactive web systems. It divides the application in three tiers: (i) presentation tier which implements the user interfaces, (ii) domain tier implementing the core business logic, and (iii) data tier managing the persistent storage. This logical division most often leads to deployment separation as well.

This thesis investigates dynamic approaches for workload distribution and resource provisioning (a.k.a. brokering) of 3-Tier applications in a Multi-Cloud environment. It advances the field by making the following key contributions:

1. A performance model and a simulator for 3-Tier applications in one and multiple clouds.
2. A system architecture for brokering 3-Tier applications across clouds, which considers latency, availability, and regulatory requirements and minimises the overall operational costs.
3. An approach for Virtual Machine (VM) type selection that reduces the total cost within a cloud site. It uses online machine learning techniques to address the variability of both the application requirements and the capacity of the underlying resources.
4. A rule-based domain specific model for regulatory requirements, which can be interpreted by a rule inference engine.
5. Design and implementation of a workload redirection system that directs end users to the individual cloud sites in a Multi-Cloud environment.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Nikolay Grozev, 22 October 2015

Preface

This thesis has been written at the Department of Computing and Information Systems, The University of Melbourne, Australia. Chapters 2 — 6 contain the main contributions of this thesis and are based on the following publications:

- **Nikolay Grozev** and Rajkumar Buyya, "Inter-cloud Architectures and Application Brokering: Taxonomy and Survey", *Software: Practice and Experience*, John Wiley & Sons, Ltd, vol. 44, no. 3, pp. 369–390, 2014.
- **Nikolay Grozev** and Rajkumar Buyya, "Performance Modelling and Simulation of Three-Tier Applications in Cloud and Multi-Cloud Environments", *The Computer Journal*, Oxford University Press, vol. 58, no. 1, pp. 1–22, 2015.
- **Nikolay Grozev** and Rajkumar Buyya, "Multi-Cloud Provisioning and Load Distribution for Three-tier Applications", *ACM Transactions on Autonomous and Adaptive Systems*, vol. 9, no. 3, pp. 13:1–13:21, 2014.
- **Nikolay Grozev** and Rajkumar Buyya, "Dynamic Selection of Virtual Machines for Application Servers in Cloud Environments", *Technical Report CLOUDS-TR-2016-1, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne*, February 5, 2016.
- **Nikolay Grozev** and Rajkumar Buyya, "Regulations and Latency Aware Load Distribution of Web Applications in Multi-Clouds", *The Journal of Supercomputing*, Springer (Under Review), 2015.

Acknowledgements

PhD is a rewarding but challenging journey, which would not be possible without the help of many people. First and foremost, I would like to express my gratitude to my supervisor Professor Rajkumar Buyya for giving me the opportunity to pursue my studies in the renowned CLOUDS Laboratory. His pragmatic guidance, insightful advice, and continuous support made it all possible.

I would like to thank the members of my PhD committee Professor James Bailey and Dr Rodrigo Calheiros for their support and guidance. I also thank Dr Amir Vahid and Dr Anton Beloglazov, who helped me define my research direction.

I would like to thank all past and current members of the CLOUDS Laboratory, at the University of Melbourne: Dileban Karunamoorthy, Adel Nadjaran Toosi, Atefeh Khosravi, Sareh Fotuhi, Yaser Mansouri, Chenhao Qu, Deepak Poola, Yali Zhao, Jungmin Jay Son, Bowen Zhou, Farzad Khodadadi, Deborah Magalhes, Tiago Justino, Safiollah Heidari, Liu Xunyun, Nitisha Jain, Saurabh Garg, William Voorsluys, and Christian Vecchiola for their friendship and support. I have had a great time with them and my other friends from the Department of Computing and Information Systems (CIS) — Mohammed Alrokayan, Jubaer Arif, Simone Romano, Goce Ristanoski, and Sergey Demyanov to name a few. I also thank my other friends in Australia, USA, and Bulgaria.

I thank the University of Melbourne, the Australian Research Council (ARC), and the Australian Government for providing scholarships and appropriate facilities to pursue doctoral studies. I also thank the administrative staff members Rhonda Smithies, Madalain Dolic, and Julie Ireland and the Head of the CIS Department Professor Justin Zobel for their guidance and advice.

I am grateful to my colleagues and professors from the Faculty of Mathematics and Informatics (FMI) at the University of Sofia "St. Kliment Ohridski", where I studied previously. They taught me fundamental concepts and understanding that enabled my further studies.

I am grateful to Microsoft for awarding me an internship in their headquarters in Redmond, Seattle. I have enjoyed and learned a lot during my 3-month stay with them. I thank Amazon for providing research grants that enabled me to validate my work in a realistic industry environment.

Finally, I would like to thank my family. Their selfless patience, love, and encouragements in times of trouble and doubt have been extraordinary.

Nikolay Grozev
Melbourne, Australia
22 October 2015

Contents

1 Introduction **1**
- 1.1 Motivation and Scope . . . 4
- 1.2 Research Problem and Objectives . . . 5
- 1.3 Evaluation Methodology . . . 7
- 1.4 Thesis Contributions . . . 8
- 1.5 Thesis Organisation . . . 10

2 Inter-Cloud Architectures and Brokering: Taxonomy and Survey **13**
- 2.1 Introduction . . . 13
- 2.2 Definitions . . . 15
- 2.3 Architectural Taxonomy . . . 17
- 2.4 Taxonomy of Inter-Cloud Application Brokering Mechanisms . . . 20
- 2.5 Application-Centric Perspective to Inter-Clouds . . . 23
  - 2.5.1 Taxonomy of Inter-Cloud Applications . . . 23
  - 2.5.2 Requirements for Inter-Cloud Environments . . . 25
- 2.6 State Of The Art in Inter-Clouds . . . 28
  - 2.6.1 Centralised Federated Inter-Clouds . . . 29
  - 2.6.2 Peer-To-Peer Federated Inter-Clouds . . . 32
  - 2.6.3 Multi-Cloud Services . . . 36
  - 2.6.4 Inter-Cloud Libraries . . . 41
- 2.7 Discussion . . . 42
- 2.8 Summary . . . 46

3 Performance Modelling and Simulation of 3-Tier Applications **49**
- 3.1 Introduction . . . 49
- 3.2 Related Work . . . 51
- 3.3 Overview . . . 54
  - 3.3.1 Architectural Setting . . . 54
  - 3.3.2 Assumptions and Scope . . . 56
  - 3.3.3 Essence of the Model . . . 58
- 3.4 Analytical model . . . 60
  - 3.4.1 Session Model . . . 60
  - 3.4.2 Modelling Resource Contention . . . 63
  - 3.4.3 Session Arrival Model . . . 64
  - 3.4.4 Performance Variability across Clouds . . . 64
- 3.5 Simulator Implementation . . . 65

3.5.1 Representation of Disk I/O Operations . . . 65
3.5.2 Provisioning of I/O Operations . . . 66
3.5.3 Representing Sessions and Contention . . . 66
3.5.4 Performance Variability . . . 70
3.5.5 Load Balancing . . . 70
3.5.6 Workload Generation . . . 70
3.6 Use Cases . . . 72
3.7 Validation . . . 74
3.7.1 Session Performance Baseline . . . 76
3.7.2 Experiment 1 — Static Workload . . . 79
3.7.3 Experiment 2 — Dynamic Workload . . . 84
3.8 Summary . . . 88

4 Multi-Cloud Provisioning and Load Distribution for 3-Tier Applications 89
4.1 Introduction . . . 89
4.2 Related Work . . . 90
4.3 Preliminaries and Scope . . . 92
4.4 Overall Architecture . . . 93
4.4.1 Architectural Components . . . 93
4.4.2 Component Interaction . . . 96
4.5 Provisioning and Workload Management . . . 97
4.5.1 Scalability Within a Cloud . . . 97
4.5.2 Data Centre Selection . . . 101
4.5.3 Fault Tolerance . . . 106
4.6 Performance Evaluation . . . 106
4.6.1 Experiment Setting . . . 107
4.6.2 Experimental Application and Workload . . . 108
4.6.3 Baseline Approach . . . 109
4.6.4 Results . . . 110
4.7 Summary . . . 114

5 Dynamic Selection of Virtual Machines for Application Servers 115
5.1 Introduction . . . 115
5.2 Related Work . . . 118
5.3 Method Overview . . . 120
5.4 Learning Application Behaviour . . . 122
5.4.1 Resource Monitoring . . . 122
5.4.2 Normalisation and Capacity Estimation . . . 122
5.4.3 Anomaly Detection Through HTM . . . 124
5.4.4 ANN Training . . . 127
5.5 Virtual Machine Type Selection . . . 131
5.6 Benchmark and Prototype . . . 133
5.7 Validation . . . 135
5.8 Summary . . . 140

6 Regulations and Latency Aware Load Distribution **141**
6.1 Introduction . . . 141
6.2 Related Work . . . 143
6.3 System Architecture . . . 144
6.4 Design and Implementation . . . 146
6.4.1 Entry Points . . . 146
6.4.2 Admission Controllers . . . 150
6.5 Performance Evaluation . . . 155
6.5.1 Experiment 1 — Large Scale Deployment . . . 155
6.5.2 Experiment 2 — Fault Tolerance . . . 159
6.6 Summary . . . 161

7 Conclusions and Future Directions **163**
7.1 Summary and Discussion . . . 163
7.2 Future Directions . . . 166
7.2.1 Models for Asynchronous I/O Bound Applications . . . 166
7.2.2 Provisioning Techniques Using A Mixture of VM Pricing Models . 167
7.2.3 Dynamic Replacement of Application Server VMs . . . 167
7.2.4 VM Type Selection In Private Clouds . . . 168
7.2.5 Regulatory Requirements Specification Using Industry Standards . 168
7.2.6 Generalisation to Multi-Tier Applications . . . 168
7.3 Final Remarks . . . 169

List of Figures

1.1 3-Tier application in a single cloud data centre. 2
1.2 An interactive Multi-Cloud system. 3
1.3 Thesis organisation. 11

2.1 An overview of Inter-Cloud approaches and use cases. 14
2.2 Classification of cloud infrastructures. 17
2.3 Architectural classification of Inter-Clouds. 20
2.4 Inter-Cloud architectures. 21
2.5 Application brokering mechanisms. 22
2.6 Classification of distributed applications. 24
2.7 Inter-Cloud projects by architecture. 43
2.8 Inter-Cloud projects by brokering mechanisms. 44

3.1 3-Tier reference architecture deployed in multiple clouds. 54
3.2 RAM load represented as stacked sessions' load. 59
3.3 Session representation. 68
3.4 Workload generation. 71
3.5 Main use case scenarios. 72
3.6 Relational model of the RUBiS workload. 75
3.7 CPU utilisation of the DB server with 1 and 100 simultaneous sessions for the initial 5 minutes. 79
3.8 CPU and disk utilisations with 300 and 600 simultaneous sessions. The plotted values are averaged for every 90 seconds. 80
3.9 Execution delays (boxplots) and predicted delays (crossing horizontal lines) in Experiment 1. 81
3.10 Predicted and actual CPU utilisation of the AS server with 300 simultaneous sessions in Experiment 1. 82
3.11 Predicted and actual disk I/O utilisation of the DB server with 50, 300 and 600 simultaneous sessions in Experiment 1. 83
3.12 Average hourly frequencies of session arrivals for the two data centres in Experiment 2. 85
3.13 CPU and disk utilisations of the AS and DB servers in DC1 and DC2 at simulation time in Experiment 2. 87

4.1 Overall layered architecture. The Brokering components mange the system's provisioning and workload distribution, while a standard 3-Tier software stack serves the end users. 94

4.2 Component Interaction. Cloud site selection happens once, upon the user arrival. Subsequent requests are handled within the selected data centre. . 95
4.3 User arrival frequencies per minute in the entry points of the EU data centres (*DC-EU-E*, *DC-EU-G*) and the US ones (*DC-US-E*, *DC-US-G*). 108
4.4 Session outcome over time. 111
4.5 Session delays in seconds. 112
4.6 Network latency between users and serving data centres in milliseconds. The mean is denoted with a rhombus. 114

5.1 A 3-Tier application in Cloud. Whenever the autoscaling conditions are activated, a new application server should be provisioned. In this cjapter we select the optimal VM type for the purpose. 117
5.2 System components and their interaction. 120
5.3 HTM region structure. 125
5.4 ANN topology. 127
5.5 The $lr_k^{(1)}$ approximation of the *learning rate* and the respective momentum during the initial ANN training stages. 131
5.6 CloudStone benchmark's extended topology. 134
5.7 Learning rate lr_k during initial stages of training the ANN. 137
5.8 RMSE-pre in the presence of a workload change. The 0 index corresponds to the first sample after the workload change. 138
5.9 Timelines and costs of all VMs grouped by experiments. DVTS is our approach. The AWS-style policies are the baselines, which statically select a predefined VM type. 139

6.1 Overall Architecture: *Entry Point* and *Admission Controller* components replace the DNS based cloud site selection. 145
6.2 *Entry Point* — *Admission Controller* interaction. 147
6.3 Domain model classes used for cloud site to user matching. 151
6.4 Layers of DRL rules. Each layer can use the rules from the underneath layers. 152
6.5 Interfaces for the relations from Layer 1 and the *ContainsJurisdiction* relation from Layer 2. 153
6.6 Experiment 1 — 4 cloud sites host the *Entry Points* and *Admission Controllers*; 6 cloud sites are used to emulate incoming users. 156
6.7 Dispatch times of users to their serving cloud sites, grouped by location of the emulated users. Mean values are denoted with a rhombus. 157
6.8 Number of users redirected to each cloud site grouped by location. 158
6.9 Experiment 2 — Number of users directed to each cloud site. 160
6.10 Experiment 2 — Dispatch times over time. 161

List of Tables

2.1 Typical Inter-Cloud brokering requirements. 28
2.2 Summary of Inter-Cloud projects. 44

3.1 Summary of related work. 52
3.2 95% Confidence Intervals (CIs) and estimates of the median of the utilisation differences between simulation and execution. Measurements are in percentages. 83
3.3 Hourly frequencies of the workload for Experiment 2. 85
3.4 95% Confidence Intervals (CIs) and estimates of the median of the utilisation differences between simulation and execution in DC1 and DC2. Measurements are in percentages. 88

4.1 Simulation parameters. 107

5.1 AWS VM type definitions. 136

6.1 Network latencies between clients and cloud sites in milliseconds. 158

Chapter 1

Introduction

CLOUD computing services offer access to third party resources and infrastructure over the Internet. Cloud providers create and maintain large scale data centres to lease IT resources in a *pay-as-you-go* manner, thus realising the long-held dream of utility computing [44, 75]. From a business perspective, it is widely viewed as a disruptive economical model for renting preconfigured technical resources [101]. Being able to rent and use facilities on demand and to avoid upfront investments in hardware and licenses proves to be very enticing for enterprises in a dynamic and unstable business environment. Cost efficiency has long been touted as the primary benefit of cloud adoption [99]. However, recent investigations have found that cloud computing has a much further-reaching impact. It enables organisations to focus on their main lines of business and decreases the cost of experimenting with technologies thus encouraging innovation [99].

Cloud services can be classified in three main categories with respect to the provided IT resources [111], namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In the IaaS model, clouds provide computing resources such as storage, networks, and processing. PaaS cloud providers offer tools and services for application development and deployment. Finally, SaaS providers offer their clients complete applications like web-based calendars and office suites.

The cloud model can be seen as an extension of Grid computing, which also envisions access to a shared pool of computing resources. One major distinction between the two is the type of applications they usually host. Grids are used for resource intensive batch processing systems, while cloud applications are much more general purpose and can also include interactive online applications [75], most of which follow the standard 3-Tier architectural model and are divided into [2, 76, 142]:

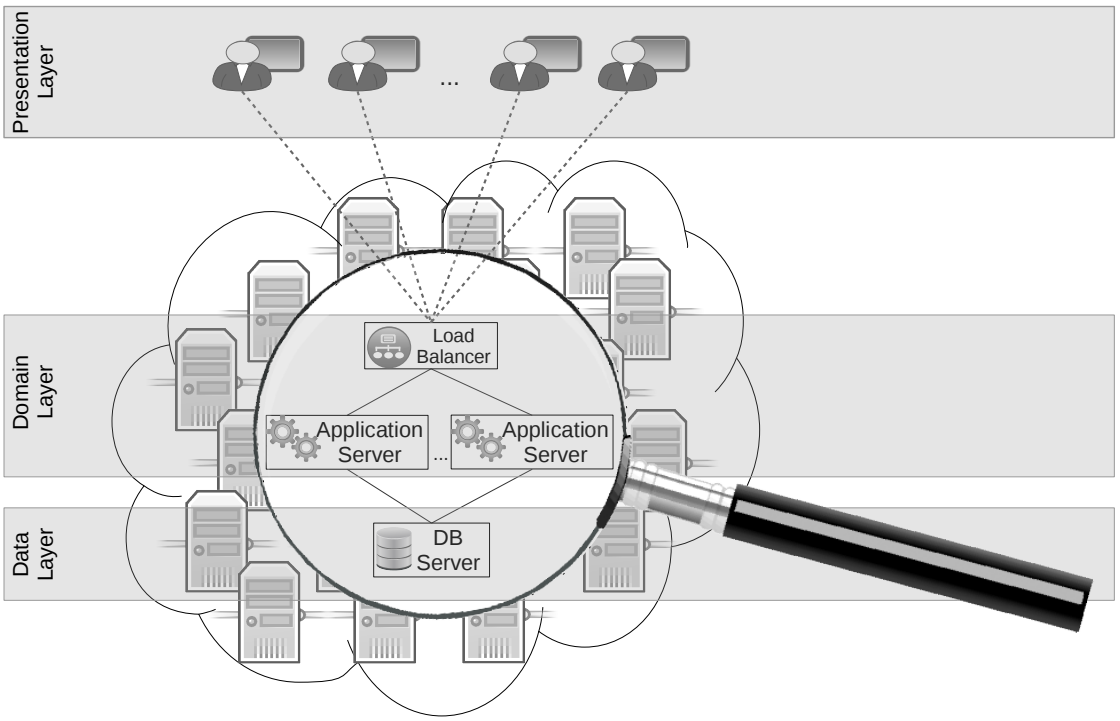


Figure 1.1: 3-Tier application in a single cloud data centre.

- **Presentation Layer** — represents the interface displayed to the user;
- **Business/Domain Layer** — implements the core application logic;
- **Data Layer** — handles access to the persistent storage.

This layering is logical. However, in the case of large scale applications it also becomes physical separation and each logical layer is deployed separately to increase the overall performance, maintenance, and scalability.

Figure 1.1 depicts a typical 3-Tier application deployed in a single cloud. The presentation layer is usually accessed through a web browser. The domain layer implements the essential application logic — e.g. management of items in a shopping cart in an online store, access to users' personal data etc. Traditionally, the domain layer is executed in one or several Application Servers (AS). They are deployed behind a Load Balancer (LB), which redirects the incoming requests among them. Servers "behind" the load balancer can be dynamically added and removed in response to a dynamic workload. The data layer facilitates access to the persistent storage. It is independent from the other two layers and in some cases it can even serve more than one application.

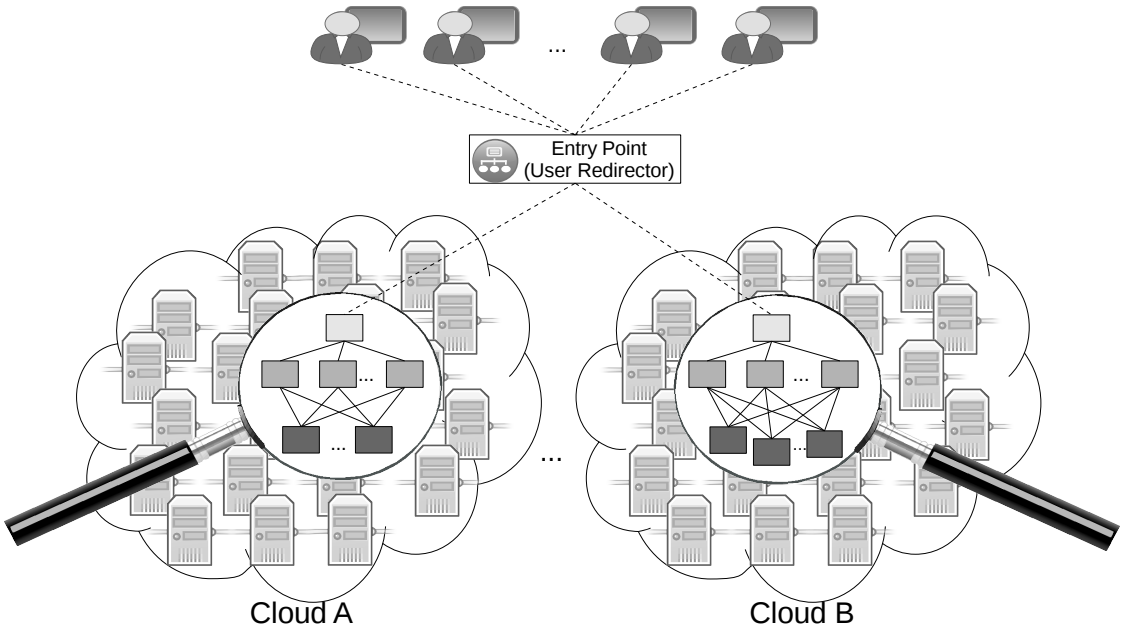


Figure 1.2: An interactive Multi-Cloud system.

The 3-Tier design in its different incarnations is a prevalent architectural model. Cloud data centres are becoming the preferred deployment environment for such applications. Enterprises hosting 3-Tier applications in cloud environments range from e-commerce businesses like *ebay* [67], which is hosted in a hybrid cloud [68], to government agency web sites, including the US Department of Treasury, hosted in Amazon EC2 [13].

However, the cloud deployment of such applications also raises several challenges for clients. A cloud service interruption may have a severe impact on clients who are left without access to essential resources [26], as highlighted by several recent cloud outages [15, 112]. Furthermore, for many applications the geographical location of the serving data centre is essential because of either legislative or network latency considerations. Lastly, it is desirable to avoid vendor lock-in and to be able to dynamically move to competitive providers to reduce the overall cost of the used resources.

To address such challenges, software engineers need to design for the cloud, not only to deploy in the cloud. Software applications need to be more scalable and fault tolerant so they can dynamically adapt to workload fluctuations and autonomously and timely address infrastructure failures. The use of multiple clouds (i.e. a Multi-Cloud) has been suggested as a potential approach to achieve this. In a Multi-Cloud environment, clients

use resources from multiple clouds to host application components without relying on any interoperability between the cloud providers. For interactive applications, usually there is an intermediate *Entry Point* redirecting users or their requests to the components in the individual clouds, as depicted in Figure 1.2.

Multi-Clouds hold the promise to facilitate interactive applications that are (i) legislation compliant, (ii) more resilient and cost efficient, and (iii) offer suitable Quality of Experience (QoE) to geographically dispersed end users. Despite some initial efforts in the area, building 3-Tier applications that meet these goals remains an open challenge. This is precisely the topic of this thesis. It investigates how a Multi-Cloud can be efficiently used as a deployment environment for interactive 3-Tier applications.

1.1 Motivation and Scope

The standard cloud computing model, where a client uses a single cloud site, has several limitations. A cloud service unavailability can leave thousands of customers relying solely on it without access to essential and paid for resources. In fact, according to a Berkeley's analysis service unavailability is the greatest obstacle for cloud adoption [26]. Relying on a single cloud site makes it impossible to provide adequate responsiveness and usability to clients distributed worldwide because of the network latency. Furthermore, many interactive applications have strict regulatory requirements as to where different users are served. For example, the Data Protection Directive of the European Union (EU) forbids the transfer of personal data to non-member countries, unless an adequate level of protection is ensured [70]. Similar laws exist in many other jurisdictions as well [39]. Therefore, no single cloud site can facilitate the requirements of a large scale interactive system serving customers worldwide and complying with various and often conflicting regulations.

These factors preclude the usage of multiple clouds (i.e. a Multi-Cloud) to achieve better Quality of Service (QoS), reliability, and flexibility. If there is a cloud site outage, users can be redirected to alternative ones. The implications from the outage of one of Amazon's data centres were very serious for customers who relied on that location only. In a post-mortem analysis Amazon advised their clients to design their applications to

use multiple data centres for fault tolerance [14]. Furthermore, only by using multiple clouds can an application provide low latency service to users worldwide and gain access to resources in multiple legislative regions in order to meet regulatory requirements. Besides, using multiple cloud sites and being able to freely transit workload among them allows enterprises to avoid vendor lock-in and take advantage of the dynamically diverging cloud prices. In case a provider changes a policy or pricing that impact negatively its clients, they could easily migrate elsewhere.

In order to work with multiple clouds, common industry standards and resource management programming interfaces must be used. Otherwise, one would have to program against the specific interfaces of each cloud provider, which would result in a brittle and difficult to extend system. Standardising PaaS and SaaS solutions of different providers has been an elusive goal because of the plethora of diverse functionalities they offer. On the other hand, IaaS cloud providers offer virtualised infrastructure in the form of a few abstractions like virtual machines, disk volumes, etc. Therefore, there are already de facto standards and mature services providing unified access to multiple IaaS cloud offerings. Hence, in this thesis we focus on IaaS Multi-Clouds, so we can take advantage of these technological advances.

Significant work has been done in the area of scaling the data layer across servers and data centres. Cattel [50] surveys more than 20 projects in the area of scalable distributed databases that balance differently between the consistency, availability, and partition tolerance requirements as defined in the CAP theorem [40, 41]. Furthermore, Google has revealed their database architecture which scales within and across geographically distributed data centres without violating transaction consistency [60]. Hence, persistent data distribution across data centres is outside the scope of this thesis. We focus on load balancing the incoming users across data centres given that the data is already appropriately distributed across the cloud sites.

1.2 Research Problem and Objectives

This thesis explores multiple aspects of resource provisioning and workload management (a.k.a. brokering) for 3-Tier applications in Multi-Clouds. More specifically it fo-

cuses on the following research question:

> *How to broker 3-Tier applications in a Multi-Cloud environment, considering Quality of Service (QoS) requirements in terms of:*
>
> - ***Network Latency Awareness*** *— end users should be served near their geographical location to experience better responsiveness.*
> - ***Pricing Awareness*** *— the overall costs for hosting should be minimised.*
> - ***Legislation/Policy Awareness*** *— legal and political considerations about where individual users are served should be honoured.*
> - ***Code Re-usability*** *— few changes to existing 3-Tier applications should be made. The technical overhead of moving an existing 3-Tier system to a Multi-Cloud should be minimal.*

To answer this question, first we need a formal performance model of Multi-Cloud 3-Tier applications. Such a model would be instrumental in the performance analysis and the development of various optimisation techniques. The main challenges in this respect are:

- Models are simplified representations of reality and almost always observe a certain level of inaccuracy. How to develop a model which balances between generality, ease of use, and accuracy?
- What stochastic processes to use when modelling the incoming workload?
- How to extract a model representation of an existing system or one under development?
- How to develop a simulator based on the model, which can be used for performance evaluations of different brokering techniques?

Secondly, we need a general architecture for brokering 3-Tier applications in Multi-Clouds. Hence, the following challenges arise:

- What should be the architectural components and their interactions in order to achieve the predefined goals?

- How to estimate the potential network latencies between users and all cloud sites, so we can redirect load efficiently?
- How to ensure existing 3-Tier applications can be migrated to a Multi-Cloud intact or with minimal changes?
- How to ensure acceptable overhead of the proposed architecture?
- How to automatically select the most suitable resource when autoscaling?
- How to conveniently model the regulatory requirements?

Based on these challenges, the following objectives can be outlined:

- Explore, analyse and systematise existing approaches for application brokering in multiple clouds. Classify them with respect to the aforementioned requirements.
- Formulate a performance model for 3-Tier applications in one and multiple clouds and implement a simulation environment based on it.
- Develop a flexible and general 3-Tier application brokering architecture.
- Propose an approach for network latency estimation between any incoming end user and the employed clouds.
- Develop an approach for efficient Virtual Machine (VM) selection upon scaling up within a cloud.
- Develop a flexible domain specific model for specifying regulatory rules for workload redirection.

1.3 Evaluation Methodology

The approaches proposed in this thesis were evaluated with realistic web application benchmarks in cloud environments and using industry standards as baselines. We used both the Rice University Bidding System (RUBiS) [16, 52, 149, 163] and CloudStone [59, 155] 3-Tier benchmarking applications. The RUBiS benchmark represents an e-commerce website similar to *ebay*, while CloudStone, which is a part of the CloudSuite benchmarking suite, is a social network website. We have extended the CloudStone benchmark to accommodate a load balancer and multiple application servers, which was

impossible with the original configuration. We have developed automated scripts and detailed documentation for CloudStone deployment [1] to facilitate the reproducibility of our test environment.

In some cases, executing repeated large scale experiments was not plausible and discrete event simulations were used instead. Thus, we developed a set of extensions to the CloudSim [46] simulator, which allow modelling and simulation of 3-Tier applications in a Multi-Cloud environment. We had to ensure that our simulations are truthful representations of the actual runtime behaviour. Thus, we conducted pairwise experiments with a benchmark and its corresponding simulation and compared the observed performances. Afterwards, we continue to use the newly developed simulator to perform large scale experiments.

1.4 Thesis Contributions

Following the previously defined research question and objectives, the **key contributions** of this thesis are:

1. A taxonomy and survey of the state-of-the-art cross-cloud application brokering techniques and architectures.

2. A formal performance model and a simulator for 3-Tier applications in one and multiple clouds:

   - An I/O performance model for IaaS cloud environments, defining how persistence storage access is shared among different applications and VMs on the same physical host.
   - Formal definition of a performance model of an end user session in an interactive 3-Tier application.
   - A parameterised stochastic process for modelling the pattern of incoming users.
   - A CloudSim based simulator which uses the aforementioned models.

[1] http://nikolaygrozev.wordpress.com/2014/06/02/advanced-automated-cloudstone-setup-in-ubuntu-vms-part-2/

- Experimental evaluation of the proposed models, by comparing the predicted (from the simulation) and the observed (from actual system execution) performances and utilisations.

3. An architecture and policies for Multi-Cloud resource provisioning and load distribution for 3-Tier applications:

   - Architecture definition in terms of architectural components and their interactions.
   - A technique for estimating network latency between end users and potential cloud sites using an Internet Performance Measurement (IEPM) service and a geolocation database.
   - An approach for estimating the cost for serving an end user within a given cloud site.
   - A workload redirection mechanism which heuristically optimises the cost and response delays without violating legislative and regulatory requirements.
   - Parameterised load balancing and autoscaling policies, which in conjunction optimise cost and resource utilisation within a single cloud site.
   - Evaluation through extensive discrete event simulations and comparison with the standard industry practices.

4. A novel approach for dynamic selection of Virtual Machines (VM) for application servers:

   - A method for automated workload change detection using Hierarchical Temporal Memory (HTM).
   - A dynamic VM performance utilisation prediction method using an Artificial Neural Network (ANN) and the aforementioned workload change information.
   - An approach for VM performance capacity estimation and normalisation.
   - An algorithm for dynamic VM selection when autoscaling, which uses the aforementioned techniques.

- Evaluation through experiments with a prototype in AWS and comparison with the standard industry practices.

5. A novel approach for regulations and latency aware load distribution across clouds:

   - A domain specific approach for modelling regulatory requirements in a rule-based format.
   - An efficient system for checking regulatory compliance using a rule inference engine.
   - An improvement of the aforementioned network latency estimation approach. It handles the case when a cloud provider can reuse the same IP address in multiple cloud sites.
   - An implementation of the system.
   - Performance evaluation using 10 cloud sites of 2 cloud providers located on 5 continents.

1.5 Thesis Organisation

Figure 1.3 shows the thesis structure in terms of chapters and their logical dependencies. Chapters 2 and 3 overview the related literature and propose a performance model and simulator respectively. They serve as a basis for Chapter 4, which introduces an overall architecture for 3-Tier application brokering across clouds. Chapters 5 and 6 are independent of each other. They introduce extensions to the architecture from Chapter 4, which improve the efficiency of the brokering mechanisms in a single cloud site and across clouds respectively. More specifically, the remainder of the thesis is organised as follows:

- Chapter 2 presents a taxonomy and survey of contemporary approaches and architecture for application brokering across clouds. It is derived from:

  – **Nikolay Grozev** and Rajkumar Buyya, "Inter-cloud Architectures and Application Brokering: Taxonomy and Survey", *Software: Practice and Experience*, John Wiley & Sons, Ltd, vol. 44, no. 3, pp. 369–390, 2014.

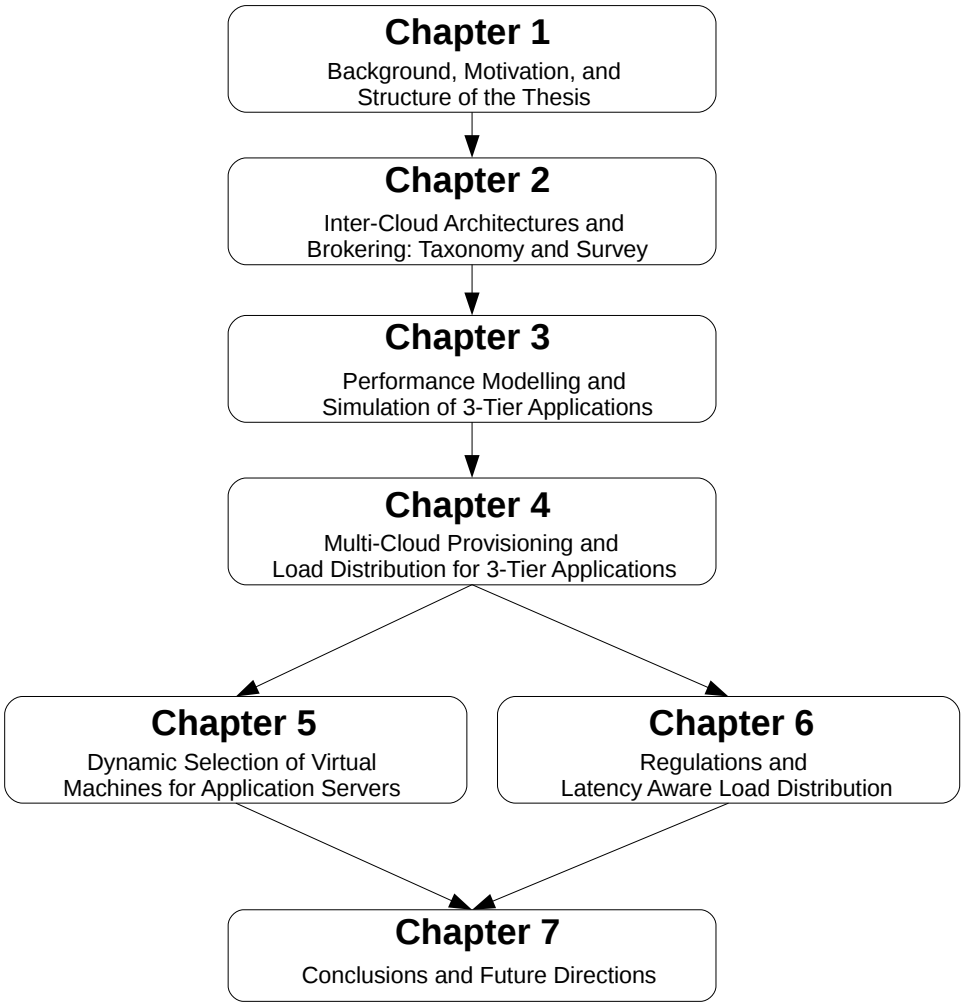


Figure 1.3: Thesis organisation.

- Chapter 3 introduces a performance model and a simulator for 3-Tier applications in one and multiple clouds. It is derived from:
  - **Nikolay Grozev** and Rajkumar Buyya, "Performance Modelling and Simulation of Three-Tier Applications in Cloud and Multi-Cloud Environments", *The Computer Journal*, Oxford University Press, vol. 58, no. 1, pp. 1–22, 2015.
- Chapter 4 proposes an architecture for Multi-Cloud resource provisioning and load distribution for 3-Tier applications. It is derived from:
  - **Nikolay Grozev** and Rajkumar Buyya, "Multi-Cloud Provisioning and Load Distribution for Three-tier Applications", *ACM Transactions on Autonomous and Adaptive Systems*, vol. 9, no. 3, pp. 13:1–13:21, 2014.

- Chapter 5 presents a method for dynamic selection of Virtual Machines for application servers using a combination of machine learning techniques. It is derived from:
  - **Nikolay Grozev** and Rajkumar Buyya, "Dynamic Selection of Virtual Machines for Application Servers in Cloud Environments", *Technical Report CLOUDS-TR-2016-1, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne*, February 5, 2016.
- Chapter 6 presents an architecture for regulations and latency aware load distribution of web applications in Multi-Clouds and presents its implementation. It is derived from:
  - **Nikolay Grozev** and Rajkumar Buyya, "Regulations and Latency Aware Load Distribution of Web Applications in Multi-Clouds", *The Journal of Supercomputing*, Springer (Under Review), 2015.
- Chapter 7 concludes the thesis, summarises its findings, and outlines directions for future work.

Chapter 2

Inter-Cloud Architectures and Brokering: Taxonomy and Survey

Although cloud computing itself has many open problems, researchers in the field have already made the leap to envision Inter-Cloud computing. Their goal is to achieve better overall Quality of Service (QoS), reliability, and cost efficiency by utilising multiple clouds. Inter-Cloud research is still in its infancy and the body of knowledge in the area has not been well defined yet. In this chapter, we propose and motivate taxonomies for Inter-Cloud architectures and application brokering mechanisms. We present a detailed survey of the most prominent developments from both academia and industry (23 projects) and we fit each project onto the proposed taxonomies. We discuss how the current Inter-Cloud environments facilitate brokering of distributed applications across clouds considering their non-functional requirements. Finally, we analyse the existing works and identify open challenges and trends in the area of Inter-Cloud application brokering.

2.1 Introduction

THE first academic publications about Inter-Clouds appeared a couple of years after the advent of cloud computing and include the following seminal works — [33, 43, 53, 94, 147]. The term Inter-Cloud has been described as a cloud of clouds [96] and a formal definition is provided in Section 2.2. Essentially, an Inter-Cloud allows for the dynamic coordination and distribution of load among a set of cloud data centres — see Figure 2.1.

In essence, the main value of an Inter-Cloud environment for cloud customers is that they can diversify their infrastructure portfolio in terms of both vendors and location.

This chapter is based on the following publication: ***Nikolay Grozev*** *and Rajkumar Buyya, "Inter-cloud Architectures and Application Brokering: Taxonomy and Survey", Software: Practice and Experience, John Wiley & Sons, Ltd, vol. 44, no. 3, pp. 369–390, 2014.*

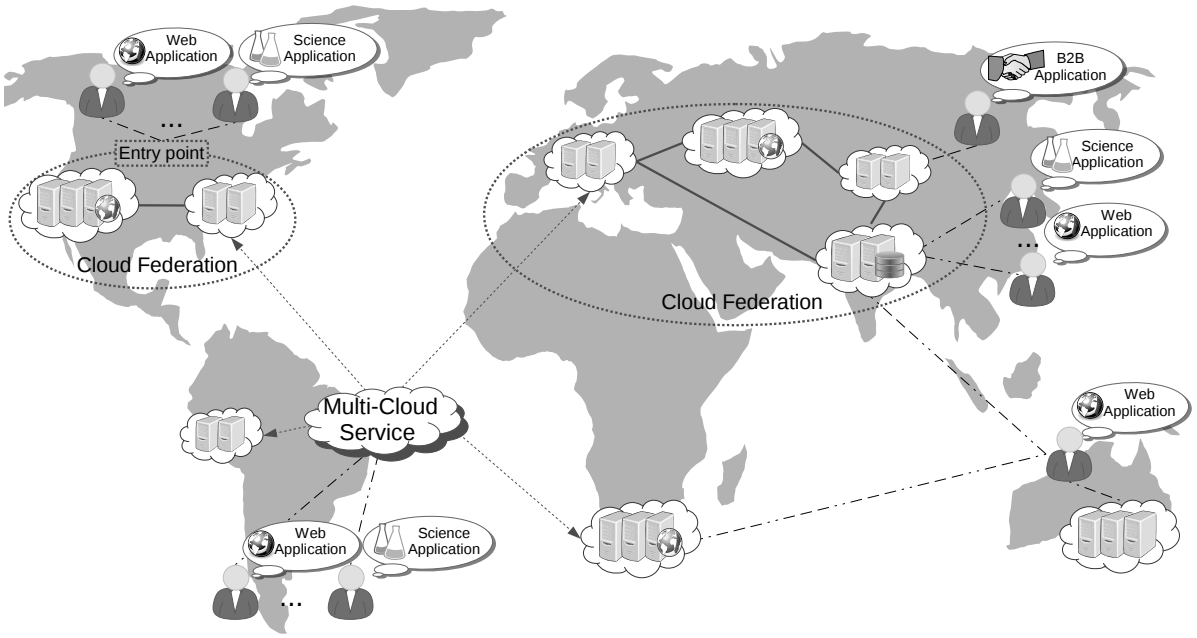


Figure 2.1: An overview of Inter-Cloud approaches and use cases.

Thus, they could make their businesses more adaptable to vendors' policy and availability changes and easily expandable in new legislative regions.

Cloud service providers may also have significant incentives from participating into an Inter-Cloud initiative. A paramount idea of cloud computing is that a cloud service should deliver constant availability, elasticity and scalability to meet the agreed customers' requirements [101]. A cloud provider should ensure enough resources at all times. But how much is enough? Workload spikes can come unexpectedly and thus cloud providers need to overprovision resources to meet them. Another issue is the huge amount of data centre power consumption [31, 44]. Keeping an excess of resources in a ready to use state at all times for coping with unexpected load spikes leads to increased power consumption and cost of operation. Cloud providers' benefits can be summarised as follows:

- **Expand on demand.** Being able to offload to other clouds, a provider can scale in terms of resources like cloud-hosted applications do within a cloud. A cloud should maintain in a ready to use state enough resources to meet its expected load and a buffer for typical load deviations. If the workload increases beyond these limits, resources from other clouds can be leased [43].

- **Better SLA to customers.** Knowing that even in a worst case scenario of data centre outage or resource shortage the incoming workload can be moved to another cloud, a provider can offer better Service Level Agreements (SLA) to customers.

However, achieving all these benefits for both cloud providers and clients should be done without violating applications' requirements. Appropriate application brokering (consisting of provisioning and scheduling) should honour the requirements in terms of performance, responsiveness, and legal considerations. Existing approaches to achieve this vary in terms of architecture, mechanisms and flexibility.

In this chapter, we investigate and classify Inter-Cloud environments and application brokering mechanisms. We focus on identifying and analysing the coarse-grained requirements and recent developments in Inter-Cloud brokering of distributed applications. With this, we perform analysis of the status quo and identify trends and open issues in the area. To the best of our knowledge, this novel area of research has not been systematically surveyed before.

Since the area of Inter-Clouds is novel and there is ambiguity about some terms, in Section 2.2 we discuss and define the main ones. In Section 2.3, we motivate and introduce an architectural taxonomy of existing Inter-Cloud developments. Section 2.4 classifies the existing approaches for brokering applications across clouds. In Section 2.5, we classify existing distributed applications, and for every major class of applications we investigate what are the requirements for Inter-Cloud brokering. In Section 2.6, we review 23 major developments in the area fitting them into the previous taxonomies and discussing their capabilities to facilitate Inter-Cloud application brokering. Section 2.7 provides an analysis of the state of the art. The final Section 2.8 summarises the discussion.

2.2 Definitions

Cloud computing is a novel area of research and still faces certain terminological ambiguity. The area of Inter-Clouds is even newer and many works use several terms interchangeably. In this section we elaborate on their meaning.

Inter-Cloud computing has been formally defined as [81]:

> "A cloud model that, for the purpose of guaranteeing service quality, such as the performance and availability of each service, allows on-demand reassignment of resources and transfer of workload through a [*sic*] interworking of cloud systems of different cloud providers based on coordination of each consumer's requirements for service quality with each provider's SLA and use of standard interfaces."

In the rest of this chapter we will adhere to this definition. The seminal works on Inter-Clouds by Buyya et al. [43] and Bernstein et al. [33] also implicitly express similar definitions. Buyya et al. emphasise the just-in-time, opportunistic nature of the provisioning within an Inter-Cloud that allows for achieving QoS and QoE (Quality of Experience) targets in a dynamic environment [43]. The term *Cloud Fusion* has been used by Fujitsu Laboratories to denote a similar notion [150].

This definition is generic and does not specify who is initiating the Inter-Cloud endeavour — the cloud providers or the clients. Also, it does not specify whether cloud providers collaborate voluntarily to form an Inter-Cloud or not. Two other terms are used throughout the related literature to differentiate between these — *Federation* and *Multi-Cloud*. A Federation is achieved when a set of cloud providers voluntarily interconnect their infrastructures to allow sharing of resources among each other [25,73,147]. The term Multi-Cloud denotes the usage of multiple, independent clouds by a client or a service. Unlike a Federation, a Multi-Cloud environment does not imply volunteer interconnection and sharing of providers' infrastructures. Clients or their representatives are directly responsible for managing resource provisioning and scheduling [73]. The term *Sky Computing* has been used in several publications with similar meaning [94,137]. Both Federations and Multi-Clouds are types of Inter-Clouds. We discuss them further in Section 2.3.

Another term used in the related literature is *Hybrid Cloud*. It has been defined as a composition of two or more different cloud infrastructures — for example a private and a public cloud [111]. Thus, a hybrid cloud is a Multi-Cloud that connects miscellaneous clouds in terms of their deployment models. Hybrid clouds are often used for *cloud bursting* — the usage of external cloud resources when local ones are insufficient.

Throughout the literature, the term *Inter-Cloud broker* has been used with different

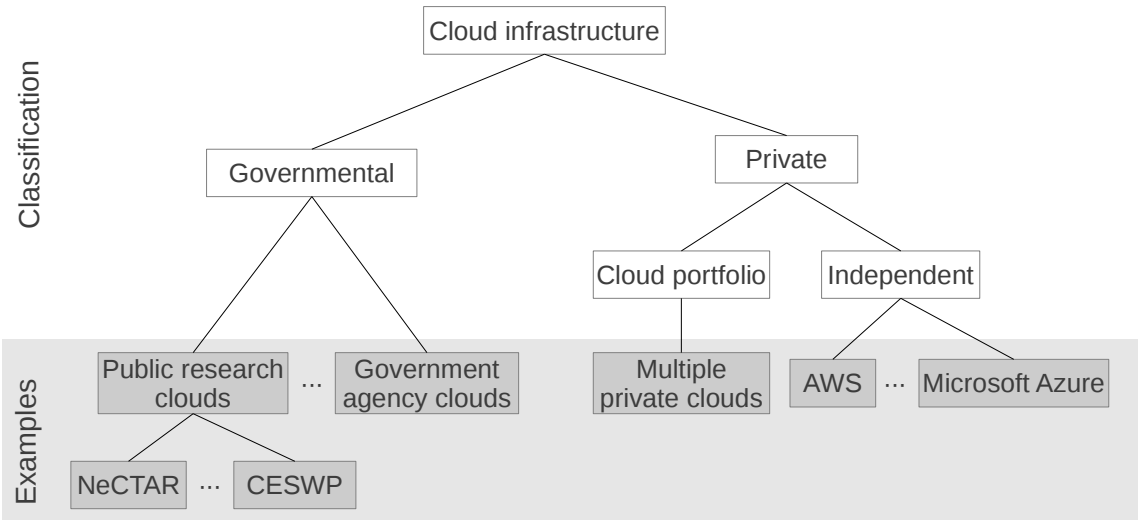


Figure 2.2: Classification of cloud infrastructures.

meanings. In most cases, it means a service that acts on behalf of the client to provision resources and deploy application components [43,73,105]. We adhere to this general idea and define an *application broker* as an automated entity with the following responsibilities:

- Automatic resource provisioning and management across multiple clouds for a given application. Typically, this would include allocation and deallocation of resources (e.g. VMs and storage).
- Automatic deployment of application components in the provisioned resources.
- Scheduling and load balancing of the incoming requests to the allocated resources.

2.3 Architectural Taxonomy

By definition, Inter-Cloud computing is an endeavour which implies interconnecting multiple cloud providers' infrastructures. The extent to which cloud providers would voluntarily lend their infrastructure within an Inter-Cloud depends on the political and financial incentives they have to do so. Figure 2.2 depicts how cloud infrastructures can be classified based on their ownership.

On the first level, we differentiate between *Governmental* and *Private* cloud infrastructures. These can be described as:

- **Governmental** — owned and utilised by a government or non-profit institution. Examples for this are science community clouds like Australia's National eResearch Collaboration Tools and Resources (NeCTAR) [118] and Canada's Cloud-Enabled Space Weather Modelling and Data Assimilation Platform (CESWP) [54].

- **Private** — owned by a private organisation.

Private clouds can be further classified as:

- **Cloud portfolio** — when the cloud is a part of a portfolio of clouds belonging to the same organisation. Examples for this are multiple private clouds belonging to a corporation.

- **Independent** — separate cloud infrastructure that is not a part of a portfolio of clouds.

Given this classification, we argue that *Independent* private clouds are less likely to participate voluntarily in an Inter-Cloud initiative. Such cloud providers will be reluctant to scale/transit their workload in the data centres of their competitors. Also, such providers may not be willing to provide access through unified APIs to their services in a federated market place because that would allow their customers to migrate easily and dynamically to competitors. This can be achieved by using specialised third party services and application programming interfaces (APIs), which will be discussed in details later.

For example, despite their data centre outages leaving customers without access to services for substantial periods of time, none of the leading commercial providers (like Amazon, Google and Microsoft) has announced plans to utilise external cloud resources to avoid future outages.

On the other hand, *Governmental* clouds are very likely to participate voluntarily within an Inter-Cloud among each other since that could benefit the quality of the overall public service. Similarly, *portfolio clouds* are likely to form an Inter-Cloud initiative among each other since they are not directly competing with each other and have clear incentives to collaborate.

Based on this observation, we can broadly classify Inter-Clouds as:

- **Volunteer federation** — when a group of cloud providers voluntarily collaborate with each other to exchange resources. As identified, this type of Inter-cloud is mostly viable for governmental clouds or private cloud portfolios.

- **Independent** — when multiple clouds are used in aggregation by an application or its broker. This approach is essentially independent of the cloud provider and can be used to utilise resources from both governmentally and private clouds. Another term used for this is *Multi-Cloud*.

From architectural perspective *Volunteer federations* can be further classified as:

- **Centralised** — in every instance of this group of architectures, there is a central entity that either performs or facilitates resource allocation. Usually, this central entity acts as a repository where available cloud resources are registered, but may also have other responsibilities like acting as a market place for resources.

- **Peer-to-Peer** — in the architectures from this group clouds communicate and negotiate directly with each other without mediators.

Independent Inter-Cloud developments can be further classified as:

- **Services** — application provisioning is carried out by a service which can be hosted either externally or in-house by the cloud clients. Most such services include broker components in themselves. Typically, application developers specify an SLA or a set of provisioning rules and the service performs the deployment and execution in the background, in a way respecting these predefined attributes.

- **Libraries** — often custom application brokers that directly take care of provisioning and scheduling application components across clouds are needed. Typically, such approaches make use of inter-cloud libraries that facilitate the uniform usage of multiple clouds.

The whole taxonomy of developments is depicted in Figure 2.3. Figure 2.4 depicts the architectures from the taxonomy. In Section 2.6, we discuss in details many examples of developments in all these areas. Note that the discussed herein classifications are orthogonal to the general classification of cloud computing services as Software as a Service

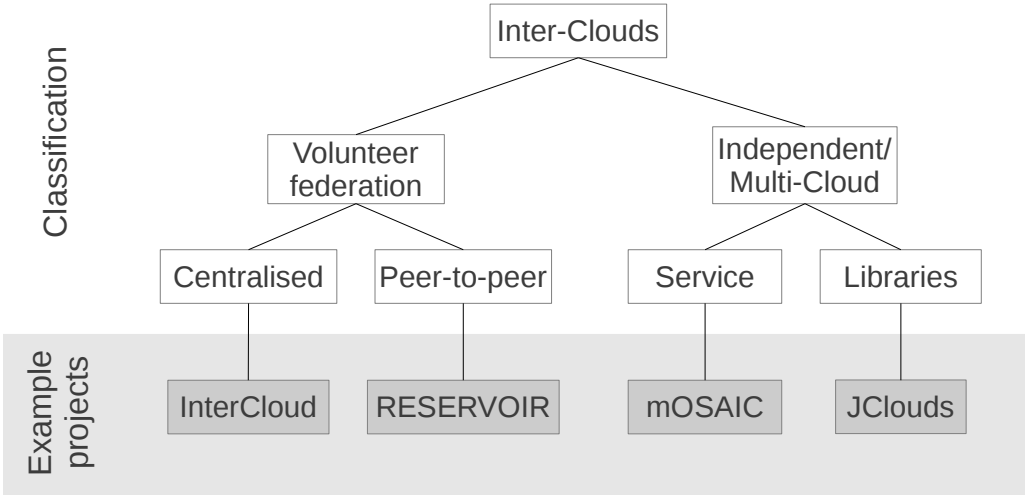


Figure 2.3: Architectural classification of Inter-Clouds.

(SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) as defined by NIST [111]. For example, cloud providers within a federation can exchange resources at the infrastructure level (e.g. VMs), but at the same time provide PaaS to clients. Also, in theory a multi-cloud service may access IaaS, PaaS, or SaaS cloud services on behalf of its clients.

2.4 Taxonomy of Inter-Cloud Application Brokering Mechanisms

There are several mechanisms for implementing application specific brokering as depicted in Figure 2.5. At the first level, we differentiate as to who is responsible to provide implementation of the application broker. Brokers using *Externally managed* mechanisms usually are not hosted in-house and are a part of the system entities that facilitate the Inter-Cloud. In the case of *Multi-Cloud Services*, application brokers are often part of the service providing access to the set of clouds. In the case of *Volunteer Federations*, application brokering is implemented either in a centralised entity or by the cloud providers. In essence, *Externally managed* brokers are transparent to the developers and provide some "hooks" for application specific logic.

Based on the way application specific logic is specified, we can further classify *Externally managed* brokering mechanisms as:

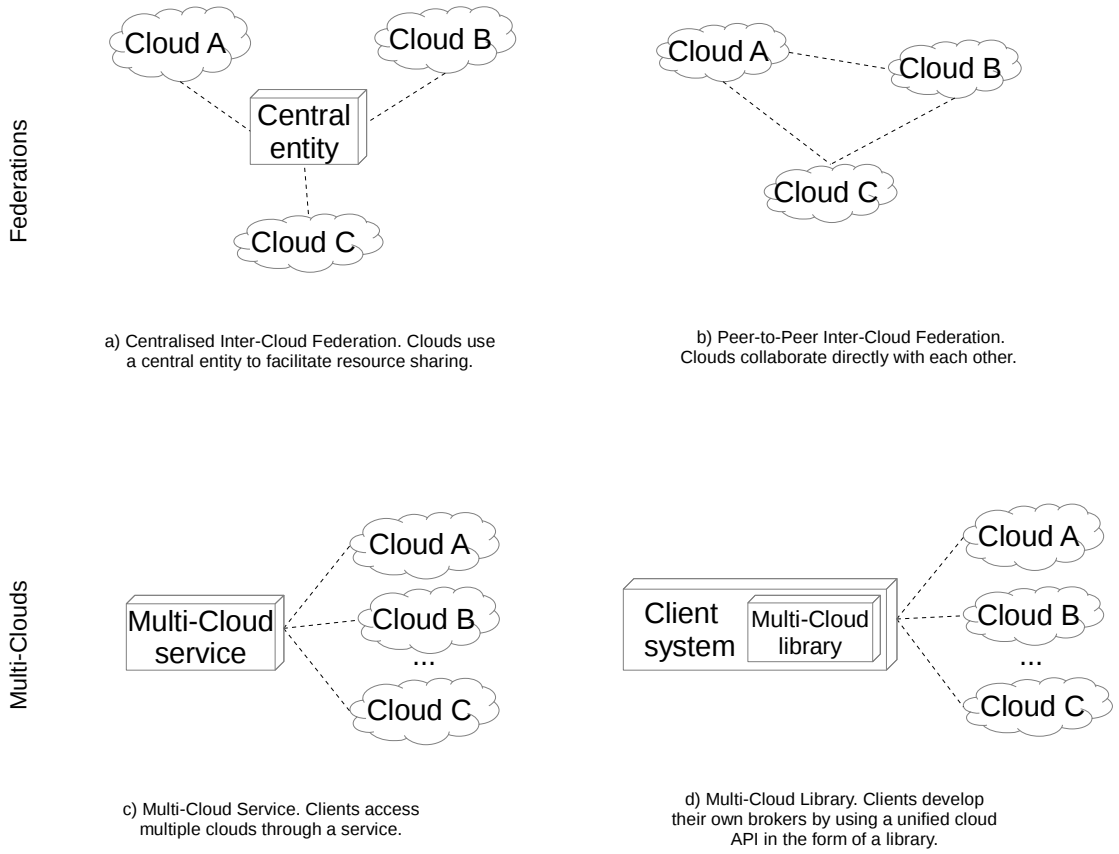


Figure 2.4: Inter-Cloud architectures.

- **SLA based** — application developers specify the brokering requirements in an SLA in the form of constraints and objectives. The cloud provider or the Inter-Cloud service acting on behalf of the client decides on brokering approach honouring the specified SLA.

- **Trigger-Action** — the application developers specify a set of triggers and associate one or more actions to each of them. A trigger becomes active when a predefined condition considering the externally visible application performance indicators becomes true. An action is executed when a correspondent trigger becomes active. Actions usually include scale up/down or other provisioning activities. For example, a trigger may become active if the number of active TCP/IP sessions becomes more than 50, or if the memory consumption of an allocated VM exceeds 512MB. An associated action may be the allocation of a new VM.

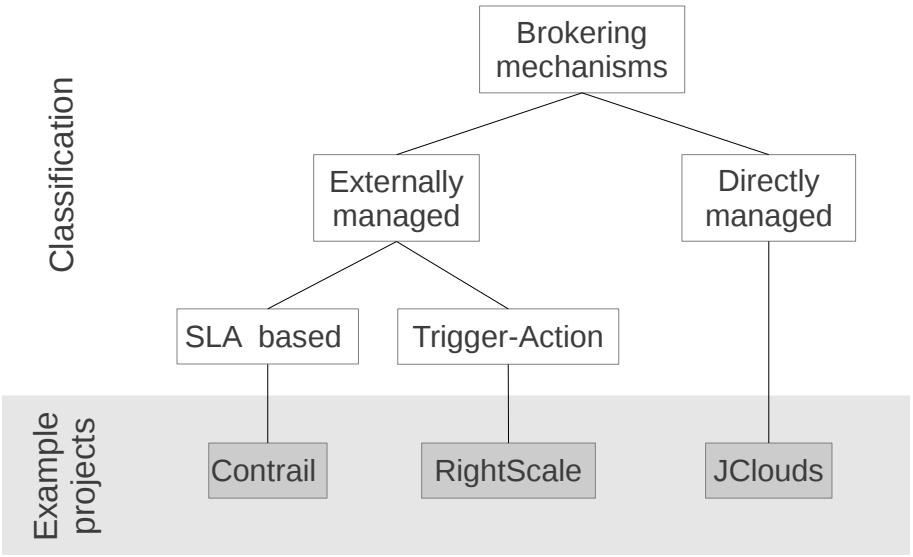


Figure 2.5: Application brokering mechanisms.

The main benefit of the *SLA based* brokering approaches is that they are completely transparent to the application developers. The provider or the service takes care of the provisioning behind the scenes. However, in *SLA based* brokering clients do not have direct control on how their applications are provisioned across clouds. Thus, there needs to be a certain level of trust between the two sides. Also, in order to specify the provisioning requirements of an application, one needs mature SLA specification formalisms and services. In a report from The Cloud Standards Customer Council (CSCC), it was emphasised that SLA contracts offered by current cloud providers are immature [62]. Often clients are offered only non-negotiable standard SLA contracts, thus limiting their ability to define application specific clauses. Most such contracts only specify performance related clauses and do not allow for provisioning restrictions — e.g. which location to use for data storage. Thus, the adoption of *SLA based* brokering approaches depends on the advancements of providers' and mediators' SLA offerings.

The *Trigger-Action* approach is less transparent than the *SLA based* one, since application developers need to specify the exact scalability and provisioning rules. This gives a more fine grained control about how the application behaves.

The *Directly managed* brokering mechanisms are mostly used when there is no mediator between the application and the set of utilised clouds. *Directly managed* brokers are hosted separately and need to keep track of the performance characteristics of the appli-

cation themselves. It is the responsibility of the application developers to develop such brokers in a way that meets the availability and dependability requirements.

2.5 Application-Centric Perspective to Inter-Clouds

Clouds as a deployment environment are general purpose and cover the whole spectrum of distributed applications. Cloud providers have always delivered computing resources to enterprises. Thus, many database centric enterprise applications are deployed within clouds. Being an inherently distributed environment, clouds are also suitable for standard Grid and Cluster job-based applications. Research community clouds like Australia's NeCTAR [118] and Canada's CESWP [54] have made the use of clouds for resource intensive jobs apparent, although there are still concerns about the worse performance of virtualised cloud clusters compared to physical ones as reported by Jackson et al. [90]. Developments like EC2 Cluster Compute Instance [12] are supposed to mitigate such issues. It is logical to expect that the same spectrum of applications will be targeted for the Inter-Cloud.

In this section, we perform a taxonomic analysis to identify classes of distributed applications with similar characteristics. Then, we continue to identify the common non-functional requirements of these application classes with respect to the Inter-Cloud environment in which they are deployed.

2.5.1 Taxonomy of Inter-Cloud Applications

Figure 2.6 depicts the taxonomy of typical distributed applications deployable in Inter-Cloud environments that will be discussed from now on. At the first level, we differentiate between *Batch processing* and *Interactive* applications. *Batch processing* applications allow the users to submit and execute jobs, which then run to completion without further user input. Thus, they can also be thought of as job-based applications. *Batch processing* applications can be further classified as allowing *Singular* or *Periodical* jobs.

Singular jobs are executed only once, unless they are rerun by the users or automatically rescheduled upon failure. This branch of the taxonomy represents typical Grid and Cluster resource intensive applications. Such applications are used by research, industry,

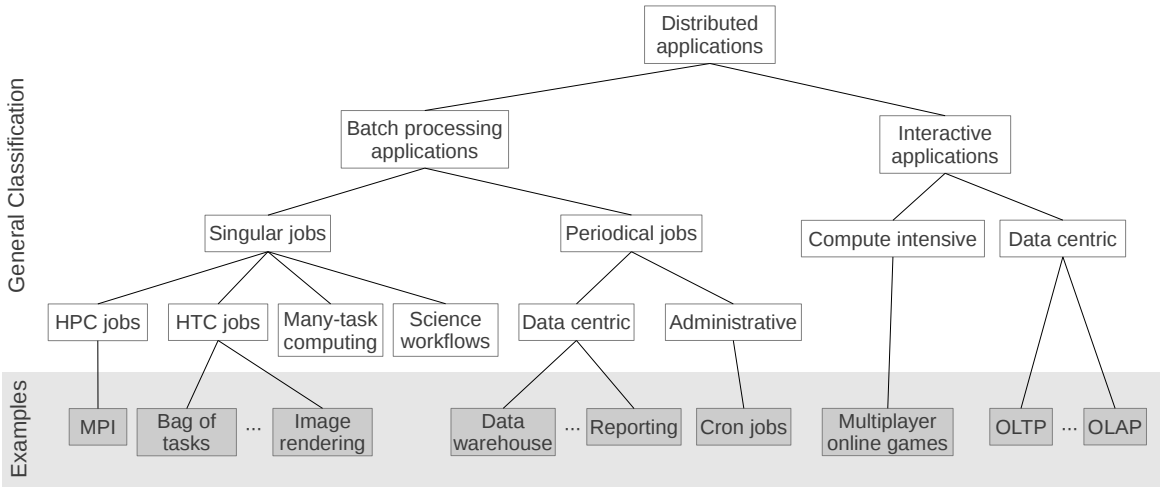


Figure 2.6: Classification of distributed applications.

and military organisations to solve optimisation models, build complex simulations, etc. Most of the applications in this branch fall into the High Performance Computing (HPC), High Throughput Computing (HTC) or Many-task Computing (MTC) categories. Raicu et al. defines these three types of applications and discusses their differences and similarities [141]. *Scientific workflows* (further discussed by Barker and Van Hemert [29]) are also a kind of *Singular* jobs, although they are composed of multiple subjobs/tasks, since they appear to the user as a single unified job.

Singular jobs typically require fast connection between the processing nodes and are less likely to be deployed across clouds. Exceptions to this are HTC applications, where independent long running tasks may be scheduled across clouds. Another reason for scheduling a job in multiple clouds is fault tolerance — if a job fails in one cloud environment it may succeed in another.

Periodical jobs are repeatedly executed over a time period. Most of them are *Data centric* and perform long running tasks over huge enterprise databases. Periodic Extract Transform Load (ETL) jobs in Data warehouses, reporting jobs, and Big Data analytical jobs are examples of this. *Data centric* periodical jobs are of great significance for enterprises with large scale information systems. They are often scheduled to run at nights or weekends to ensure smooth business operation during working hours. *Administrative* periodical jobs are relatively simple and take care of maintaining the system in a healthy and running state. Examples are scripts that periodically consolidate log files from all

nodes or release unneeded resources across nodes.

Data Centric Periodical jobs typically need to be executed close to where the corresponding data is stored. Thus, Inter-Cloud scheduling and provisioning are sensible when the persistent data is dispersed in multiple clouds.

Interactive applications are also known as *Online* applications. Unlike the *Batch processing* ones, they imply constant interaction and input from the user. Most interactive distributed systems are *Data centric*. That is, their main goal is to facilitate user access to a persistent storage (most often a relational database). Typical examples are Online Transaction Processing (OLTP) applications. Another group of *Interactive* applications are the *Compute Intensive* ones. Examples for such are multilayer online games.

Unlike *Batch processing* applications, *Interactive* ones need to be constantly available and thus would benefit the most from using multiple clouds. Also, many online applications serve users from around the world and thus would benefit from the geographical diversity of an Inter-Cloud's data centres.

2.5.2 Requirements for Inter-Cloud Environments

As discussed, the main benefit from the clients' perspective of an Inter-Cloud environment is the diversification in terms of vendors and locations. It is however important that this is done without violating the non-functional application requirements. Not all Inter-Cloud environments allow for such control. Thus, in the rest of this section we define several requirements for Inter-Cloud environments and discuss for which of the main applications classes these are important. Table 2.1 summarises the discussion. We only focus on requirements that are essential for the objectives of this thesis. For a more detailed discussion of the open challenges in interconnected cloud environments, the reader is referred to the work of Toosi et al. [161].

Data Location Awareness

Ideally, the persistent data and the processing units of an application should be in the same data centre (even on the same rack) and should be connected with a high speed network. If they reside on remote clouds, the performance penalty could be grave.

This consideration is especially important for interactive applications and periodical jobs using huge databases. Actually, in this case the database considerations are becoming a driving factor in Inter-Cloud brokering. The database often becomes the performance bottleneck of a distributed enterprise application. Additional approaches like distributed caching and sharding are often used to mitigate to some extent the performance issues of traditional databases [61].

In recent years, there has been a trend to move away from standard relational databases to the so-called NoSQL and NewSQL ones. There has been significant amount of developments in these fields (Cattel reports on more than 20 projects [50]). The term NoSQL denotes a broad range of non-relational database developments. In essence, the NoSQL approach aims at balancing between consistency and partition tolerance following the result of the famous CAP theorem [40, 41], while NewSQL approaches keep to standard relational approaches but discourage or ban performing certain inefficient operations [50].

Many enterprise systems will need to use several types of data sources simultaneously. For example, payment information may be stored in a transactional relational database to ensure consistency, but other data like catalogues of products may be stored in a NoSQL storage for better performance. This phenomenon has been termed *Polyglot persistence* by several practitioners [77, 100].

But why is this important in terms of Inter-Cloud brokering? An inter-cloud broker of a data-centric application is essentially responsible for allocating resources on multiple clouds and scheduling incoming requests to these clouds appropriately. Developers need the flexibility to implement brokering policies considering the structure and the location of distributed persistent data and cache servers. Thus, a successful broker should be well integrated with its application and should get feedback from it regarding the structure and location of data.

Most Singular job-based applications are compute intensive programs and thus avoid substantial I/O operations. Some of them, however, work on datasets that have been extracted in advance for example by conducting experiments. In such cases, the data is usually stored in standalone persistent storage — e.g. relational database or a data file. Typically, either the data is replicated to the node running the job or the job is scheduled

on a node near the data. The latter is preferred when the dataset has significant size. There has been substantial work in the last decades on data aware scheduling of singular jobs in Grids and it has been well summarised by Dong and Akl [65]. To apply these approaches in an Inter-Cloud environment, the location of the data (or its replicas) needs to be known.

Geo-location Awareness

Besides database considerations, the location of the incoming requests is also important to consider when brokering interactive applications. Ideally, requests should be scheduled near the geographical location of their origin to achieve better performance. This is not usually considered when brokering jobs, since the users submit the jobs only once and do not interact with the system until they finish. In interactive systems, however, network latency caused by geographical distances may significantly hurt the user experience.

Pricing Awareness

A common consideration for all types of applications deployed in multiple clouds is the pricing. Different providers have different pricing policies and mechanisms and minimising the overall hosting price is a non-trivial task. An application broker would need detailed and up to date information about providers' prices and policies to perform fiscally efficient provisioning.

Legislation/Policy Awareness

For some applications, the application broker should take into account legislative and political considerations upon provisioning and scheduling. For example, such a broker could avoid placing part of the data outside a given country or can avoid using cloud services of a specific vendor.

This is a typical requirement in both interactive and batch processing data centric applications. In rare cases, this can be a requirement for singular job applications as well — e.g. when performing research on a dataset of medical records.

Table 2.1: Typical Inter-Cloud brokering requirements.

Awareness	Job based		Interactive applications	
	Singular Jobs	Periodical Jobs	Compute Intensive	Data Centric
Data Location	✓	✓		✓
Geo-location			✓	✓
Pricing	✓	✓	✓	✓
Legislation/ Policy	✓	✓		✓
Local Resources	✓	✓	✓	✓

Local Resources Awareness

Another consideration that is common to all application types is the usage of local in-house resources with higher priority than external ones. That is, cloud bursting should be allowed only if the local resources are insufficient or unsuitable in terms of geographical location, proximity to data, or operational costs. Furthermore, some applications can only be hosted in private infrastructure for security reasons. Deploying software components and data in external infrastructure may not be allowed in some cases. In fact, local resources awareness can be considered a special case of policy and pricing awareness. We list it separately because it is an often recurrent characteristic.

2.6 State Of The Art in Inter-Clouds

In this section, we summarise several state-of-the-art Inter-Cloud developments. They were identified after an extensive analysis of the related literature and include both academic an industry projects. Each of the following subsections discusses the projects that fall into one of the architectural groups introduced in Section 2.3. Besides a technical summary for each project, we also discuss where it fits within the introduced taxonomies and how it addresses the requirements from Section 2.5.

2.6.1 Centralised Federated Inter-Clouds

InterCloud

The *InterCloud* [43] project developed at the University of Melbourne is one of the first initiatives in the field. It extends previous efforts form the *InterGrid* project to allow the sharing of resources across cloud providers [64]. The proposed architecture is centralised and is built around a central entity called *Cloud Exchange* (CEx). In essence, CEx acts like a marketplace where clouds can sell resources. The buyers may be other clouds or application brokers. Thus, the architecture is clearly market oriented and facilitates pricing aware application brokering.

Each cloud data centre participating in an *InterCloud* federation needs to have an agent called *Cloud Coordinator* (CC) installed. *Cloud Coordinators* manage the federation memberships of the clouds by communicating with CEx on their behalf [43]. An architecture of an extensible *Cloud Coordinator* has been further discussed by Calheiros et al. [45].

To serve as a federation wise marketplace, the *Cloud Exchange* (CEx) maintains an information registry of the clouds' resources. The *Cloud Coordinators* of the participating providers periodically update their details within this information registry [43]. Geographical location of resources can be maintained within CEx and thus location aware brokering is possible. Brokering in *InterCloud* can be either *SLA based*, when a cloud provider acts on behalf of the client, or *Directly managed*, when clients directly buy and use resources from CEx.

Contrail

The architecture of the European project *Contrail* [49] is built around a centralised composite entity that acts as a single entry point to a federation of cloud providers. It is responsible for periodically observing the states of the cloud providers and facilitating a federation wide SLA. It also provides single sign on, so that users need to authenticate only once to work with the entire federation. A special architectural component called *Federation Runtime Manager* (FRM) is dedicated to map users' requests to cloud resources. It implements cost and performance optimisation heuristics and ideally should have access to the geographical location and other meta information about every cloud

provider [49]. Application brokering is achieved by specifying a detailed application SLA. It is the responsibility of the FRM module to provision in accordance with the SLA and to minimise the costs. However, the documentation is not specific whether users are allowed to specify geographical, legislative, and data location constraints in the SLA.

The *Adapters Layer* architectural component constitutes of adapters for every cloud provider in the federation. The adapters facilitate the communication between the federation management components and the clouds. The adapters can be classified as:

- **Internal adapters** — for clouds running the Contrail software. These are called Contrail clouds. Contrail clouds allow for the federation management components to reserve and configure their resources [49].
- **External adapters** — for clouds that do not run the Contrail software.

When using *External adapters*, the cloud providers themselves are agnostic of the federation — they do not voluntarily participate in it. Thus, in terms of the presented architectural taxonomy, a Contrail federation using *External adapters* is considered a Multi-Cloud and will be discussed in a later section. On the other hand, a Contrail federation using *Internal adapters* is considered a centralised federation. In other words, *Contrail* supports both types of architectures. It even supports a hybrid architecture that utilises both clouds participating voluntarily within a federation and federation agnostic clouds.

Dynamic Cloud Collaboration (DCC)

In the centralised federation architecture *Dynamic Cloud Collaboration* (DCC), the central entity that facilitates the federation is one of the clouds called *primary cloud provider* (pCP). This is the first cloud provider in the federation — the one that actually established it [84, 85]. The other cloud providers are called *collaborating clouds*. The primary cloud (pCP) maintains a registry of the services of the collaborating clouds. Application brokering is done via SLA contract with pCP. Cloud clients submit requests to the pCP, which based on the requests' characteristics allocates resources within the federation. It has not been specified whether clients are allowed to declare requirements about the geographical location of the allocated resources. However, there has been work on the facilitation of a cloud market place where auctions for resources are held [85].

Federated Cloud Management (FCM)

The *Federated Cloud Management* (FCM) architecture relies on a generic repository called *FCM Repository* to store virtual appliances for all federated services [108]. It is replicated to the native repositories of the different IaaS providers. Clients interact only with the *Generic Meta-Broker Service* (GMBS) describing the requested service and, as far as they are concerned, further provisioning and scheduling is transparent [108].

For every IaaS provider in the architecture, there is a correspondent broker called *CloudBroker* managing the allocation and deallocation of VMs and dispatching incoming application calls to appropriate VMs. GMBS has access to the *FCM Repository* and can communicate with the *CloudBroker* components of the federated clouds. When GMBS receives a request from a user, it performs matchmaking between the request and an appropriate *CloudBroker*. The matchmaking is based on information from the *FCM Repository* and runtime metrics provided by the *CloudBrokers*.

Each *CloudBroker* maintains an internal queue of incoming application calls and a separate priority queue for every virtual appliance. VM queues represent the resources that can serve a virtual appliance related service call. The priority of each VM queue is based on the currently available requests in the queue, their historical execution times, and the number of running VMs. Based on the VM queues, the *CloudBroker* needs to perform appropriate VM creation and destruction. A *CloudBroker* also handles the aforementioned queue of incoming calls by redirecting them to the VMs created as a result of the management of the VM queues.

Application brokering in FCM is done transparently to the client. In the architecture definition nothing has been said about user-level control over the location of the used resources and how cost optimisation can be achieved [108]. We could speculate that location specific requirements can become a part of the SLA between the clients and GMBS. In a separate work, Kecskemeti et al. extend the FCM approach to facilitate self-adaptable and autonomous management of the federation [95]. One of their goals is to achieve resource optimisation (and consequently costs efficiency) without violating the SLAs. However, this optimisation concerns the resource utilisation of the whole federation and does not necessarily lead to cost optimisations for all federation's clients.

Rafhyc

The *Resilient Architecture of Federated Hybrid Clouds* (Rafhyc) is a layered Inter-Cloud architecture enabling the development of scientific PaaS and SaaS solutions [117]. It has a *Multi-Cloud layer* that provides a unified access to multiple cloud sites. Individual clouds must implement industry standards in order to work with the *Multi-Cloud layer*. Next is the *Federated Hybrid Cloud Services Layer*, which provides service monitoring, an information system with all available resources, and unified access point to the aggregated resources in the form of web services. The *Resilient Services Layer* uses the lower layers to provide high level management services over the entire federation. These are the functionalities that eScience PaaS and SaaS developers use to build their environments.

Rafhyc is essentially a centralised architecture. The aforementioned layers and the components they are composed of govern how the underlying resources are managed. Its target applications are scientific resource intensive batch processing jobs. Other application domains have not been explored at this point. The *Resilient Services Layer* has an *Efficiency and Price Optimiser* component, which uses cost minimisation resource selection heuristics. In the initial Rafhyc specification, this component only accounts for cost and various SLA characteristics and is not geo-location, policy, and local resources aware. The *Efficiency and Price Optimiser* component can be considered to implement SLA based brokering, as it incorporates certain SLA metrics and QoS estimations, although its documentation does not specify if these can be provided as input parameters at this point.

2.6.2 Peer-To-Peer Federated Inter-Clouds

RESERVOIR

The *Resources and Services Virtualization without Barriers* (RESERVOIR) project is a European project that extends previous research on interconnecting Grids. Its architecture does not feature a central entity and is peer-to-peer — clouds communicate directly with each other to negotiate resource leasing. The usage of multiple clouds is facilitated through *Claudia* — an abstract layer for service execution on top of a cloud federation [148].

Custom application brokering in RESERVOIR can be achieved through elasticity rules of the type *Trigger-Action* [146]. Developers can also decalratively specify how their applications scale. This is done by specifying the SLA of the application in terms of performance objectives (e.g. response time) and a control strategy characteristic (e.g. number of virtual machines). RESERVOIR then creates an approximate elasticity behavioural model that optimises the control strategy characteristic without violating the SLA constraints [146].

RESERVOIR requires that all deployed applications should be agnostic of the hosting data centre location, thus allowing for transparent deployment anywhere within the federation. Clients and their application brokers have no control over the selected hosting cloud [146]. RESERVOIR can be considered pricing aware, since the price of the used resources could be optimised if set as a control strategy characteristic.

Open Cirrus

Open Cirrus is a test bed for distributed systems research and federates several research data centres across the globe. Open Cirrus is not an Inter-Cloud environment per se because it allows users to work directly with the hosts besides utilising virtualised resources [28,47]. Publications about the project do not give much details about the middleware involved in it. It is known that the test bed runs the experimental cluster management software Tashi [98], the Hadoop framework [20], and a custom physical resources management system called Zoni [28].

The Open Cirrus environment has been designed to be data location aware. Tashi and the location metadata provided by the Hadoop Distributed File System (HDFS) are used to schedule computing tasks on the nodes where the correspondent persistent data resides [28]. However, in the publications on the topic nothing has been said about location awareness when using datasets other than HDFS files (e.g. relational database). Also, it has not been stated if users can control the location of the used resources and if they have access to pricing information if pricing is present in this research test bed at all.

OPTIMIS

The OPTIMIS toolkit requires that OPTIMIS software agents are deployed within cloud providers and application brokers. These agents communicate with each other to implement Inter-Cloud provisioning.

One of the agent components is the *Deployment Engine* (DE). It is responsible for the discovery and negotiation with suitable clouds for hosting a particular service. Firstly, a set of suitable clouds meeting some predefined conditions are identified and the *service manifest* (an SLA template) is sent to each of them. Cloud providers answer either with a rejection to accept the service or a deployment offer. DE selects the best offer based on the quantitative (e.g. cost) and qualitative (e.g. some measurement of trust) aspects of the offer. DE is also responsible for initiating the deployment of the service within the selected cloud by providing the appropriate VM images that constitute the needed service. After the deployment of the service, the *Service Optimizer* module is responsible for continuously monitoring the service parameters for SLA violations [73].

In the cloud that is contacted by the DE, the *Admission Controller* (AC) is responsible for either making an offer or rejecting the request. The decision is based on the current workload of the cloud, the requested resources, and an evaluation of the potential profit. Allocation of resources for the provided VMs is done by the *Cloud Optimizer* (CO) [73].

Application brokering can be implemented by configuring the DE components of the application broker or the hosting cloud provider. During the selection process, DE considers the prices of the deployment offers together with other metrics including trust, risk, and energy impact numerical assessments. The exact algorithm for the selection and how configurable it is have not been disclosed. Also, it has not been stated whether DE considers the geographical location of the clouds and the persistent data the provisioned service uses. It has only been mentioned that juridical and political restrictions can be specified and then honoured by the respective DE [73].

The OPTIMIS toolkit supports peer-to-peer federation, when cloud providers communicate directly with each other transparently to their clients. It also supports Independent Inter-Clouds (Multi-Clouds), which will be discussed in a successive section.

Arjuna Agility

Arjuna Agility [24] is a commercial framework for establishing Inter-Cloud federations. It is mostly tailored for federations of in-house data centres, but can also facilitate cloud bursting (usage of public clouds) if the demand increases beyond in-house resource capabilities. Each federation site needs to install an Agility software agent that governs the interaction with the other sites [23]. Each site has its own policy regarding resource sharing and can define what resources in terms of hardware and software are shared. Being targeted mostly at in-house cloud federations, Arjuna Agility addresses provider specific problems like reducing the power consumption [109, 110]. To this point, it does not feature cost optimisation. Priority usage of local resources is addressed as a data centre borrows resources only if it can not meet its own demands. Provisioning decisions are governed by provisioning policies set by an administrator. They are federation specific, not application specific [23].

Global Inter-Cloud

Bernstein et al. envision a worldwide federation of cloud providers, rather than separate small scale federations [35]. They draw analogies from previous experience in integrating separate systems into large scale utility services — e.g. electricity and phone systems and the Internet. They propose a new Inter-Cloud architecture following some of the main design principles of the public Internet.

In a global Inter-Cloud environment, it is important for cloud providers to discover each other and to match their needs for resources. Bernstein et al. propose the usage of a global resource catalogue called *Cloud Computing Resource Catalogue*. It contains information about the shared resources within the federation and should be hosted by several community governed *Intercloud Root* servers. They replicate the catalogue among each other to provide better performance and availability.

Negotiation between providers is facilitated by *Intercloud Exchanges* — distributed servers that allow cloud providers to discover suitable resources. *Intercloud Exchanges* perform match making between cloud providers based on specified preferences and constraints. *Intercloud Exchanges* use the resource catalogue stored on the *Intercloud Roots* to

provide a Distributed Hash Table (DHT) suitable for fast querying. After the negotiation is done, cloud providers continue to communicate directly with each other.

The proposed architecture is not centralised because the *Intercloud Roots* and *Exchanges* are distributed and replicated. Thus, a failure of any of them can not cause a failure of the federation as a whole.

To facilitate all phases of the described global Inter-Cloud system, each participant should have an entity called *Intercloud Gateway*. These entities allow cloud providers, *Intercloud Roots*, and *Exchanges* to communicate with each other using a set of Inter-Cloud protocols and standards. To date, such interoperability protocols and standards have not been widely utilised, although there are already ongoing works in the area [33,34,36,37, 166].

From the clients' viewpoint, application brokering is achieved through specifying appropriate constraints and preferences in the SLA with the cloud provider. The cloud provider should then respect the SLA when submitting requests to an *Intercloud Exchange*. Both private and public clouds can participate within the envisioned global federation. Thus, private clouds can consider using local resources before utilising external clouds. Bernstein et al. envision that cloud providers should be able to specify resource location preferences and constraints on behalf of their clients when negotiating to meet legislative and political requirements [35]. Cloud providers could also implement cost optimisation policies on behalf their clients, although this has not been discussed in the related literature.

2.6.3 Multi-Cloud Services

OPTIMIS

Besides federation, OPTIMIS also supports Independent Inter-Clouds (Multi-Clouds) [73]. In terms of the OPTIMIS architecture, a Multi-Cloud is achieved when clients use their *Deployment Engine* (DE) and *Service Optimizer* (SO) directly to launch and monitor services within multiple cloud providers.

A major drawback here is the need to work with cloud providers that have OPTIMIS agents deployed in their data centres. As discussed, often cloud providers would

not wish to voluntarily participate in an Inter-Cloud and install such agents. The OPTIMIS vision is to allow for externally developed adapters for such clouds which should facilitate the communication between OPTIMIS agents and OPTIMIS agnostic cloud providers. Such adapters should be developed for every cloud service provider.

Contrail

Like OPTIMIS, the previously discussed Contrail project [49] also supports both federation and independent Inter-Clouds. Similar to OPTIMIS, a major drawback of Contrail in terms of supporting independent Inter-Clouds is the need to develop and maintain multiple vendor specific Contrail adapters.

mOSAIC

The *mOSAIC* open source API and platform allow for the development and deployment of applications that use multiple clouds [137]. Unlike most other Inter-Cloud technologies, mOSAIC makes some assumptions about the architecture of the deployed application. It is assumed that the application is divided into components with explicit dependencies in terms of both communication and data between them. Also, the application architecture must be service oriented (SOA) and must use only the mOSAIC API for inter-component communication. Other types of communication (e.g. direct sockets) are disallowed. For each component application, developers must specify the resource requirements in terms of computing, storage, and communication. Some resource requirements are specified for the whole application as well — e.g. overall budget [137].

The mOSAIC platform automatically provisions such applications across a set of predefined clouds, without direct involvement from the user. Thus, the only way to control the application brokering is through the predefined SLA on performance indicators at component and application level.

Application brokering in mOSAIC is done by a component called *Resource Broker* (RB). It is responsible for the mediation between clients and cloud providers. RB is further composed of a *Cloud Agency* (CA) which is responsible for discovery and negotiation with cloud providers and a *Client Interface* (CI) responsible for requesting additional resources

from the *Application Executor* component. The *Application Executor* (AE) component manages the deployment, execution and monitoring of the application within the reserved resources [137]. In this architecture, pricing awareness is promoted through the *Cloud Agency* component, which takes into consideration and negotiates on the pricing policies of the cloud providers.

STRATOS

STRATOS is a cloud broker service [129]. The entry point of the system is the *CloudManager*. Clients describe the application topology and requirements in a *Topology Descriptor File* (TDF) and submit it to the *CloudManager*. It contacts the *Broker* component, which determines the optimal initial resource allocation across clouds. The *Broker* uses another architectural component called *Translation Layer* to access miscellaneous clouds through a uniform API. The *CloudManager* and the *Broker* continuously receive monitoring information based on which they take further provisioning decisions.

Application specific brokering is achieved by specifying requirements and policies within the TDF. STRATOS is obliged to honour the terms from the TDF. Thus, the TDF can be considered as an SLA document between the client and the STRATOS service. The TDF format is XML based and allows for the specification of deployment topology configuration (in terms of VM images, middleware, storage, etc.), constraints, and objectives. The goal of the *Broker* is to optimise the objectives without violating the constraints or breaking the application topology. To build optimisation models, the *Broker* needs adequate measurements of the performance characteristics of cloud providers. For this purpose, STRATOS utilises the Service Measurement Index (SMI) [58] and the metrics defined by Garg [79] and Zachos et al. [168].

The cost is a usual candidate for an objective and thus STRATOS can be pricing aware [129]. We could speculate that Geo-location and Legislation/Policy awareness can be promoted via TDF constraints, although this has not been discussed in the related literature. Also, the mapping of incoming requests to nodes near the appropriate data persistence storage has not been discussed.

SALOON

The SALOON platform helps in the process of cloud resource selection and configuration [140]. SALOON uses extended *Feature Models* (FMs) to represent meta information about the resources within each cloud site. The FMs are afterwards mapped to an ontology called $Onto_{Cloud}$, which models cloud resource concepts and relationships. A second ontology, which is called $Onto_{Dim}$, models the required application characteristics and is user defined. Based on $Onto_{Cloud}$ and $Onto_{Dim}$, SALOON identifies potential deployment configurations [139]. This process is not completely automated and application developers need to manually select among the offered deployment options [140].

The SALOON documentation does not specify if regulatory of data location constraints can be specified in the FMs, $Onto_{Cloud}$, and $Onto_{Dim}$. However, the overall cost of deployment is considered [140]. $Onto_{Dim}$ is generic and can describe applications with different types. It can be considered as a detailed SLA specification that is honoured by SALOON, and hence it can be classified as a SLA-based approach.

MODAClouds

The *MOdel-Driven Approach for design and execution of applications on multiple Clouds* (MODAClouds) brings Model Driven Development (MDD) and cloud computing together [22, 134]. In MDD, software engineers initially define abstract application models and then gradually transform them (manually or automatically) into more concrete models or the final application source code. One of the most popular MDD approaches is the Model Driven Architecture (MDA), which dictates that an application should be developed as a series of 4 types of models: (i) Computation Independent Models (CIM), (ii) Platform Independent Models (PIM), (iii) Platform Specific Models (PSM), and (iv) Implementation Specific Models (ISM) [124]. Hence, in MDA software engineers gradually add computational, platform, and implementation details. This provides the flexibility to change the computational model, platform, or implementation details without propagating changes across the entire stack of models.

Similarly, in MODAClouds a system is designed using the following models:

- **Cloud Computational Independent Model (CCIM)** — generic description of the

required functionality without specific technical details;

- **Cloud Platform Independent Model (CPIM)** — describes the application modules, their interactions, and deployment environments without specifying the concrete cloud platforms;

- **Cloud Platform Specific Model (CPSM)** — refined version of CPIM, which models how the application components use specific cloud services.

The benefit is that individual cloud services can be replaced without having to re-engineer the entire application. Unlike applications built using cloud agnostic APIs, significant application changes may still be needed. However, cloud agnostic APIs allow developers to use only the common features of the underlying cloud services, which can be rather limiting. MODAClouds allows the advanced features of each individual cloud offering to be used through its final CPSM models.

MDD approaches are typically enabled by sophisticated design tools that facilitate the model development, management, and transformations. In MODAClouds, the tooling is in the form of an Integrated Development Environment (IDE) that allows modelling in a domain specific language called *ModaCloudML*. The IDE facilitates semi-automatic model transformation and direct deployment on the used clouds. It also provides cost analysis and information about the geographical location of the cloud sites to be considered. Whenever the models are changed, the IDE allows a new application version to be redeployed.

The IDE interacts with the MODAClouds runtime environment, which is a Multi-Cloud service for resource monitoring, utilisation reporting, and adaptation. It provides feedback to the application developer through the IDE. In addition to the directly managed resource provisioning through the IDE, the runtime also allows trigger-action rules to be executed whenever certain performance thresholds are reached.

Commercial Cloud Management Systems

Several companies provide paid independent Multi-Cloud services. In essence, their features are similar and they compete against each other on the basis of the performance

overhead they add and the variety of cloud providers they can integrate with. It is beyond the scope of this chapter to survey all players in this dynamic market niche and here we only outline the features provided by some of the major companies.

RightScale offers a service for deploying and managing applications across clouds. Users can manage virtual machines on multiple clouds through the RightScale console [144]. Application brokering is achieved through the *alert-action* mechanism, similar to the *Trigger-Action* mechanism. All RightScale virtual machines have predefined "hooks" that continuously send information back to the RightScale console. This facilitates the check of the alert conditions and the execution of the predefined actions. Actions can define scale up/down policies but can also be administrative in nature (e.g. sending email to the system administrator). As an action, a user is allowed to specify in which cloud and location resources should be provisioned. Thus, scaling within RightScale can be location aware. Users can add private clouds within the RightScale console to facilitate local resources utilisation.

Generally speaking, the services of Enstratius (formerly enStratus) [69], Scalr [151], and Kaavo [92] are comparable to those of RightScale. Naturally, there are differences in the pricing, the set of supported cloud vendors and technologies, and some terminology. Alike RightScale, they allow users to deploy virtual machines across different public and private clouds. Application brokering is achieved through automated triggers, similar to RightScale's alerts, which can trigger scale up/down actions.

2.6.4 Inter-Cloud Libraries

Several Inter-Cloud libraries have been developed in recent years. Examples are the Java library JClouds [21], the Python library Apache LibCloud [18], and the Ruby libraries Fog [74] and Apache DeltaCloud [17]. Another library project at an early stage of development is Apache Nuvem [19].

All these libraries are designed to abstract the programmers from the differences in the management APIs of clouds and provide control over the provisioning of resources across geographical locations. Such libraries can be used to ease the development of cloud adapters for technologies like OPTIMIS and mOSAIC. By using such libraries, application developers can program their own application specific brokers. Such brokers

directly manage the underlying cloud infrastructures and thus can meet all previously outlined requirements. A major issues with this approach is that developers need to define appropriate deployment and replication of the broker for availability reasons.

2.7 Discussion

There are fundamental technical, conceptual, and philosophical differences between Inter-Cloud Federations and Multi-Clouds. Some argue that the Multi-Cloud approach can not deliver the required level of transparent interoperability that will enable truly scalable cross cloud applications. They envision that this could be achieved only through Inter-Cloud Federations based on well established standards and communication protocols [32]. Other researchers argue that Inter-Cloud Federations are often not feasible due to the required complex multilateral agreements between competing parties [135, 136]. They see Multi-Clouds as a viable alternative, which moves the management of heterogeneous cloud resources to an external entity — a service or the end client.

In this chapter, we have discussed and classified 23 major Inter-Cloud developments, identified after detailed analysis of the related work. These include both academic and industry projects, which are summarised in Table 2.2. Figure 2.7 outlines how many of them fall into each of the taxonomic groups discussed earlier. The sum of the counts is greater than 23, since some projects support multiple Inter-Cloud topologies.

Most *Multi-Cloud* projects are industry driven. With a few exceptions, they are not research projects supported by academic grants or developed in an academic environment. On the contrary, almost all Inter-Cloud Federation projects are visionary research projects. The only exceptions is Arjuna Agility, used to build in-house Inter-Cloud federations.

This comes to show that currently there is a market demand for Multi-Clouds and the technologies to facilitate them are available. As discussed, the predominant use of *Multi-Clouds* is to provide cloud agnostic access to multiple competing *Independently owned* clouds without their collaboration and knowledge. Such mediating services and libraries are especially useful for small and medium sized businesses that want to deploy applications in multiple public cloud environments for better availability, flexibility, and

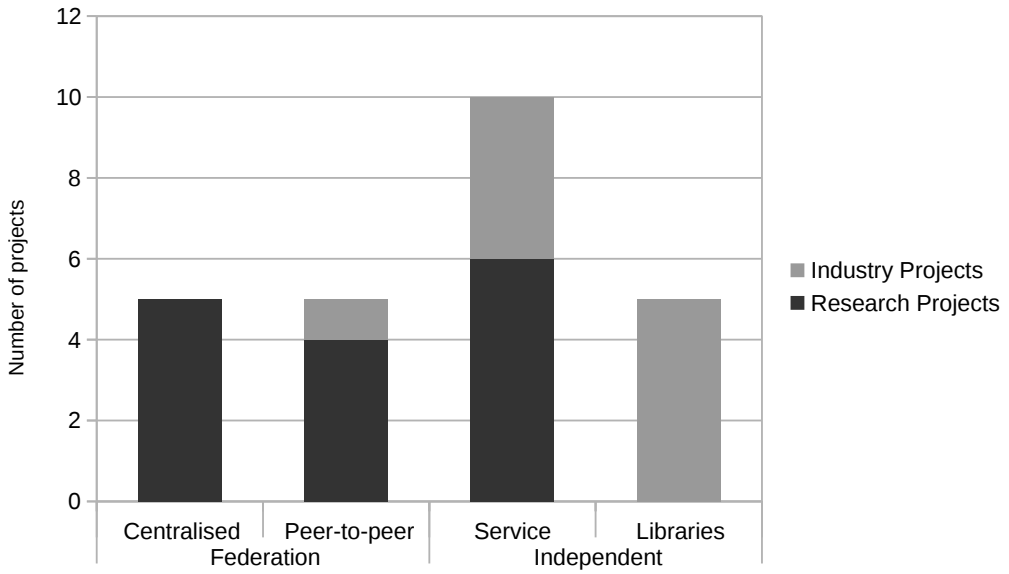


Figure 2.7: Inter-Cloud projects by architecture.

responsiveness. Thus, there is already competition between the providers in this market niche.

On the contrary, almost all federation initiatives are research projects at an early stage of development and adoption. This is because of the many open issues that need to be addressed with respect to provisioning resources and scheduling application components on behalf of the clients.

The majority of the discussed developments take an *SLA based* approach to Inter-Cloud application brokering. These projects try to make the Inter-Cloud transparent to its clients, but as discussed, the SLA based approach meets series of shortcomings. Thus, all of them are research projects at an early stage of development — see Figure 2.8. The direct approach is the most flexible but is also the hardest to work with from the clients' perspective, since clients would need to program their own application brokers. The *Trigger-Action* approach is used by both federations and Independent Inter-Cloud services. It combines the declarative nature of the SLA approach and the provisioning flexibility of the direct approach. All discussed industry projects take either direct or trigger-action approach to application brokering.

Pricing awareness is the only brokering characteristic that can be facilitated by almost all Inter-Cloud projects — see Table 2.2. Most federations do not support brokering con-

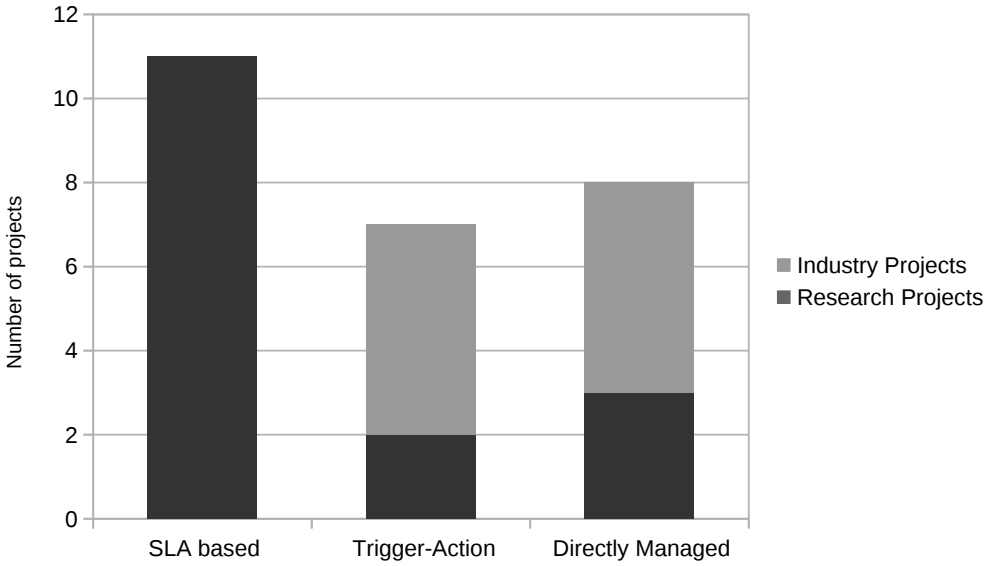


Figure 2.8: Inter-Cloud projects by brokering mechanisms.

sidering any other of the predefined requirements. Few of them have limited awareness of data location, legislation/policy, and local resources. On the contrary, all directly managed Inter-Clouds cover the whole range of discussed requirements and thus provide means for adequate brokering of miscellaneous applications.

As a consequence, most current federation projects are not tailored for *Interactive* applications, since they do not allow application brokering to consider such aspects. On the contrary, *Independent* Inter-Cloud projects can facilitate all of these and thus are more general purpose in nature.

Table 2.2: Summary of Inter-Cloud projects.

Project	**Type, Organisation**	**Architecture**	**Brokering Approach**	**Application Type**	**Awareness**
InterCloud	Research project, University of Melbourne	Centralised federation	SLA based and Directly managed	Singular Jobs	Geo-location, Pricing
Contrail	Private and public European research organisations Funded by EU	Centralised federation and Independent Service	SLA based	Singular Jobs	Pricing

Project	Type, Organisation	Architecture	Brokering Approach	Application Type	Awareness
Dynamic Cloud Collaboration (DCC)	Academic research project supported by South Korean research funds.	Centralised federation	SLA based	Singular Jobs	Pricing
Federated Cloud Management (FCM)	Academic research project supported by EU funds.	Centralised federation	SLA based	Singular Jobs	Pricing
Rafhyc	Academic research project supported by EU and Spanish government funds	Centralised federation	SLA based	Singular Jobs	Pricing
RESERVOIR	Private and public European research organisations Funded by EU	Peer-to-peer federation	SLA based and Trigger-Action	Singular Jobs	Pricing
Open Cirrus	Research testbed by academic and industry partners. Partially funded by US NSF.	Peer-to-peer federation	Directly managed	Singular Jobs	Data location
OPTIMIS	Private and public European research organisations Funded by EU	Peer-to-peer federation and Independent service	SLA based	Singular Jobs, Periodical Jobs, Compute and Data Intensive Interactive application	Pricing
Arjuna Agility	Commercially owned	Peer-to-peer federation	Trigger-Action	Singular Jobs, Periodical Jobs, Compute and Data Intensive Interactive application	Local resources
Global Inter-Cloud	Publications are by people from miscellaneous companies - CISCO, Huawei Technologies, EMC Corporation	Peer-to-peer federation	SLA based	Singular Jobs, Periodical Jobs, Compute and Data Intensive Interactive application	Data location, Local resources
mOSAIC	Private and public European research organisations Funded by EU	Independent service	SLA based	Singular Jobs, Periodical Jobs, Compute and Data Intensive Interactive application	Pricing

Project	Type, Organisation	Architecture	Brokering Approach	Application Type	Awareness
STRATOS	York University. Supported by Canada's NSERC funds, Amazon and CA Inc.	Independent service	SLA based	Singular Jobs, Periodical Jobs, Compute and Data Intensive Interactive application	Geo-location, Pricing, Legislation/ Policy and Local resources
SALOON	Academic research project supported by EU funds.	Independent service	SLA based	Singular Jobs, Periodical Jobs, Compute and Data Intensive Interactive application	Pricing
MODAClouds	Academic research project supported by EU funds.	Independent service	Triger-Action & Directly managed	Singular Jobs, Periodical Jobs, Compute and Data Intensive Interactive application	Geo-Location and Pricing
Commercial Cloud Management Systems (RightScale, Enstratius, Scalr, Kaavo)	Commercially owned	Independent service	Trigger-Action	Singular Jobs	Geo-location, Data location, Pricing, Legislation/ Policy and Local resources
Libraries (JClouds, LibCloud, DeltaCloud, SimpleCloud, Apache Nuvem)	Open source projects	Multi-Cloud libraries	Directly managed	Singular Jobs, Periodical Jobs, Compute and Data Intensive Interactive application	Geo-location, Data location, Pricing, Legislation/ Policy and Local resources

2.8 Summary

In this chapter, we have classified and analysed the state-of-the-art in Inter-Cloud developments. More specifically, we focused on how current Inter-Cloud projects facilitate the brokering of different applications across multiple clouds.

We introduced classifications of Inter-Cloud architectures and brokering mechanisms. Then, we classified Inter-Cloud applications and discussed their coarse-grained properties and brokering requirements. After an extensive analysis of the related literature, we identified, discussed, and fit into the aforementioned classifications 23 major projects. After that, we analysed the state of the art and the trends in the area.

We have identified that there is a major difference in the directions that early research projects and industry projects are heading. While most research projects focus on de-

veloping Inter-Cloud federations, most industry projects provide services or libraries for direct provisioning and scheduling across clouds. Also, most research projects focus on SLA based brokering, while industry driven ones focus on trigger-action based or directly managed brokering.

This evidences that the current Multi-Cloud technologies and environments are mature and can be used to address the outlined challenges. Therefore, in this thesis we take a Multi-Cloud approach to using multiple clouds and introduce architecture, methods, and algorithms for directly managed brokering of interactive 3-Tier applications. To facilitate our work, in the next chapter we introduce a performance model and a simulator for 3-Tier applications in clouds. The newly introduced model is instrumental in the development of some of the algorithms we present in the later chapters. The simulation environment allows for their evaluation.

Chapter 3

Performance Modelling and Simulation of 3-Tier Applications

The research in scheduling and provisioning 3-Tier applications in clouds and across clouds is still in its infancy. To foster the research efforts in the area, we propose an analytical performance model of 3-Tier applications in cloud and Multi-Cloud environments. It takes into account the performance of the persistent storage and the heterogeneity of cloud data centres in terms of Virtual Machine (VM) performance. Furthermore, it allows for modelling of heterogeneous workloads directed to different data centres. Based on our model, we have extended the CloudSim simulator, through which we validate the plausibility of our approach. The conducted experiments with the RUBiS workload show that the predicted performance characteristics by the simulation approximate well those of the modelled system.

3.1 Introduction

THERE are significant differences between the characteristics of batch processing and interactive 3-Tier applications. In essence, while batch processing programs execute without continuous user interaction, interactive applications have to respond to user interactions constantly and timely. This imposes substantially different requirements for application scheduling and provisioning. Foster et al. highlight that clouds, unlike Grids, are especially suitable for interactive real-time applications [75]. Hence, the development of provisioning and scheduling techniques for interactive applications is an important research area.

This chapter is based on the following publication: ***Nikolay Grozev*** *and Rajkumar Buyya, "Performance Modelling and Simulation of Three-Tier Applications in Cloud and Multi-Cloud Environments", The Computer Journal, Oxford University Press, vol. 58, no. 1, pp. 1–22, 2015.*

However, most research works in the area focus on resource intensive Grid-like batch processing applications, leaving interactive ones beyond their scope. For example, Sotiriadis et al. present a literature review of meta-scheduling approaches in cloud environments and discuss their applicability in the area of cross-cloud scheduling [156]. They overview 18 meta-scheduling approaches, all of which schedule batch processing applications and none of them schedules interactive ones. In addition, in Chapter 2 we have outlined that most existing approaches for application brokering across clouds specialise in batch processing applications only. This is not reflective of the more general nature of Clouds, which in contrast to Grids tend to host a wider range of business applications, many of which are interactive and follow the 3-Tier reference architecture.

There are many impediments to the research in interactive application provisioning and scheduling in Clouds. Firstly, installing and configuring different middleware and software components (e.g. application and database servers and load balancers) can be complex and laborious. Another problem is the selection of appropriate workloads and populating the database with suitable data in order to test different scenarios in practice. Last but not least, the incurred financial costs for repeatedly performing multiple tests on large scale applications can be significant.

Historically, similar problems have been encountered by researchers in the area of batch processing applications in Grids and Clouds. These have been resolved by introducing formal performance models and simulation environments that allow for the evaluation of solutions without deploying and executing large scale systems. For example, previous works on distributed systems simulation have significantly fostered the research efforts and have been used in both academia and industry for quick evaluation of new approaches. Unfortunately, existing simulators like GridSim [42], CloudSim [46], and GreenCloud [97] can only be used to simulate batch processing and infrastructure utilisation workloads and are not suitable for interactive 3-Tier applications.

With this work, we aim to fill this gap and make **contributions** that bring the benefits of modelling and simulation to the area of 3-Tier application provisioning and scheduling in Cloud and Multi-Cloud. More specifically we (i) propose a novel analytical model for 3-Tier applications, (ii) define algorithms for implementing a simulator based on the model, (iii) describe our enhancement of CloudSim supporting modelling and simulation

of such applications, and (iv) validate our model through comparison with an actual system. In this chapter we walk the entire path from defining an abstract, theoretical performance model to implementing a full scale simulator based on the model, which is used to validate its adequacy. We identify and address the shortcomings of existing approaches and thus in our model we incorporate the following aspects, which are not accounted for by most related works:

- **Performance Variability** — in a cloud environment, VMs' performance can vary depending on their placement among the physical hosts. VMs with equal characteristics may observe significant performance differences [63].
- **Disk I/O performance** — many 3-Tier applications perform a huge amount of disk operations, which often cause a performance bottleneck.
- **Usage Patterns** — applications' workload change over time. We describe it analytically and implement utilities in the simulation environment to generate workload based on an analytical description.

The rest of the chapter is organised as follows: In Section 3.2, we describe related works and compare the present one to them. In Section 3.3, we provide a succinct description of the targeted class of applications, define the scope and assumptions of this work and summarise our model. Section 3.4 formally defines the performance model. Section 3.5 describes the design of a simulator following this model. Section 3.6 describes several use cases of our model. Section 3.7 evidences the plausibility of our simulator through experiments. Section 3.8 summarises the chapter.

3.2 Related Work

Urgaonkar et al. discuss a model for multi-tier web applications which represents how web requests are processed at the different tiers [164]. This seminal work has been the basis for many other efforts in the area. Their model is represented as a linear network of queues — $Q_1, Q_2 ... Q_m$. Each queue corresponds to an architectural tier. A request comes to a tier's queue Q_i and after being served it is transferred to the queue of the next level Q_{i+1} with probability p or to the queue of the previous tier Q_{i-1} with probability $1 - p$.

Table 3.1: Summary of related work.

Work	Analytical approach	Application type	Server concurrency	Level of Granularity	Server Type
Urgaonkar et al. [164]	Closed network of queues	Multi-tier	Single Threaded	Request	Hardware
Zhang et al. [169]	Closed network of queues	Multi-tier	Single Threaded	Transaction	Virtual Machine
Bi et al. [38]	Hybrid queuing model	Multi-tier	Single Threaded	Request	Virtual Machine
Zhang and Fan [170]	Queuing model	Single tier	Single Threaded	Request	Hardware
Kamra et al. [93]	Queuing model, Control Theory	Single tier	Single Threaded	Request	Hardware
Chen et al. [55]	Closed multi-station queuing network	Multi-tier	Multi-Threaded	Request	Virtual Machine
Our Work	Performance model based on step functions	Three-tier	Multi-Threaded	Session	Virtual Machine

To model web sessions, a new queue Q_0 is introduced which is linked to both Q_1 and Q_m thus forming an infinite queuing system. For each active session a dedicated server is assigned in Q_0. Requests exiting from the last tier proceed to Q_0 and from there they re-enter the system. The time spent at the virtual server in Q_0 models the idle user time.

In their model, Urgaonkar et al. do not account for the performance variability in a virtualised cloud environment. They consider the service time of the servers in the different tiers to be a random variable which is independent of the number of concurrently served sessions. In our model these are reflected. They represent sessions as a set of requests while in our model they are represented as atomic entities continuously creating performance load on the tiers' servers.

A similar analytical model based on a network of queues has been proposed by Zhang et al. [169]. Unlike Urgaonkar et al. they use statistical regression to estimate the CPU cost of each server in the system. Also, the unit of work is transactions, not web requests. They do not consider the inherent performance variability in a cloud environment and the load on the servers in terms of used memory and disk I/O.

Several other approaches also use queuing theory for performance modelling. For example, Bi et al. propose a hybrid queuing model for provisioning multi-tier applications in cloud data centres [38]. They model the system at the level of incoming requests not sessions. Zhang and Fan have developed a queuing model of a web system with one load balancer and two web servers [170]. Their model does not represent a multi-tier system. Kamra et al. model the entire application as a single queue [93]. That is, the different layers are not modelled separately.

In standard queuing theory, a server can not serve more than one request at a time. This is an inherent problem when modelling multi-tier applications, since most application servers are multi-threaded and can serve multiple requests simultaneously. This issue has been identified by Chen et al. and thus they model the whole system as a closed multi-station queuing network to capture this aspect [55]. Every server in the application's architecture is represented as N servers in the queuing model, where N is the maximum number of concurrent sessions in a server. Also, in their work they assume a virtualised environment. A major difference between our work and theirs is that alike other queue-based models, they model the system behaviour at a lower level in terms of requests. Another difference is that they do not reflect the performance variability of the virtualised environment.

A major difference between all these works and ours is that our model is much more coarse-grained — we model the workload in terms of sessions and do not represent each and every request. This makes it much easier to define and experiment with different workloads. Also, since we do not represent requests, our model is more general and essentially agnostic of the underlying technologies. While other models are constrained to a specific type of technology, our model can be used for standard web applications, Rich Internet Applications (RIA), thick-client 3-Tier applications, etc. Lastly, unlike others, our work takes into account the performance and workload heterogeneity of Cloud and Multi-Cloud environments. Table 3.1 further summarises the related work.

In terms of simulation environments GridSim [42] and CloudSim [46] are the most relevant related projects. They are suitable for simulating batch processing workloads, and our present work extends CloudSim to support interactive 3-Tier applications as well. Unlike our approach the GreenCloud [97] project focuses on simulating data centre energy consumption, rather than application performance. Similarly, MDCSim [102] focuses on evaluating the overall energy consumption and latency, but does not facilitate application performance modelling. The SPECI [157] simulator is designed to evaluate the scalability and resilience of data centre infrastructure, and also can not be used for application performance evaluation. The iCanCloud simulator [121] has similar capabilities and design philosophy to CloudSim, but is designed to be faster for large workloads and has a graphical user interface (GUI). However, It does not allow modelling 3-Tier

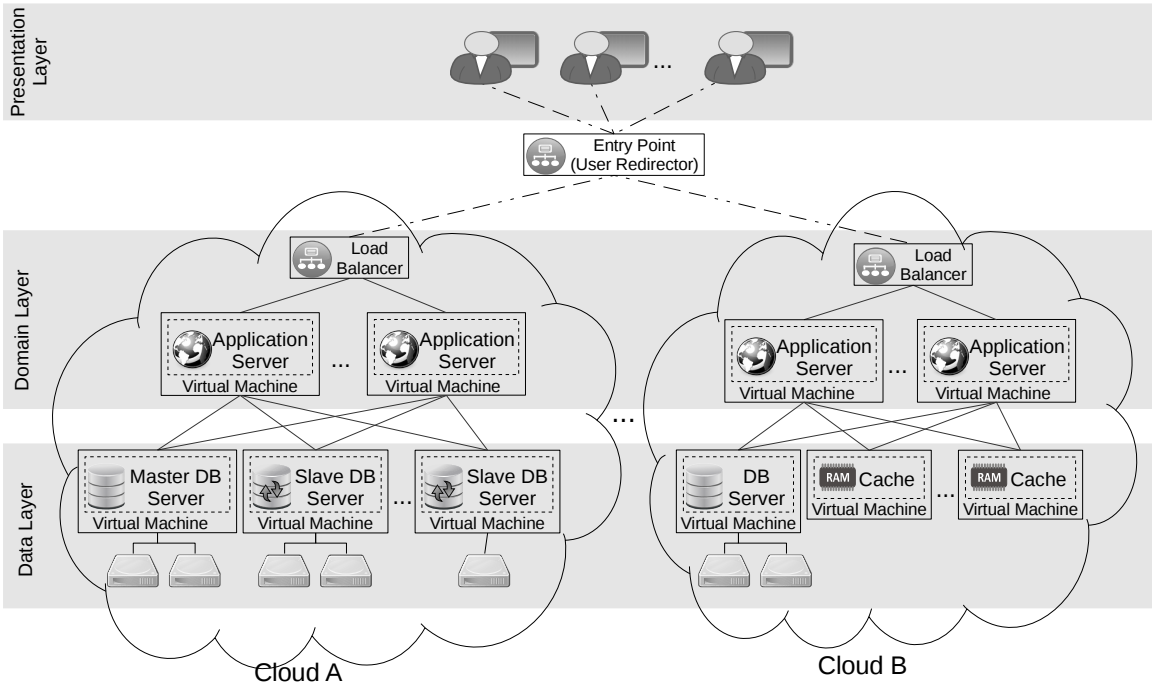


Figure 3.1: 3-Tier reference architecture deployed in multiple clouds.

applications. In fact, we argue that our approach can be implemented as an extension of iCanCloud, as we have done it for CloudSim.

3.3 Overview

3.3.1 Architectural Setting

Dividing a system architecture into layers is a fundamental approach for managing complexity. In essence, a system is constructed as a sequence of layers/tiers, each of which is independent of the successive ones and communicates only with the previous tier(s) through well defined interfaces. This approach brings numerous benefits including increased understandability and maintainability since the layers can be developed, understood, and investigated relatively independently [76].

As to Martin Fowler, the main challenge when designing a multi-tier software system is to decide what are the layers and their encapsulated functionalities [76]. A design error at this stage could result in having hard to maintain coarse grained layers, which contain too much functionality or too many small layers that communicate excessively.

This has precluded the development of layered architectural patterns. Of these, the most prominent is the 3-Tier software architectural pattern, which allows for building scalable and adaptable information systems. As discussed, the 3-Tier architectural pattern dictates that applications are divided into presentation, business/domain, and data layers. Although not web specific, it has gained significant momentum with the advent of the web as it allows quickly migrating traditional applications to the web by replacing only the user interface layer [76].

The presentation layer implements the user interface which the end user interacts with. It can be either a separate application (the so-called "fat client") or web interface accessed through a web browser. Either way, it is executed on the end user's machine and thus we do not take it into account in the servers' performance. Traditionally, the domain layer is executed in one or several Application servers (AS) which communicate with the presentation and the data layers. In the case of Infrastructure as a Service (IaaS) cloud offerings, AS servers are deployed in separate virtual machines (VMs).

The data layer facilitates access to the persistent storage. It is independent from the other two layers because the persistent data often outlives the applications that use it and in some cases it can even serve more than one application. The data layer is executed in one or several database servers (DB).

Traditionally, the data layer has been the most architecturally challenging part of a 3-Tier application, since it is hard to scale horizontally as demand rises. For example, this is the case when the data tier is implemented with traditional relational databases, which are widespread in virtually every application domain. Several approaches like master-slave replication, caching, NoSQL and NewSQL databases [50] have been used to mitigate this issue and have been widely adopted in cloud environments as well.

This architectural pattern is very generic and there are a lot of possible adjustments that can improve the overall system performance. For example, the number of AS servers can be different and can increase or decrease dynamically in response to the demand. As discussed, the data layer can be composed of both transactional relational databases, caches and novel approaches like NoSQL and NewSQL databases. In this chapter, we introduce a general modelling and simulation approach that encompasses all of them, and can be used to evaluate different design alternatives — e.g. use of caching vs master-

slave replication in the data layer.

We consider two main scenarios: (i) a 3-Tier application is deployed within a single data centre and (ii) a 3-Tier application is deployed in multiple Clouds (i.e. a Multi-Cloud) with additional redirection of users to appropriate data centres. Clouds A and B on Figure 3.1 depict two possible deployments. For each application in a data centre, requests arrive through a single load balancer, which can be implemented in different ways. For example, it can be deployed in a VM or provided by the Cloud as a service. Load balancers distribute the incoming workload to a set of application server VMs deployed in the Cloud. Consequently, clients establish sessions with the assigned application servers. The AS servers communicate with the servers from the data layer. Each DB server has one or several hard disks attached, which are used to store persistent data. In the Multi-Cloud scenario, client requests come to a global entry point, from where they are distributed to data centres, each of which has the aforementioned software stack deployed as depicted in Figure 3.1.

3.3.2 Assumptions and Scope

In this chapter, we consider stateful (i.e. maintaining session data) 3-Tier applications. Examples of session data are user preferences, shopping carts, and history of user actions. Also, we assume that the DB servers are deployed in separate VMs. Alternatively, persistent storage could be provided by a cloud provider as a service. There is a plethora of vendor specific Database-as-a-Service (DBaaS) solutions and relying on them can easily lead to vendor lock-in. Hence, it is a best practice to use database servers deployed in separate VMs [3]. Furthermore, many DBaaS solutions like Amazon RDS [7] actually provide VM instances to customers, and thus they can be represented by our model as well.

Very few cloud providers offer their clients non-virtualised physical servers which are optimised for a specific purpose (e.g. running a database). This can be considered an exception form the general practice of building cloud data centres from commodity hardware and providing it to clients in the from of virtualised resources [26, 44]. Hence, most cloud simulators do not represent application execution in non-virtualised environments. An example of such cloud service is the *Oracle Database Cloud*, which offers

database hosting on *Oracle Exadata Database Servers* [126, 127]. Such scenario can still be represented by our approach by modelling a host with the desired characteristics and allocating a single VM to it, which utilises all its resources. Since in the model we do not consider virtualisation overhead, this will be equal to running the workload on the host directly.

We consider that the load balancers implement "Sticky load balancing" policies. That is, after a session between a client and an application server is established, all subsequent requests from this session are redirected to the same AS server. This is a reasonable assumption, since session sharing among servers often leads to overall performance degradation.

Clouds have enabled their clients to execute complex and time consuming analysis over huge amounts of data (i.e. "Big Data") [88]. Often there is a standard 3-Tier web application serving as a front-end for accessing the data intensive analytical back end. Thus, we can largely classify two types of data processing — synchronous (a.k.a online) and asynchronous (a.k.a offline). The synchronous data processing is concerned with performing data retrieval or manipulation in real time — e.g. extracting a list of items from online shop and passing them to the AS server for display to the user. In this case, the user interactions are blocked until the operations are complete. Asynchronous data processing is concerned with running long batch processing jobs in the background. In this case, user interactions are not blocked while the jobs are running. We address synchronous data processing as it is in the standard 3-Tier reference model.

We have outlined modelling data centre heterogeneity as one of our goals. More specifically, in this chapter we focus on the following aspects of heterogeneity:

- **Workload heterogeneity** — for a Multi-Cloud application, sessions arrive with different frequencies based on the cloud geographical location, time zone, etc.
- **Cloud offerings heterogeneity** — different cloud providers offer different VMs in terms of their CPU, memory, and disk capacity.
- **VM consolidation policies** — different cloud providers have different policies for placing VMs on physical hosts, which can affect CPU and disk performance.
- **Performance Heterogeneity** — differences in the hardware capacity of the used

physical machines.

In this chapter, when multiple data centres are used we assume that the workload is distributed among data centres based on its geographical origin and do not model the work of the entry point. That is, clients directly submit their sessions to the nearby data centre. In later chapters, we discuss complementary methods for user redirection across clouds.

3.3.3 Essence of the Model

We propose a session based analytical approach for performance modelling. The workload is represented in terms of user sessions established between users and application servers. Each session incurs performance load on the application and the database servers. The workload of a given server at any point in time is defined as the sum of the workloads of the served sessions. Sessions make use of CPU time and RAM on the application servers and CPU, RAM, and disk I/O on the database servers. We assume that the disk operations on the application server are negligible — e.g. logging, loading configuration files, etc.

A key problem is how to represent the workloads (in terms of CPU, RAM, and I/O) incurred by a session. Logically, they should be represented as continuous functions of time so that workload changes over time can be modelled. Since we aim to implement a discrete event simulator, we need to represent these continuous functions discretely. One approach is to use stepwise functions. That way, by using a finite number of values one can define a continuous function of time.

Figure 3.2 illustrates how a server's RAM usage can be composed of the RAM usages of three sessions and the underlying software stack — i.e. operating system and middleware. The system's RAM footprint can be defined as the used RAM, when no sessions are being served. The RAM usage of the sessions is represented by stepwise functions and the system's RAM usage is considered to be constant. Similar diagrams could be drawn for the CPU utilisations as well. The same approach can be used to represent the I/O utilisation of a given hard disk on a DB server. That is, the utilisation of a hard disk is defined by the I/O operations performed on it by the sessions using this disk from any

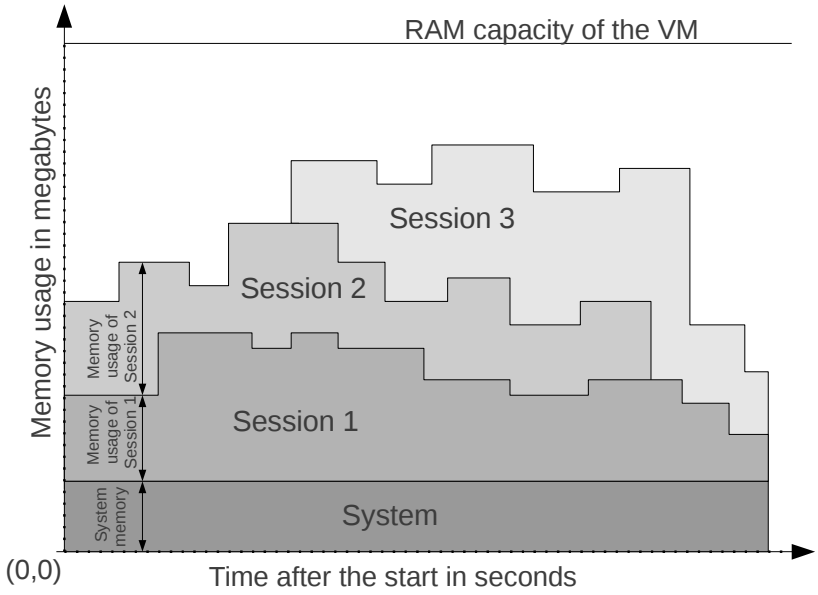


Figure 3.2: RAM load represented as stacked sessions' load.

of the DB servers. A minor difference is that the CPU and I/O capacity are not constant and change over time, because as discussed VMs can observe performance variability.

The frequency of the establishment of sessions is another aspect to model. Typically, such phenomena are modelled with Poisson distribution, representing that sessions arrive with some predefined frequency λ [48, 145]. Furthermore, Paxson and Floyd have found that events like user arrivals/logins are well modelled with a Poisson process for fixed time intervals [130]. However, in the case of a large scale application it is not typical to observe constant arrival rates at all times. Moreover, in a Multi-Cloud environment the workload of the different data centres may be substantially different. For example, a data centre in the US will have higher arrival rates during the working hours in the US time zones and lower otherwise. Thus, we model session establishment rates for a given data centre as a Poisson distribution of a frequency function of time $\lambda(t)$, which is also a stepwise function. If $\lambda(t)$ is a constant function, the arrival rates remain with pure Poisson distribution, and thus our approach is more general than the usual one. Our approach allows to model demand fluctuations and spikes over time and to evaluate how the modelled system behaves in such cases.

If we have several types of sessions, in terms of the incurred performance, we can model the arrival rate of each of these as a separate Poisson distribution over a different

frequency function.

3.4 Analytical model

3.4.1 Session Model

We take a session based approach and instead of jobs or requests, the unit of workload in our model is a session, which represents a series of consecutive user actions incurring system load. Each session is assigned to an application server VM and can use the available DB servers in the data centre for the application.

Unlike jobs, a session generates variable workload over time. For example, for the first few minutes after its establishment a session may utilise heavily the processor of the application server, for the next five minutes it may incur very low resource consumption, and over the next five minutes it may incur significant disk I/O on a database server. In contrast, a job is usually modelled in terms of the required resources (CPU, RAM, etc.) and there are no requirements as to how these resources are provided over time. More formally, a session is defined with the following variables and functions:

Ideal Session Duration

By τ_i we denote the ideal duration of a session s_i measured in seconds. It is the session duration given that all resources for its execution are available. If there are more sessions than a tier can handle, there will be some resource pre-emption and eventually the session will be served in more than τ_i seconds or will fail. By Δ_i we denote the time by which s_i has exceeded τ_i and we consider it as a measure of application responsiveness.

Data Item

To model the locality of disk operations, we need to represent the data resident on the disks. Thus, we introduce the notion of *data item* which represents an entity stored on a disk. Examples of data items are files and portions of database records, which are often accessed and stored together (e.g. a database shard). In the model, we consider that a finite number of data items $d_1 ... d_n$ is specified.

Step Size

We define a common step size δ for all stepwise functions in our model. Generally, we would aim for small value of δ, since that would make the simulation more refined.

CPU Load of the Application Server

By $\nu_{as}(t)$ we denote the CPU load on the application server caused by the session. It is measured in millions instructions per second (MIPS) and represents the average number of CPU instructions required by the session over time. At a given point in time, the required number of operations can be defined as $\frac{n(t)}{2\epsilon}$ where $n(t)$ is the number of instructions required by a session in the time interval $[t-\epsilon, t+\epsilon]$ after its start given a small predefined $\epsilon > 0$. The ϵ value is an input parameter to the model (e.g. $\epsilon = 0.001$). Based on that, we can formally define the values of the stepwise function ν_{as} for the j-the step (i.e. $t \in [(j-1)\delta, j\delta)$) as $\nu_{as}(t) = \frac{1}{\delta} \int_{(j-1)\delta}^{j\delta} \frac{n(x)}{2\epsilon} \, \mathrm{d}x$.

Memory Load of the Application Server

The stepwise function $\phi_{as}(t)$ denotes the RAM usage in the application server by the session over time and is defined analogously to $\nu_{as}(t)$. $\phi_{as}(t)$ defines how many megabytes of memory the session uses t seconds after its start.

CPU Load of the Database Servers

By $\nu_{db}(t, d_k)$ we denote the number of CPU operations needed for the processing of data item d_k by a database server. It is measured in MIPS and is defined analogously to $\nu_{as}(t)$.

Memory Load of the Database Servers

$\phi_{db}(t, d_k)$ defines the RAM usage needed for the processing of data item d_k by a database server and is formally defined analogously to $\phi_{as}(t)$.

Disk I/O Load

By $\sigma_{db}(t, d_k)$ we denote the number of required disk I/O operations on data item d_k over time. It is measured in Millions of Input/Output Operations Per Second (MIOPS). The number of I/O operations with d_k at time t can be defined as $\frac{n(t,d_k)}{2\epsilon}$ where $n(t, d_k)$ is the number of instructions with data item d_k required by a session in the time interval $[t - \epsilon, t + \epsilon]$ and given the predefined $\epsilon > 0$. Analogously to ν_{as}, we can define the "averaged" step values of the σ_{db} functions for the j-the step (i.e. $t \in [(j-1)\delta, j\delta)$) as $\sigma_{db}(t, d_k) = \frac{1}{\delta} \int_{(j-1)\delta}^{j\delta} \frac{n(x,d_k)}{2\epsilon} \, \mathrm{d}x$.

Network delay and "think times"

Within our architectural settings, we can largely classify two types of network: (i) the internal data centre network, used to connect application tiers within a cloud and (ii) the network between the end users and the serving data centre.

Nowadays the internal data centre network usually has sophisticated topology and offers high speed and low latency [89]. Moreover, previous research in multi-tier applications shows that inter-tier network performance typically does not cause significant performance penalty compared to CPU, RAM and I/O utilisation [91, 106]. In fact, Lloyd et al. have empirically shown that for a multi-tier application in a cloud environment, the outgoing and incoming VM network transfer between tiers explain respectively only 0.75% and 0.74% of the overall performance variability. In contrast, they have found that CPU load and disk reads and writes explain respectively over 71%, 37% and 14% of the observed performance [103]. Hence, we will consider the effect of the internal network to be negligible and will not present it in our model.

In contrast, the delays caused by the wide area network, connecting end users with the serving data centre, can be significant. Similarly, users' "think times" are typically present and need to be modelled. From the perspective of the servers, both the external network delays and "think times" result in idle periods, during which the servers are not utilised by the session. More formally, if a session is idle (either because of a network delay or user's inactivity) during the time period $[t_1, t_2]$, then the number of required CPU instructions by the AS server will be $n(t) = 0$ for $t \in [t_1, t_2]$. Since $\nu_{as}(t)$ is dependent

on $n(t)$, its value for this time interval will be affected accordingly. The values of ν_{db} and σ_{db}, representing the CPU and disk utilisation of a DB server, for an idle time period can be defined analogously. The values of ϕ_{as} and ϕ_{db} for an idle period can be defined as the amount of session data kept in the memory by the respective AS and DB servers during these periods.

3.4.2 Modelling Resource Contention

Resource contention appears when some of a server's resources (e.g. CPU) are insufficient to serve all its sessions timely. Although clouds offer seemingly endless resource pools, it is still possible for an application to experience resource shortage. Firstly, some applications allocate resources statically and given a substantial workload their capacity can be exceeded. Secondly, public online applications can experience a sudden and unexpected demand peak. In some cases, resources can not be provisioned quickly enough and thus the already provisioned servers experience contention. Thirdly, in many 3-Tier deployment models, the data tier can not scale horizontally. For example, when a single relational DB server is used without any replication or sharding. In such cases, the DB servers can experience contention. To explore the system performance in such cases, we need to be able to model and simulate contentions. Next, we describe how different resource contentions are handled in our model.

CPU resource contention

When the CPU resources of a server are insufficient, standard process/thread scheduling policies are used to assign processing time to each session. For the time periods that the session is preempted in a tier, it is also considered preempted on the other application tier. In other words, resource preemption in one tier is reflected by an overall slowdown of the entire session, which is representative of synchronous data processing.

I/O resource contention

When the I/O throughput of the DB server is insufficient, standard I/O scheduling policies are used to assign I/O processing to each session. Alike with CPU, an I/O preemp-

tion in one tier leads to a preemtion in the other tier as well.

RAM resource contention

RAM resource contention occurs when at a given point in time the sum of the sessions' and the system's memory usages exceed the amount of memory the server has. When RAM resource contention appears the server stops, resulting in *"out of memory"* error. Consequently, all sessions served on this server fail and it stops accepting new ones. The work of the other servers however continues. For simplicity, in our model we do not consider the usage of swapped disk space. The modelled behaviour is representative of an operating system going out of memory and killing the process of the server.

3.4.3 Session Arrival Model

As discussed, we model the arrival rate of a session of a given type in a data centre as a Poisson distribution of a frequency function — $Po(\lambda(t))$, where $\lambda(t)$ is represented as a stepwise function of time. In a Multi-Cloud scenario, the arrival rates in each Cloud should be defined. Hence, frequency functions need to be specified per data centre. This allows for modelling different workloads coming at different times within each data centre. Thus, we can represent workload influencing factors like time zone differences. Formally, for each session type st_i and data centre dc_j, the number of arrivals can be modelled as $Po(\lambda_{ij}(t))$, where $\lambda_{ij}(t)$ is a user specified function modelling the arrival rate.

3.4.4 Performance Variability across Clouds

The VM performance in a Multi-Cloud environment can vary significantly. The two main factors for this are:

- **VM setup** — a typical VM configuration allows a VM to use additional (but still limited) resources in terms of CPU and I/O if the host machine provides for that. Thus, in the model it is important to specify what is the set-up of the VMs with respect to sharing hardware resources.

- **VM consolidation** — if the VM setup allows for opportunistic usage of host resources, then it is important to note what the VM consolidation policy of each cloud provider is.

In order to adequately model the variability of the VM performance in a Multi-Cloud environment, we need to define both these characteristics for every data centre. In the implementation section we discuss further how these policies can be specified.

3.5 Simulator Implementation

The previously described analytical model allows for coarse grained performance analysis of 3-Tier applications. In this section, we describe our implementation in terms of algorithms and data structures which extend the CloudSim [46] environment to support our model. CloudSim is a mature simulation environment and by extending it our approach can make use of all of its existing features like VM placement policies and energy consumption evaluation.

3.5.1 Representation of Disk I/O Operations

To represent an application's performance in terms of access to persistence storage, we extended CloudSim to support disk operations. This was done at three levels:

- **Host** — we extended the base CloudSim *Host* (physical machine) entity by adding hard disks to it. Each disk is represented as an extension of the *Processing Element* (Pe) component. A Pe in CloudSim models a core of a processor.

- **VM** — similarly, we also extended the VMs to support disk operations. We also extended the schedulers that distribute VM resources among the executing jobs to distribute the available I/O operations.

- **Cloudlet (Job)** — each job (*cloudlet* in terms of CloudSim) has been extended to define a number of required I/O operations. Also, each cloudlet is associated with the data item it uses. Lastly, each cloudlet defines a boolean value, showing if it modifies the data item or not.

Users are allowed to specify the policies for sharing I/O operations among VMs in a Host and cloudlets in a VM the same way they do it for CPU.

3.5.2 Provisioning of I/O Operations

Disk I/O operations need to be distributed at two levels. Firstly, a *Host* should distribute the available disk operations among the *VM*s using the disks. Secondly, *VM*s should distribute their portion of the I/O operations among the jobs/cloudlets deployed in them. There is clear analogy between distributing I/O and CPU operations among *VM*s and cloudlets. Furthermore, as we represent hard disks as extended *Processing Element*s (Pe) we could directly reuse the already existing CloudSim policies for CPU scheduling.

However, unlike CPU, I/O operations are localised. I/O operations on a data item stored on one drive can not be executed on another, while CPU operations can be executed on every available CPU. Since CloudSim does not support such locality in the assignment of CPU operations, directly reusing these policies will lead to erroneous simulation behaviour. To overcome this problem we assign a separate scheduler to each of the hard disks of a *Host*. Thus disk operations can be provisioned per disk. This allows for VMs to have access only to a portion of the available disks and to achieve locality. We have also implemented locality in an extension to the job/cloudlet scheduling policies, so that jobs utilise only the disks, where the corresponding data items reside.

3.5.3 Representing Sessions and Contention

In the analytical model, we represented a session as several stepwise functions defining its resource demands on the servers. Within each step, a session has constant resource requirements in terms of CPU, RAM, and I/O operations. To bring this concept to the simulation environment, we represent the behaviour of each session within a step as one cloudlet executed by the AS server and several cloudlets on the DB servers. Each of the DB cloudlets is associated with the data item it uses. The breakdown of a session's behaviour for a given step into DB cloudlets is based on the data items it accesses during this period. For example if a session accesses three data items during a given step, then there will be three DB cloudlets for this step.

ALGORITHM 1: Session Submission.

input : s_i - session, t_{curr} - current rime, σ - step size

1 submit s_{i1}^{as} to the assigned AS server;
2 submit $\{s_{i11}^{db}, ...s_{i1l_{i1}}^{db}\}$ to the *DBCloudletScheduler*;
// Sets the start times of the cloudlets at position 2 in the queues
3 setStartTime(s_i, 2, $t_{curr} + \sigma$)

In other words for each session and for each step in the simulation we have one cloudlet assigned to the AS server and several cloudlets assigned to the DB servers. Thus, in terms of implementation the step values of ν_{as}, ϕ_{as}, ν_{db}, ϕ_{db}, σ_{db} define the corresponding cloudlets' resource requirements in terms of CPU, RAM and I/O.

The assignment of cloudlets to DB servers is based on the data items they use — cloudlets are assigned to servers which have access to their data. This assignment is done by a new simulation entity *DBCloudletScheduler*. The default implementation of this entity assigns each DB cloudlet to the first server which has access to its data item. By specifying a custom *DBCloudletScheduler* one can simulate different policies in the data layer. For example, cloudlets performing read operations could be redirected to slave database servers, or can be cancelled if the data they retrieve is in a cache.

Formally, for a given session the required amount of CPU operations in the AS cloudlet corresponding to the step interval $[t_i, t_i + \delta]$ is $\int_{t_i}^{t_i+\delta} \nu_{as}(t)\,\mathrm{d}t$, which simply equals $\delta\nu_{as}(t_i)$ because $\nu_{as}(t)$ is a stepwise function, has a constant value within this interval. The other resource requirements (CPU and RAM of a server and I/O on a disk) of the cloudlets are defined analogously.

Each session is represented with two equal-sized queues — one for the AS server and one for the DB servers. The elements of both queues correspond to the steps in the model. The elements of the AS queue are single cloudlets, which are executed sequentially. The elements of the DB queue are sets of cloudlets, whose members execute simultaneously. Cloudlets from subsequent elements of the DB queue run sequentially. The cloudlets from the corresponding elements of both queues are associated with the same *start time*, and no cloudlet is submitted to the servers before its *start time* has passed.

This is depicted in Figure 3.3. For each session s_i on the diagram, the j-th element of the AS queue is s_{ij}^{as} and the j-th element of the DB queue is the set $\{s_{ij1}^{db}...s_{ijl_{ij}}^{db}\}$ where l_{ij}

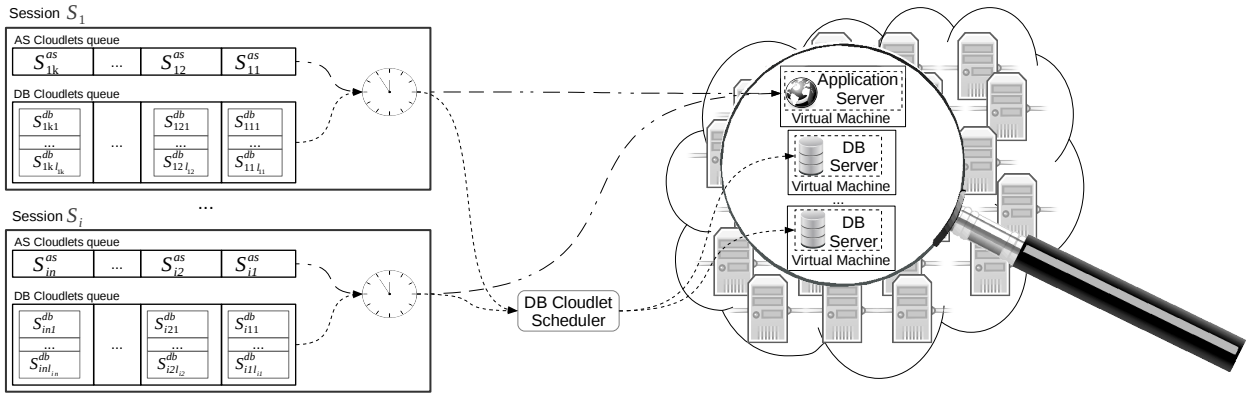


Figure 3.3: Session representation.

is the set size. Cloudlets associated with the j-th step are submitted only if all their *start times* have passed and all cloudlets from the previous steps have finished execution.

After a session s_i is submitted, the cloudlets form the heads of both queues are set the current simulation time t_{cur} as *start times*. The head of the AS queue s^{as}_{i1} is directly submitted to the assigned AS server. Cloudlets $s^{db}_{i11}...s^{db}_{i1l_{i1}}$ from the head of the DB queue are sent to the *DBCloudletScheduler*, which submits them to the DB servers in accordance with the DB layer architecture — e.g. Master-Slave, caching, etc. The cloudlets from the next elements of the queues $s^{as}_{i2}, s^{db}_{i21}...s^{db}_{i2l_{i2}}$ are set starting time $t_{curr} + \sigma$. This procedure is defined in Algorithm 1.

Upon the completion of every submitted cloudlet and the elapse of the *start time* of every cloudlet, which has not been yet submitted, the heads of the two queues are inspected. If all cloudlets from the two heads have finished and the *start times* of the cloudlets from the next queue elements have passed, then the following actions are taken:

1. The heads of the queues are removed;

2. The cloudlets from the new heads are submitted;

3. The cloudlets from the next elements (after the heads) of the queues are set to have starting times σ seconds after the present moment.

Algorithm 2 defines formally the previously described procedure for updating a session. The two key invariants of this algorithm are:

ALGORITHM 2: Update Session Processing — executed when a cloudlet finishes, or the start time of a cloudlet elapses.

input : s_i - session, t_{curr} - current rime, σ - step size

```
1  j ← index of the heads of the queues;
2  if t_curr ≥ getStartTime(s_i, j + 1) then
3      submitNextStep ← TRUE;
4      for c ∈ {s_ij^as, s_ij1^db ... s_ijl_ij^db} do
5          if c is not finished then
6              submitNextStep ← FALSE;
7              break;
8          end
9      end
10     if submitNextStep then
11         pop s_i's queues;
12         j ← j + 1;
13         submit s_ij^as to the assigned AS server;
14         submit {s_ij1^db, ...s_ijl_ij^db} to the DBCloudletScheduler;
15         setStartTime(s_i, j + 1, t_curr + σ);
16     end
17 end
```

- No cloudlet is submitted before its start time and the start time of its counterparts from both queues have come. Formally, no cloudlet $c \in \{s_{ij}^{as}, s_{ij1}^{db} ... s_{ijl_{ij}}^{db}\}$ is submitted before the *start times* for all $s_{ij}^{as}, s_{ij1}^{db} ... s_{ijl_{ij}}^{db}$ have elapsed. This ensures that even if one of the servers is quick to finish its cloudlets, this will not lead to premature exhaustion of the session's cloudlets and the session will still take at least its predefined ideal time.

- No cloudlet is submitted before all its predecessors and all predecessors of its counterparts from both queues have finished. Thus, if there is contention/bottleneck in any of the tiers this will result in an overall delay of the whole session.

As a consequence a bottleneck in a tier results in an overall delay of the served sessions.

CloudSim is a discrete event simulator and one can associate custom logic with every event. This allowed us to check throughout the simulation if the memory usage within a server exceeds its capacity. In accordance with the described model, in such a case the VM simulation entity is stopped and all of its running sessions are marked as failed.

3.5.4 Performance Variability

CloudSim supports a special CPU provisioning policy called *VmSchedulerTimeSharedOverSubscription*. It allows setting the maximum CPU capacity of a VM. If there are sufficient resources, this requested capacity is provided. Otherwise, a smaller share is allocated. This allows us to implement the previously discussed VM performance variability with respect to their mapping to hosts. Since we represent hard disks as extended processors, we reuse this policy to allocate I/O instructions among VMs.

When instantiating a physical machine in the simulation environment, two policies need to be specified — one for sharing CPU time among VMs and one for sharing I/O operations. By choosing the appropriate policy one can either take into account the performance variability or assume that VMs use constant resources.

By using *VmSchedulerTimeSharedOverSubscription* and implementing different VM consolidation logic in each data centre, the performance heterogeneity of a Multi-Cloud environment can be represented.

3.5.5 Load Balancing

To distribute the incoming sessions among AS servers, we introduce a new entity called *LoadBalancer*. It is responsible for assigning the incoming sessions to the AS servers within a data centre. The load balancer is a separate entity and is not deployed within a VM. We implemented a *default load balancer* that distributes the incoming sessions to the AS server with the least percentage of CPU utilisation. However, one can implement different load balancing policies and run simulations with them.

3.5.6 Workload Generation

In our implementation, workload is generated by a new entity called *WorkloadGenerator*. It generates workload for a single data centre and application. Thus, each generator is associated with a single load balancer. At the start of every simulation step, each generator is queried and returns a list of sessions that will be submitted to the load balancer during the step.

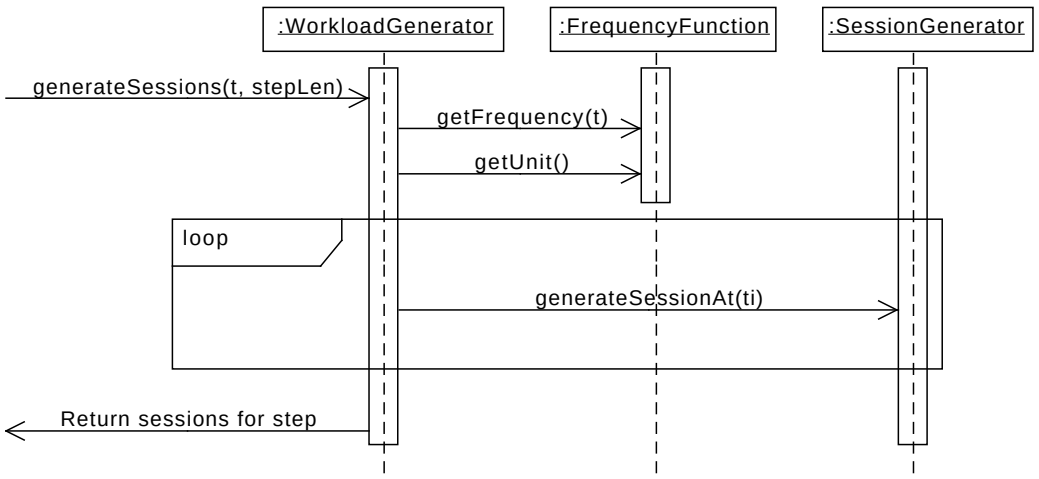


Figure 3.4: Workload generation.

To define the behaviour of a generator, two additional parameters are needed — a *FrequencyFunction* and a *SessionGenerator*. The *FrequencyFunction* represents the $\lambda(t)$ function from the formal model. In our implementation, it has an additional property — the *unit* which represents the time period the frequency is defined for. For example, if the *unit* is 60sec. and the frequency is $\lambda(t) = 3$ for every t, then sessions appear with frequency 3 per minute during the entire simulation. One can implement arbitrary frequency functions that meet the requirements of their simulation.

The default implementation is *PeriodicStochasticFrequencyFunction*, which defines a periodic frequency function. For example, it can be used to define the frequencies on a daily or weekly basis. Using it, one can further define frequency values as samples of a normal distribution. For example, one can specify that on a daily basis in the period 13h-15h the session occurrence frequency per hour has a mean of 20 and a standard deviation of 2. At simulation time, the workload generator generates values that meet these criteria. One can also specify the so-called *null point* of such a function. It represents the moment at which a period starts. *Null points* represent an easy way to define workloads originating in different time zones. That is, if we have a given workload generator, we can "move" it through time zones by just setting the appropriate *null value*.

The *SessionGenerator* is responsible for creating sessions. The default implementation is *ConstSessionGenerator* which generates equal sessions every time it is called. Alike fre-

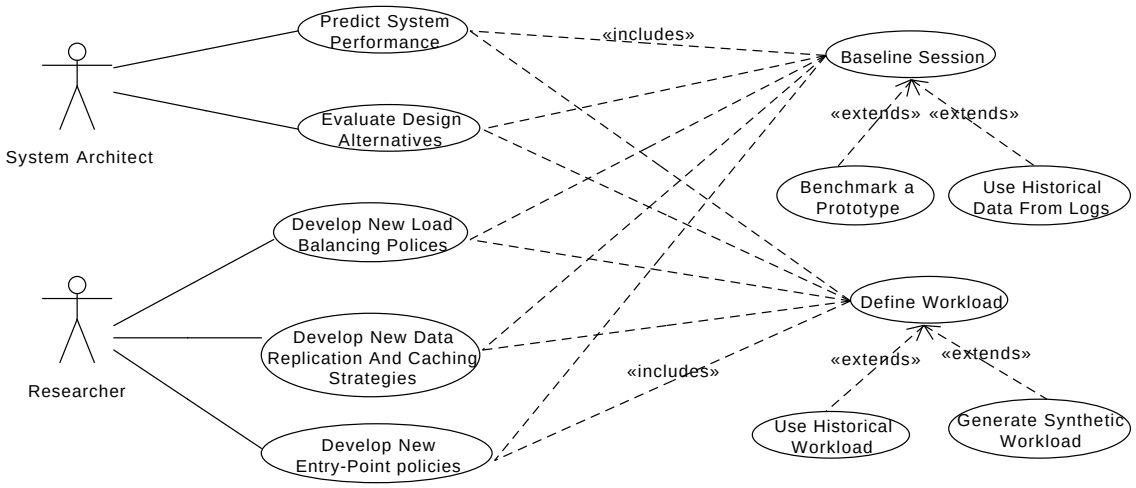


Figure 3.5: Main use case scenarios.

quency functions one can implement and plug their own session generation policies.

The work of a workload generator is depicted in Figure 3.4. At the beginning of every step of the simulation, the *WorkloadGenerator* is requested to generate the sessions for the step. Based on the step length, the frequency and the function's unit the generator computes how many sessions need to be submitted for the step and asks the *SessionGenerator* to create them. The resulting sessions' submission times are uniformly distributed throughout the step's duration.

3.6 Use Cases

This section describes several typical use cases of the proposed modelling and simulation environment. We envision that the proposed environment can be used by both researchers in the field of distributed systems and system architects for performance evaluation and comparison of different provisioning and scheduling approaches. The common use cases are depicted in Figure 3.5.

During the early stages of designing a system, an architect often needs to predict how the designed system would behave under different workloads and if it is going to meet the stated non-functional requirements at all times. Key non-functional requirements usually are the response time to users and the cost of the utilised Cloud resources. Having multiple configurable Commercial Off-The-Shelf (COTS) and Open Source software

components and middleware at their disposal, they need to compare the available approaches, to weight the architectural trade-offs and make an architecture wise decision. This is usually achieved by developing small-scale Proof of Concept (POC) prototypes that demonstrate the overall plausibility of an approach. The cost and efforts for developing multiple POCs to explore various architectural alternatives have been a major inhibitor for system architects. Not to mention that small scale POCs can not be used to test for workload peaks and scaling under extreme workload.

Hence, a simulation environment can be used during these early stages of system development. An architect can perform simulations by "plugging" different policies for load balancing, data replication, and caching by implementing the *LoadBalancer* and *DBCloudletScheduler* APIs. This would allow for evaluation and comparison of different approaches.

Similarly, the proposed modelling and simulation environment could be also of use to researchers developing new provisioning and scheduling approaches for 3-Tier applications in Clouds and Multi-Clouds. New load balancing, data replication, and caching strategies can be developed and evaluated through the *LoadBalancer* and *DBCloudletScheduler* APIs. New entry point policies can also be evaluated.

A key point in all these use cases is the definition of the workload directed to every data centre. Firstly, one needs to define the resource consumption of the sessions that make up the workload. One approach is to define one or several types of sessions, whose performance characteristics over time (expressed in the step-wise functions) are averaged from monitoring data from actual executions. For this, baseline benchmarks of application executions are needed. They can be extracted from historical data from log files of a system prototype. Alternatively, they can be extracted by performing experiments with a prototype of the system or one with similar characteristics.

Once the characteristics of the session type(s) have been defined, the frequencies of the session establishments need to be defined. One approach is to use historical workload — e.g. the times of session establishment can be taken from logs. However, it is often the case that no historical information is available — e.g. if a new system is architected. Moreover, one of the main benefits of performance modelling and simulation is that one can test with a range of workloads including both high and low ones and with static or

dynamic arrival frequencies over time. The aforementioned workload generation functionality of the simulation environment can be used to generate synthetic workload. As discussed, it allows for the generation of arbitrary time dependent workloads.

3.7 Validation

In this section, we model and simulate the behaviour of an actual 3-Tier system and compare the performances of the simulation and the system itself. The goals are two-fold: (i) to demonstrate how one can model a system's behaviour and (ii) to show that our model can predict a system's performance with reasonable accuracy.

A simulator is a generic tool with a wide applicability scope. Testing and validating our simulator against all possible middleware technologies, operating systems, configurations, virtualisation environments, and workloads is an extremely laborious and time consuming task. In this section, we demonstrate the plausibility of our approach with two experiments with the established benchmarking environment Rice University Bidding System (RUBiS) [16, 52, 149, 163]. It consists of an e-commerce website similar to eBay.com and a client application that generates requests to the website.

RUBiS follows the standard 3-Tier architecture and has an application and a database server. We use the PHP version of RUBiS with a MySql relational database. The user interface consists of several web pages with dynamic content [52]. For each incoming HTTP request for a page, the PHP server parses the HTTP parameters, queries or updates the MySql database, and generates the dynamic content of the page, which is then returned to the end user. Hence, each page request induces load to both servers. The MySql server ensures the data is accessed in a transactional and consistent manner, which is important for an e-commerce application. The database consists of 7 main tables, containing information about items, categories, bids, comments, users, regions, and direct purchases [149]. Figure 3.6 shows the relational model in more details. There is an additional table called "old-items" with the same schema as "items", which contains historical data about goods, which are no longer for sale. In the RUBiS workload, the database has been populated with data in a manner similar to *ebay*. More specifically, at every moment there are about 33 000 items classified in 20 categories and 1 million users in 62 regions.

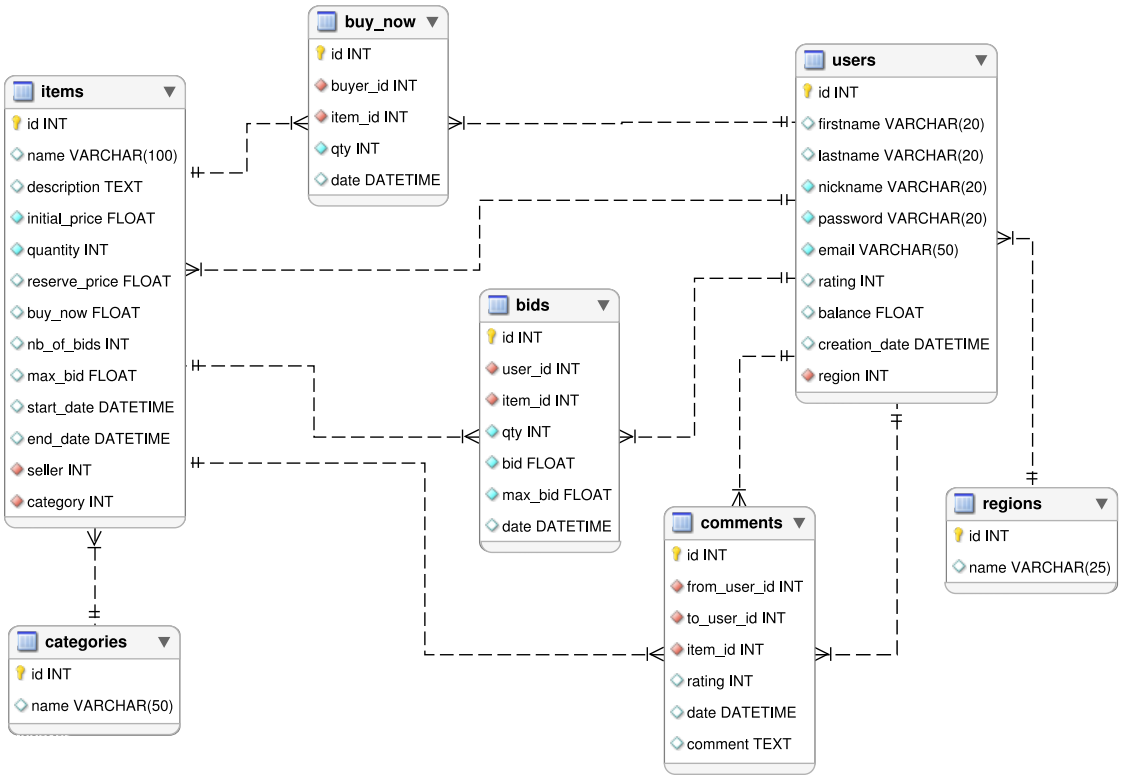


Figure 3.6: Relational model of the RUBiS workload.

In addition, there are half a million comments. The total size of the MySql database and the accompanying indices is approximately 1.4GB [149].

In RUBiS, the workload consists of sessions, each of which comprises a sequence of requests. The succession of requests is based on a transactions table/matrix specifying the probabilities for a user visiting a given page to navigate to every other page or to leave the website. RUBiS also emulates the "think times" between requests. By default, the lengths of the "think times" follow a negative exponential distribution as advised by the TPC-W specification [162]. The mean of this distribution and the number of sessions are parameters of the workload. At run time, for each session the RUBiS client starts by sending an initial request to the "Home" page. Then, based on the mean "think time", the last accessed page, and the transactions table it randomly generates a "think time" period, waits for it to expire, and generates the next request. We have modified the client, so that this process continues until a predefined number of requests have been served or the user leaves the system.

RUBiS comes with two types of workload, defined in the provided transactions tables.

The first one consists of browsing actions only that do not result in a modification of the persistent data. Examples of such actions are browsing items and reviewing sellers' information and ranking. The second workload consists of 85% browsing actions and 15% actions resulting in a modification of the database. It is a much more realistic auction site's workload [16,52]. This is the default RUBiS workload and it is the one we use in all our experiments.

In RUBiS, performance utilisation data is gathered from both servers through the SAR (System Activity Report) system monitor, provided by the SYSSTAT package [158]. At execution time, it collects detailed information regarding CPU, RAM, I/O, and network utilisation. Among these, the measurement of the actual RAM consumption is an elusive goal, because (i) operating systems often keep huge caches of persistent data in memory to avoid some disk operations (ii) in many middleware technologies memory is not automatically released until garbage collection. Consequently, deciding what part of the allocated memory is actually used by the application is not trivial. To bypass this issue, in our experiments we use the SAR metric for amount of *"active memory"*. As to the specification it represents the amount of memory which has been used recently and is not likely to be claimed for other purposes.

In all following experiments we set-up the RUBiS client to execute sessions consisting of maximum 150 requests, and having 7 seconds average "think time". We use the default RUBiS transitions table to emulate users browsing through the website. In all simulations we consider a step time $\delta = 60$ seconds.

3.7.1 Session Performance Baseline

As discussed, one can extract a typical session's performance characteristics either from historical data or by benchmarking. Due to the unavailability of suitable performance logs for RUBiS we use benchmarking, which is described in this section.

Firstly, we assume that all sessions have similar performance characteristics. For a RUBiS workload's sessions this is reasonable, since they all follow the same statistical patterns. This is also typical in real life, where one can often find that "on average" users follow similar usage patterns. In our simulation we will have one session type/prototype defining a session's performance demands over time in terms of the ν_{as}, ϕ_{as}, ν_{db}, ϕ_{db},

σ_{db} functions. The values of these functions are derived from benchmarks of the system's performance. Hence, when we emulate n sessions with the aforementioned statistical properties in RUBiS, in the corresponding simulation we will simulate n sessions from this type. Intuitively, as the session type is representative of a typical session the simulation and the actual execution should have similar performance characteristics given a sufficient number of sessions. We test this assumption in the later subsections.

For some applications, one can observe that different groups of users (e.g. merchants and buyers in an online store) have diverse usage patterns, resulting in different resource utilisation. In such case, different session types should be defined and simulated. However, in the RUBiS environment all users follow the same usage pattern, and thus we consider only one type.

We conduct the benchmarking on a computer with an Intel(R) i7-2600 chipset, which has 4 hyper-threading 3.4GHz cores, 8 GB RAM, and is running 64bit Ubuntu 12.10 operating system. We have deployed the AS and DB servers in separate VirtualBox virtual machines, each of which has 512 MB RAM, a dedicated CPU core from the host and is running 32bit Ubuntu 12.4.

The RUBiS benchmark uses a single DB server and does not use any caching, replication or sharding. Thus, in the formal model we consider a single data item d_1, which is used by all disk operations. Also, in our model the ν_{db}, ϕ_{db}, σ_{db} functions de facto become functions of time only.

We perform two simple RUBiS benchmark tests to define the session type for the simulation. For both tests we get the values of the resource utilisation in terms of CPU, RAM and disk I/O from the SAR monitoring service. We take measurements for all servers and for every second from the tests. First we test the system with only 1 session. The goal is to measure the utilisation caused by the operating system, the middleware and the monitoring service. Secondly, we test with 100 simultaneously started sessions. We attribute the differences in system utilisation between the two tests to the increment of 99 sessions in the workload and based on that we estimate the overhead of a single session.

In the rest of the section, we describe how the AS server CPU utilisation caused by a session can be represented in our model. The procedure for defining the model rep-

resentation based on the benchmarks is not CPU specific and only uses the reported by SAR utilisations in percentages. Thus, without loss of generality, the same approach can be applied to model the incurred RAM utilisations of both servers and the CPU and disk utilisations of the DB server.

Based on the two tests, we can define a session's AS CPU utilisation $t_i \in N$ seconds after its start as:

$$F^{as}_{cpu}(t_i) = \frac{F^{as(100)}_{cpu}(t_i) - \overline{F^{as(1)}_{cpu}}}{99} \tag{3.1}$$

where $F^{as(1)}_{cpu}(t_i)$ and $F^{as(100)}_{cpu}(t_i)$ are the measurements of the CPU utilisation of the AS server for the experiments with 1 and 100 sessions respectively and $\overline{F^{as(1)}_{cpu}}$ denotes the mean of all CPU measurements for the AS server from the test with 1 session. The rationale for Eq. (3.1) is that $\overline{F^{as(1)}_{cpu}}$ denotes the typical overhead utilisation caused by the system components like middleware and OS. Thus, when we subtract it from the overall utilisation and divide by the increment of sessions we get an estimation of the overhead caused by a single session. We call the values $F^{as}_{cpu}(t_i)$ *derived observations* of the session's CPU utilisation on the AS server. Analogously, we can define the session's *derived observations* on both servers in terms of CPU, I/O and RAM.

Next, we need to define the step values of the model step functions based on the derived observations in terms of CPU, RAM and disk I/O. One approach is to use the means of the derived observations falling within a step. However, we observe that the monitoring values returned by SAR for the two experiments include many fluctuations/spikes — see Figure 3.7. Thus, to overcome the effects of outliers, we use the medians of the derived observations falling within a step. Recalling that we consider steps of size 60 seconds, and that we have derived observations for every second, the CPU utilisation of the AS server for the j-th step (i.e. $t \in [t_{60(j-1)}, t_{60j}]$), can be defined as follows:

$$\nu_{as}(t) = median(F^{as}_{cpu}(t_{60(j-1)})...F^{as}_{cpu}(t_{60j-1})) \tag{3.2}$$

As discussed, we define the step values of ϕ_{as}, ν_{db}, ϕ_{db}, σ_{db} analogously, which gives us a complete session performance model.

Lastly, we need to define the ideal session duration τ of the sessions from the session

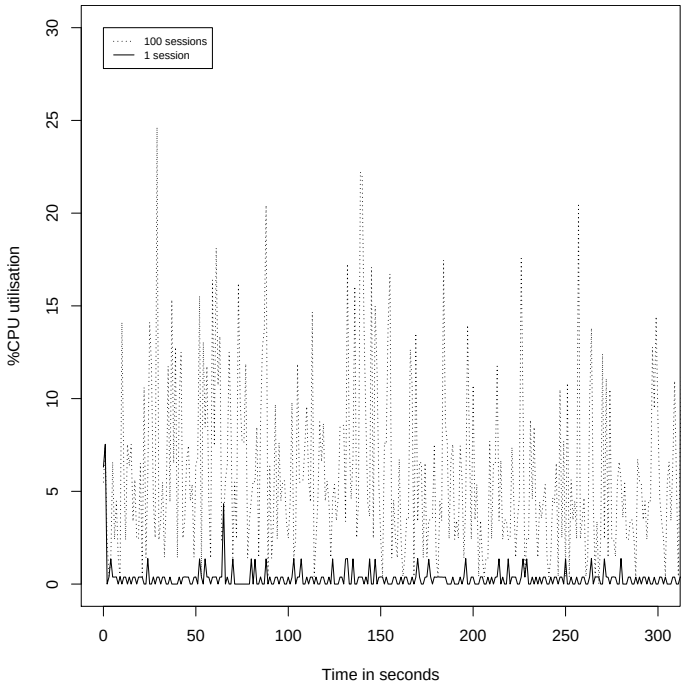


Figure 3.7: CPU utilisation of the DB server with 1 and 100 simultaneous sessions for the initial 5 minutes.

type. In the experiment with 100 sessions, the utilisation of all performance characteristics (CPU, RAM and disk I/O) is much less than 100% at all times. Hence, there has not been any resource contention during this test and thus we can define τ as the mean of the execution times of all sessions from this experiment.

3.7.2 Experiment 1 — Static Workload

The goal of this experiment is to demonstrate that a simulation can be used to predict the resource utilisation of the AS and DB servers and the delays in the sessions' execution. We execute several RUBiS benchmarks, and model and simulate each of them using our approach. We use different workloads in terms of their intensity to test our simulation in the presence of both low and high resource consumption and CPU and disk I/O contention. Then, we compare the predicted by the simulations performance with the ones from the actual execution.

For this experiment, we use the same environment as for the baseline measurement we did previously. We execute workloads consisting of 50, 200, 300, 400, 500, and 600

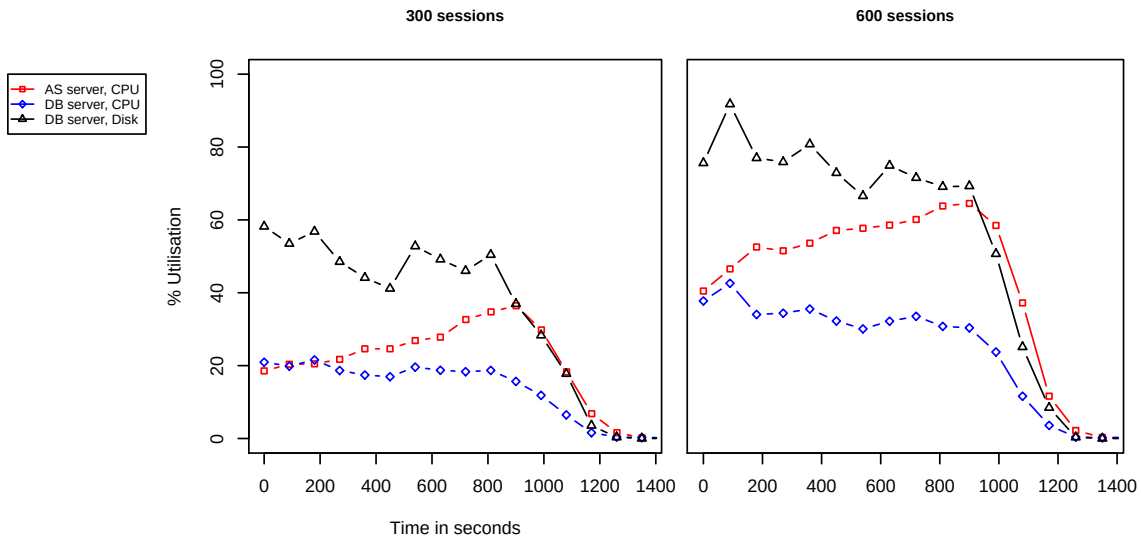


Figure 3.8: CPU and disk utilisations with 300 and 600 simultaneous sessions. The plotted values are averaged for every 90 seconds.

simultaneously starting sessions and we record the delays and the utilisation over time. The sessions are based on the default RUBiS transitions table, which defines regular transactional load in an e-commerce website and which we used for the baselining.

From the monitoring data we can conclude that the used workloads are mostly transactional and data intensive, since the DB disk utilisation dominates the CPU utilisations of the two servers for the majority of the execution time. This is shown in Figure 3.8 which depicts the CPU and disk utilisations of the two servers at execution time given workloads of 300 and 600 sessions. We have plotted the averaged utilisations for every 90 seconds. An interesting observation is that by the end of the execution, the disk utilisation decreases significantly. One reason for this is the suspension of some simultaneously started sessions, whose lengths are generated by RUBiS as values of a random variable. In fact, this is why all resource utilisations decrease in the latest stages of the experiment. Another reason is the in-memory caching of disk operations by the operating systems and the database server, whose effect gradually increases during the experiment. Another interesting observation is that, the CPU utilisation of the AS server increases, as the disk utilisation decreases. This is because the throughput of the data tier is increased, which allows the AS server to serve more user requests timely.

In our parallel simulation environment we define a single host, with two VMs with parameters modelled after the real ones. Then we execute simulations with the same

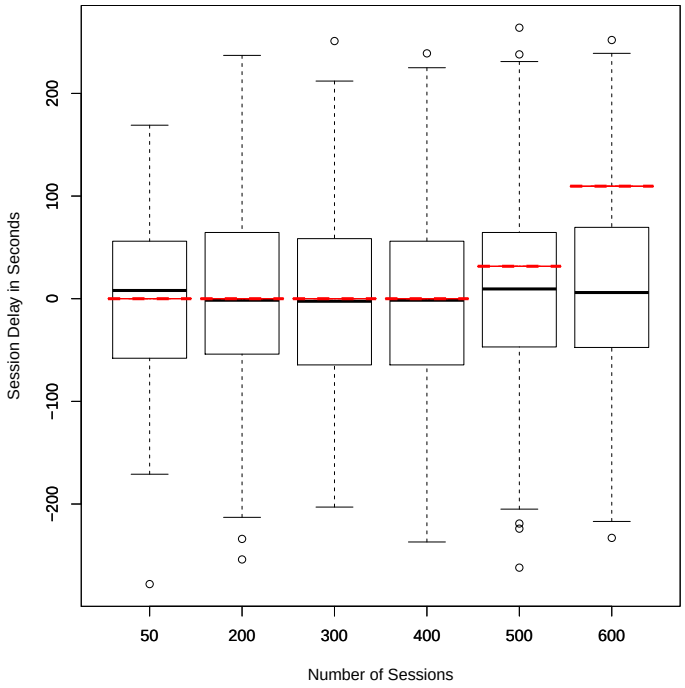


Figure 3.9: Execution delays (boxplots) and predicted delays (crossing horizontal lines) in Experiment 1.

number of sessions as in the aforementioned RUBiS experiments and we record the resource utilisation of the two VMs. The step functions, defining a sessions' behaviour, are the ones extracted from the previous section.

First, we compare the accumulated session delays from the RUBiS execution (Δ_i) with the ones from the simulation. Since in the simulation we have identical sessions, which start simultaneously and whose performance utilisation does not depend on a random variable, we get a single numeric value as a predictor of the delay. However, in the RUBiS benchmark the delays vary, as the "think times" and the succession of requests vary based on random variables. Figure 3.9 depicts the estimated by the simulations execution delays and the actual ones and shows that the predicted delay is a good estimator of the actual delays, always falling in the interquartile range, except for the case with 600 sessions, when the 3rd quartile is exceeded with about 40 seconds. We believe this difference is caused by performance improvements like in-memory caching, which are not represented in our model, and which can mitigate the consequences of contention in case of high workload. Since the most utilised resource is the disk of the DB server (see Figure 3.8) this has impact on the overall delay. It is worthwhile noting that some

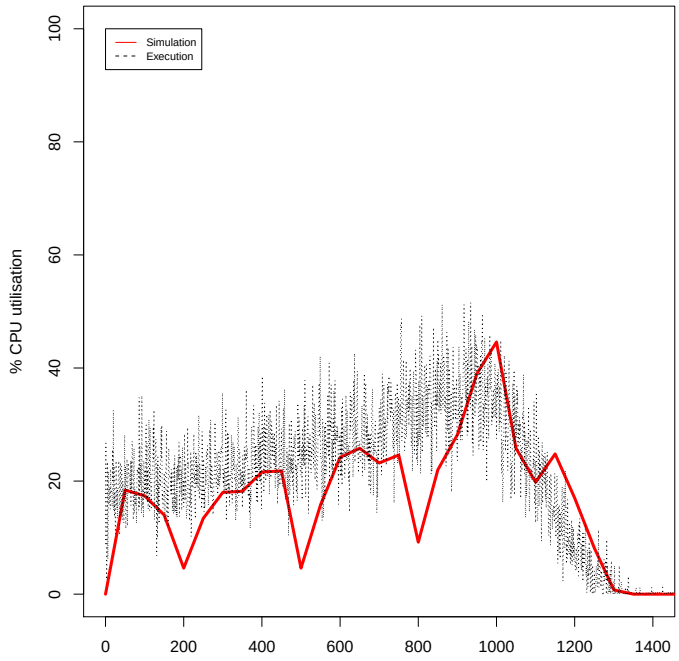


Figure 3.10: Predicted and actual CPU utilisation of the AS server with 300 simultaneous sessions in Experiment 1.

execution delays are actually negative, which means that some sessions finish before the ideal execution time has passed. This is because we defined the ideal execution time τ as the mean of the execution times of a multitude of sessions that do not experience any contention. Thus, by pure chance in the actual execution we have sessions, whose own ideal duration will be shorter.

Next, we compare the utilisations over time in terms of CPU, RAM and I/O of the servers in the execution and simulation experiments. Figure 3.10 shows how the predicted CPU utilisation of the AS server approximates the real one for a workload of 300 sessions. To make the diagram clearer we have plotted the averages of each 50 observations from the simulation, not of each single observation. Figure 3.11 depicts how the predicted by the simulation disk I/O utilisations approximate the one observed at execution time for workloads of 50, 300 and 600 sessions. Again, we have shown the averages of each 50 observations from the simulation. The figure shows that with the increase of the workload, the actual disk I/O utilisation increases as well, and causes contention in the benchmark with 600 sessions. This behaviour is well approximated by the predicted by the simulation disk I/O utilisation.

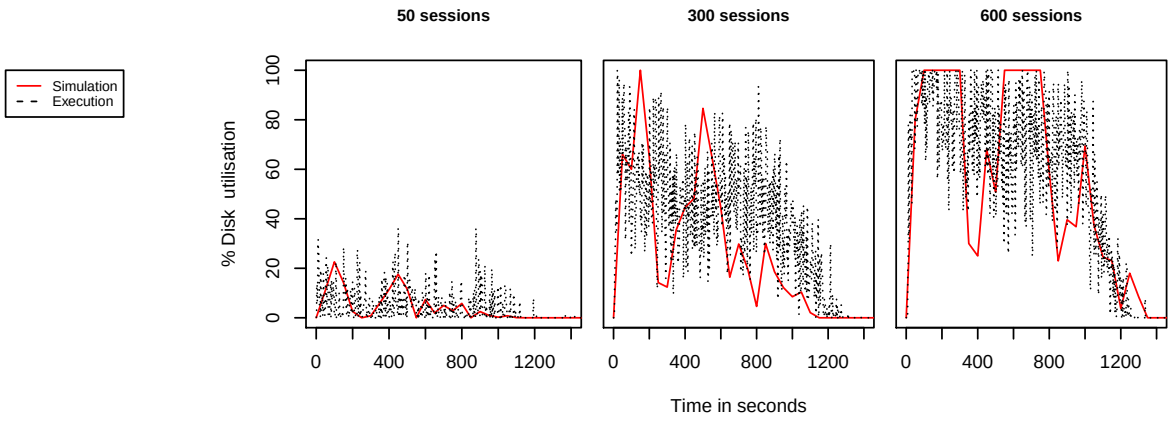


Figure 3.11: Predicted and actual disk I/O utilisation of the DB server with 50, 300 and 600 simultaneous sessions in Experiment 1.

Table 3.2: 95% Confidence Intervals (CIs) and estimates of the median of the utilisation differences between simulation and execution. Measurements are in percentages.

Num.	AS server, CPU		AS server, RAM		DB server, CPU		DB server, RAM		DB server, I/O	
Sessions	95 % CI	Est.	95 % CI	Est.	95 % CI	Est.	95 % CI	Est.	95 % CI	Est.
50	(-5.43, -4.93)	-5.22	(-1.61, -1.58)	-1.59	(-2.90, -2.39)	-2.84	(-1.93, -1.91)	-1.92	(-8.48, -4.64)	-6.89
200	(-13.41, -12.51)	-12.97	(-3.27, -3.22)	-3.25	(-7.67, -6.15)	-7.05	(-1.70, -1.69)	-1.7	(-19.81, -15.05)	-17.45
300	(-15.32, -12.82)	-14.05	(-0.67, -0.56)	-0.61	(-2.31, 14.14)	-0.32	(-0.11, -0.09)	-0.11	(-17.16, -11.01)	-13.96
400	(-18.29, -14.04)	-16.52	(-1.05, -0.91)	-0.99	(22.92, 26.66)	24.97	(2.62, 2.70)	2.68	(-13.99, -6.92)	-10.45
500	(-17.16, -5.29)	-9.01	(-2.47, -2.11)	-2.29	(24.21, 27.43)	25.86	(4.36, 4.38)	4.37	(-16.90, -9.65)	-13.25
600	(-11.09, -7.06)	-8.98	(-0.91, -0.25)	-0.63	(27.46, 31.13)	29.34	(5.29, 5.38)	5.33	(-17.32, -9.84)	-13.59

For every workload, server, and resource (i.e. CPU, RAM and I/O) we have two matched time sequences of observations for every second of the execution — one from the RUBiS benchmark and one from the simulation. Since in the simulation we can do many monitoring operations without hurting the performance, for better precision we perform 10 observations for each second and then we average them. In other words we have paired samples of the utilisation from the executions and the simulations. All utilisation measurements are in terms of percentage — e.g. 90% disk utilisation. Based on the procedure of the Wilcoxon signed rank test with continuity correction we can compute the 95% confidence intervals (CI) and an estimate (the pseudomedian) for the median of the difference between the populations.

Table 3.2 lists the 95% CIs and the estimates for all experiments and utilisation characteristics. For all experiments and characteristics, except for the CPU utilisation of the DB server, the 95% CIs for median of the difference in utilisation contain values within

20% range of the ideal case of 0%. The only exception is the CPU of the DB server, the estimate of the deviation for which reaches nearly 30% for the case of 600 sessions. Once again this difference can be explained by performance optimisations like caching, that we do not account for. Such optimisations can mitigate contention and improve performance given a significant workload, which is reflected by the increase of inaccuracies when the number of sessions is increased.

3.7.3 Experiment 2 — Dynamic Workload

The goal of this experiment is to show that simulation can be used to predict the resource utilisations given a dynamic workload in a heterogeneous Multi-Cloud environment. For this purpose, we set up two installations of RUBiS. The first one is the environment we used in the previous experiment and for the benchmarking. We call this environment Data Centre 1 (DC1). The second one is in the Asia Pacific (Singapore) region of Amazon EC2. It consists of two *m1.small* VM instance — one for the AS and one for the DB server. We call this environment Data Centre 2 (DC2). The two environments have different underlying hardware, virtualisation technologies and provide VMs with different capacities. Hence, with this experiment we demonstrate how we can model and simulate application execution in heterogeneous cloud environments.

We modify the RUBiS client, so that it can dynamically generate sessions over a 24h period, based on predefined frequencies for each hour. For example, we can specify that between the second and third hour of the test the number of arriving sessions is normally distributed with mean μ and standard deviation σ. Then at execution time, at the beginning of the second hour of the test, the RUBiS client generates a random number n, which is an instance of this distribution. Then n sessions are submitted at equal intervals between the second and the third hour.

Table 3.3 lists the frequencies we use in this experiment. They are representative of the daily workload of a small website. We use them to generate workload for DC1. To demonstrate the effects of heterogeneous time zone dependent workloads, we assume that DC2 is in a location with 12h time zone difference and has the same workload pattern. Thus, for DC2 we "shift" the frequencies with 12 hours. Figure 3.12 depicts the mean session arrivals for the two data centres.

Table 3.3: Hourly frequencies of the workload for Experiment 2.

Time period	Mean hourly frequency	Standard deviation
0h-6h	10	1
6h-7h	30	2
7h-10h	50	3
10h-14h	100	4
14h-17h	50	3
17h-18h	30	2
18h-24h	10	1

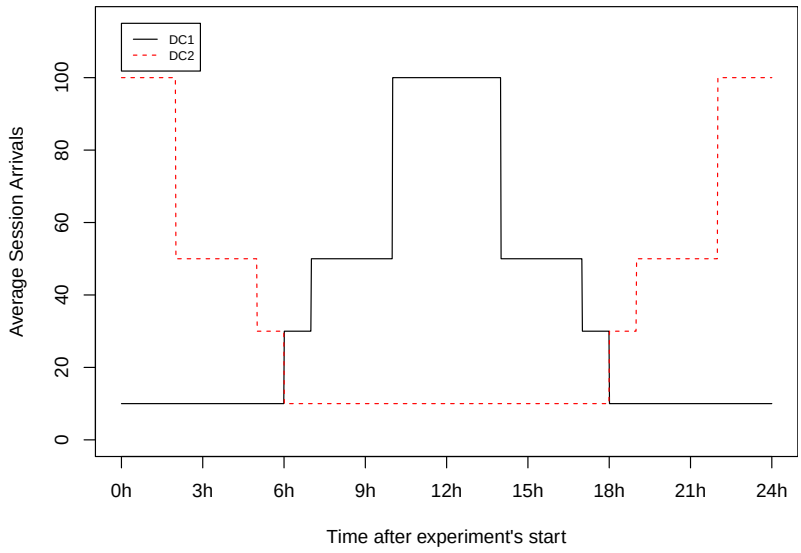


Figure 3.12: Average hourly frequencies of session arrivals for the two data centres in Experiment 2.

In the simulation environment we use the data centre we used in the previous experiment to represent DC1. In our simulation, DC2 has the same set-up as DC1. However, the RAM, CPU, and I/O capacity of the VMs and the physical machines (hosts) are modelled after those or AWS.

However, m1.small instances in Amazon EC2 do not have a dedicated processing core, like the ones in our local environment (DC1). Hence, some of the CPU time may be distributed to other VMs located in the same host. As discussed, the dynamism of VM placement can result in significant performance variability. Amazon defines the metric EC2 Compute Unit, to measure the amount of CPU capacity a m1.small VM has. However, often this is just a lower bound, and one can observe better performance if for example the VM does not share the host with others. Typically, one should evaluate their

scheduling and provisioning policies against the lower bound capacities of the provider's SLA. However, in this comparative study we aim to statistically compare simulation and execution and thus we need as accurate VM performance data as possible.

To achieve this, firstly we examine the capacity of the host CPU, as defined in the */proc/cpuinfo* Linux kernel file of the two VMs. The SAR monitor defines the *%steal* metric, which describes the percentage of "stolen" CPU operations by a hypervisor — e.g. for serving other VMs. This metric is always zero for VMs with dedicated cores. At execution time we account for the *%steal* value when we define the CPU utilisation. In the simulation environment the CPU capacity of the VMs in DC2 is defined as the mean CPU host capacity, which has not been "stolen" during execution. Similarly, Amazon does not define (not even with a lower bound) how many disk I/O operations a VM with an *Instance Store* can perform for a second. They do so only for EBS backed instances. Hence, we benchmark our instances to get an estimate for I/O performance. We use the aforementioned VM characteristics to define the DC2 VMs in the simulation environment.

As a result of the aforementioned performance measurements, we have found that there are significant differences in the VM capacities in DC1 and DC2. While the VMs in the local environment DC1 have dedicated CPU cores, the ones in Amazon EC2 do not. Our measurements show that on average the CPU capacities of the AS and DB VMs in DC2 equal approximately 66% and 76% of the capacities of the respective servers in DC1. Similarly, the disk capacity of the DB server in DC2 is approximately 75% of the disk capacity of the DB server in DC1. Furthermore, the VMs in DC2 are standard Amazon EC2 instances with 1.7GB of RAM, while our local VMs have only 512MB of RAM. Hence, with this experiment we demonstrate application modelling and simulation in heterogeneous environments with respect to both the observed performance and the incoming workload.

To simulate the workload, we use the previously defined workload functionalities of the simulator. During the execution, we sample the performance utilisations every minute. In the simulation, we take utilisation samples for every second and then define the utilisation for every minute as the average of the samples for the included seconds.

Figure 3.13 displays the average VM utilisations of of the servers for every 5 minutes, as predicted by the simulation. It shows that the predicted resource utilisations of the

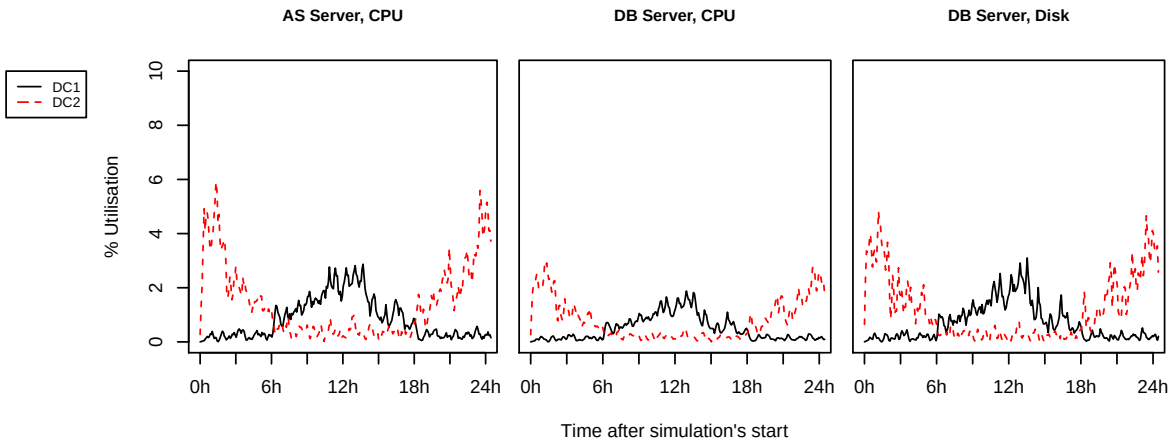


Figure 3.13: CPU and disk utilisations of the AS and DB servers in DC1 and DC2 at simulation time in Experiment 2.

servers follow the patterns of the workloads shown in Figure 3.12. As a result of the heterogeneity of the hardware and VM capacities in the two data centres, the utilisations of the servers in the two data centres differ significantly. For example, the CPU utilisation of the AS server in DC2 at the peak times of the workload exceeds almost twice the utilisation of the DC1 server at its peak workload time. This is because the average CPU capacity of the AS server in DC2 is about 66% of the one in DC1. Similarly, the disk utilisation during the workload peak of the DB server in DC2 is much higher than the one in DC1, as a result of having only 76% of the disk capacity of DC1. Given intensive workloads, this performance heterogeneity can easily result in different contentions and consequently application performance in the different data centres.

Table 3.4 lists the 95% CIs and the estimations of the medians of utilisation differences of the servers in DC1 and DC2. As in the previous experiment we use the procedure of the Wilcoxon signed rank test with continuity correction to compute the 95% confidence intervals (CI) and the estimates. From the table we can see that the estimated error (the pseudomedian of the differences) is less than 11% for all characteristics. The accumulated delay for every simulated session in this experiment is 0. This shows that at execution time there were no resource contentions with the given workload.

Table 3.4: 95% Confidence Intervals (CIs) and estimates of the median of the utilisation differences between simulation and execution in DC1 and DC2. Measurements are in percentages.

Data centre	AS server, CPU		AS server, RAM		DB server, CPU		DB server, RAM		DB server, I/O	
	95 % CI	Est.	95 % CI	Est.	95 % CI	Est.	95 % CI	Est.	95 % CI	Est.
DC1	(-0.43, -0.37)	-0.4	(-4.06, -3.97)	-4.02	(-0.35, -0.31)	-0.33	(-1.06, -0.89)	-0.97	(-1.16, -0.97)	-1.07
DC2	(0.65, 0.74)	0.7	(-10.46, -10.31)	-10.39	(-2.78, -2.71)	-2.75	(-7.12, -7.07)	-7.1	(0.49, 0.58)	0.53

3.8 Summary

We have introduced a new approach for performance modelling of 3-Tier applications and their workload in Cloud and Multi-Cloud environments. Unlike others, our model is coarse grained, session based, and does not represent each and every user interaction. Thus, it is suitable for quick prototyping and testing of scheduling, provisioning, and load balancing approaches. We have implemented a CloudSim extension realising this model, which can be used for performance evaluation without the need to deploy large scale prototypes.

By definition a system model is a simplified description of the original system, and as such our model makes some simplifying assumptions. To validate it, we have conducted two experiments demonstrating the plausibility of our approach by comparing the simulation results with the results from an actual system execution. By using both public Cloud and private in-house infrastructures with different virtualisation environments, settings, and workload dynamism we demonstrated the plausibility of our approach in a wide range of environments.

In the next chapter, we introduce an architecture and algorithms for load distribution and resource management for 3-Tier applications in multiple clouds. We use the simulation environment, presented in this chapter, to evaluate the performance of our approaches under varying workload.

Chapter 4

Multi-Cloud Provisioning and Load Distribution for 3-Tier Applications

In this chapter, we introduce a general approach for deploying 3-Tier applications across multiple clouds to satisfy their key non-functional requirements. We propose adaptive, dynamic, and reactive resource provisioning and load distribution algorithms that heuristically optimise the overall cost and the response delays, without violating essential legislative and regulatory requirements. Our approach makes use of features common to all Infrastructure as a Service (IaaS) providers and does not rely on any interoperability and portability capabilities of the used clouds. Thus, it bypasses the open issues in these areas, which makes it viable right away without relying on further interoperability and portability developments. Our simulation with realistic workload, network, and cloud characteristics shows that our method improves the state of the art in terms of availability, regulatory compliance, and Quality of Experience (QoE) with acceptable sacrifice in cost and latency.

4.1 Introduction

TRANSITIONING existing applications to clouds or Multi-Clouds is not straightforward. A cloud is not merely a deployment environment, where existing software solutions can be transferred. It introduces novel characteristics not existing in traditional in-house deployment environments like seemingly endless resource pool and the risk of an unpredictable outage in external infrastructure [165]. Hence, software applications need to be more scalable and fault tolerant so they can dynamically adapt to workload fluctuations by adequately allocating and releasing computing resources and autonomously and timely address infrastructure failures. Software engineers need to de-

This chapter is based on the following publication: ***Nikolay Grozev*** *and Rajkumar Buyya, "Multi-Cloud Provisioning and Load Distribution for Three-tier Applications", ACM Transactions on Autonomous and Adaptive Systems, vol. 9, no. 3, pp. 13:1–13:21, 2014.*

sign for the Cloud, not only to deploy in the Cloud. This is even more important when using multiple data centres situated in different legislative domains, constructed with different hardware, network and software components, and prone to different environmental risks.

The key **contributions** of this chapter are (i) a design approach for interactive 3-Tier Multi-Cloud applications and (ii) adaptive dynamic provisioning and autonomous workload redirection algorithms ensuring imperative constraints are met with minimal sacrifice in cost and QoE. Our approach does not modify the 3-Tier pattern itself, but introduces additional components managing the cross-cloud resource provisioning and workload distribution. This is essential because it allows the migration of existing applications to a Multi-Cloud environment. Also, new Multi-Cloud 3-Tier applications can be developed using the plethora of existing architectural frameworks thus leveraging proven technologies and existing know-how. The newly introduced components facilitate the implementation of 3-Tier systems which observe: (i) increased availability and resilience to cloud infrastructure failure, (ii) legislation and regulation compliance, (iii) high QoE, and (iv) cost efficiency.

The rest of the chapter is organised as follows: In Section 4.2, we provide an overview of the related works and compare them to ours. Section 4.3 details the targeted class of applications. Section 4.4 outlines our architecture. Section 4.5 motivates and details our algorithms for load balancing, autoscaling and cloud selection. Our experimental settings and results are discussed in Section 4.6. The final Section 4.7 summarises our findings.

4.2 Related Work

There have been significant efforts in the development of Multi-Cloud open source libraries for different languages like JClouds [21], Apache LibCloud [18], Apache DeltaCloud [17], and Apache Nuvem [19]. All of them provide a unified API for the management of cloud resources (e.g. VMs and storage), so that software engineers do not have to program against the specifics of each vendor's API. While not providing application brokering (consisting of provisioning and scheduling) themselves, these libraries

can be instrumental in the development of new cross-cloud brokering components. Similarly, services like RightScale [144], Enstratius (formerly enStratus) [69], Scalr [151], and Kaavo [92] only provide unified user interface, APIs and tools for managing multiple clouds and it is the clients' responsibility to implement appropriate provisioning and scheduling.

Apart from these Multi-Cloud libraries and services, the OPTIMIS [73], Contrail [49], mOSAIC [137], MODAClouds [22], and STRATOS [129] projects also facilitate Multi-Cloud application deployment. In all of these projects the geographical locations of the serving data centres cannot be considered. Thus, often it is not possible to implement legislation aware application brokering. In contrast, in our approach the *Entry Points* and *Data Centre Control* layers enable legislation compliant user routing to eligible clouds through a process called *matchmaking broadcast*. Secondly, all of these projects only manage resource allocation and software component deployment, and none of them facilitates the distribution of the incoming workload to the allocated resources. Their components are only concerned with resource provisioning and set-up and do not deal with the load distribution and autoscaling of the application once it is installed. In contrast, in our work we manage the incoming workload and dynamically provision resources accordingly through the components of the *Entry Points* and *Data Centre Control* layers.

Furthermore, these projects are SLA-based, which means the application brokering is specified in a Service Level Agreement (SLA) using a declarative formalism. The Cloud Standards Customer Council (CSCC) discusses in a technical report that SLAs currently offered by cloud providers are immature [62]. Thus, to achieve flexible application brokering these approaches rely on advances in the currently adopted SLA practices, or the introduction of new brokering components that can interpret novel SLA formalisms. In contrast, our approach directly manages the underlying provisioning and the mapping of workload to resources, without relying on advances in SLA specifications, and thus it is applicable right away.

Cloud services like Route 53 [8] and AWS Elastic Load Balancer (ELB) [11] can distribute incoming users to servers in multiple data centres using standard load balancing techniques. AWS ELB can distribute workload among servers located in a single or multiple AWS availability zones, but cannot direct users to clouds of other providers.

Route 53 is Amazon's Domain Name System (DNS) web service. It supports Latency Based Routing (LBR), which redirects incoming users to the AWS region with the lowest latency. Both Route 53 and ELB do not consider applications' regulatory requirements when selecting a data centre site. Moreover, they do not consider the cost and the degree of utilisation of the employed within a data centre resources. In contrast, our approach for directing users to cloud sites accounts for all these aspects.

4.3 Preliminaries and Scope

In a Multi-Cloud environment, the standard 3-Tier software stack is replicated across all used cloud sites. Clients arrive at one or several *Entry Points*, from where they are redirected to the appropriate data centre to serve them.

The domain layer within a data centre can scale horizontally by adding more AS VMs. For a given application, within a data centre there is a load balancer, which distributes the incoming requests to the AS servers. Every request arrives at the load balancer, which selects the AS to serve it. There are two types of 3-Tier applications in terms of the domain layer design — *stateful* and *stateless*. Stateful applications keep session data (e.g. shopping carts and user meta-data) in the memory of the assigned AS server. Hence, they require *sticky load balancing*, which ensures all requests of a session are routed to the same server. Stateless applications do not keep any state/data in memory and therefore their requests can be routed to different AS servers.

The data layer often becomes the performance bottleneck because of requirements for transactional access and atomicity. This makes it hard to scale horizontally. As to the famous CAP theorem [40], [41] a distributed database architect should balance between persistent storage consistency, availability, and partition tolerance. The field of distributed horizontally scaling databases has been well explored in recent years. For example Cattell [50] surveyed over 20 novel NoSQL and NewSQL distributed database projects. Traditional techniques like replication, caching and sharding also allow for some level of horizontal scalability.

The eligible data caching and replication strategies are very much application specific and it is impossible to incorporate them within a general framework encompassing

all 3-Tier applications. In other words the right balance between the CAP (consistency, availability and partition tolerance) requirements is domain inherent. For example one application may require that data is not replicated across legislative regions, while another may allow it to achieve better availability. Therefore, in this chapter we do not deal with the application specific data deployment. We investigate flexible provisioning and load distribution provided the data is already deployed with respect to the application specific CAP requirements. It is the system architect's responsibility to design the data layer in a scalable way obeying all domain specific legislation rules so that it can be accessed quickly from the domain layer. This is a reasonable constraint, as database design is usually the first step in a 3-Tier system design and it often serves other applications (e.g. reporting and analytics) as well. Our approach ensures that once the data is deployed appropriately, users will be redirected accordingly and enough processing capacities will be present in the AS layer.

4.4 Overall Architecture

4.4.1 Architectural Components

Figure 4.1 depicts the proposed architecture. We augment the traditional 3-Tier architectural pattern with two additional layers of components:

- **Entry Point Layer** — responsible for redirecting incoming users to an appropriate cloud data centre to serve them.
- **Data Centre Control Layer** — responsible for: (i) providing information to the *Entry Point Layer* regarding the suitability of a data centre for a given user, (ii) monitoring and scaling of the provisioned resources within a data centre, and (iii) directing the incoming requests.

The *Entry Point Layer* consists of one or several VMs, which can be deployed in several data centres for better resilience. When users come to the system, they are initially served at an *Entry Point* VM. Based on the users' location, identity, and information about each data centre, the *Entry Point* selects an appropriate cloud and redirects the user to it. After this, the user is served within the selected data centre and has no further interaction with

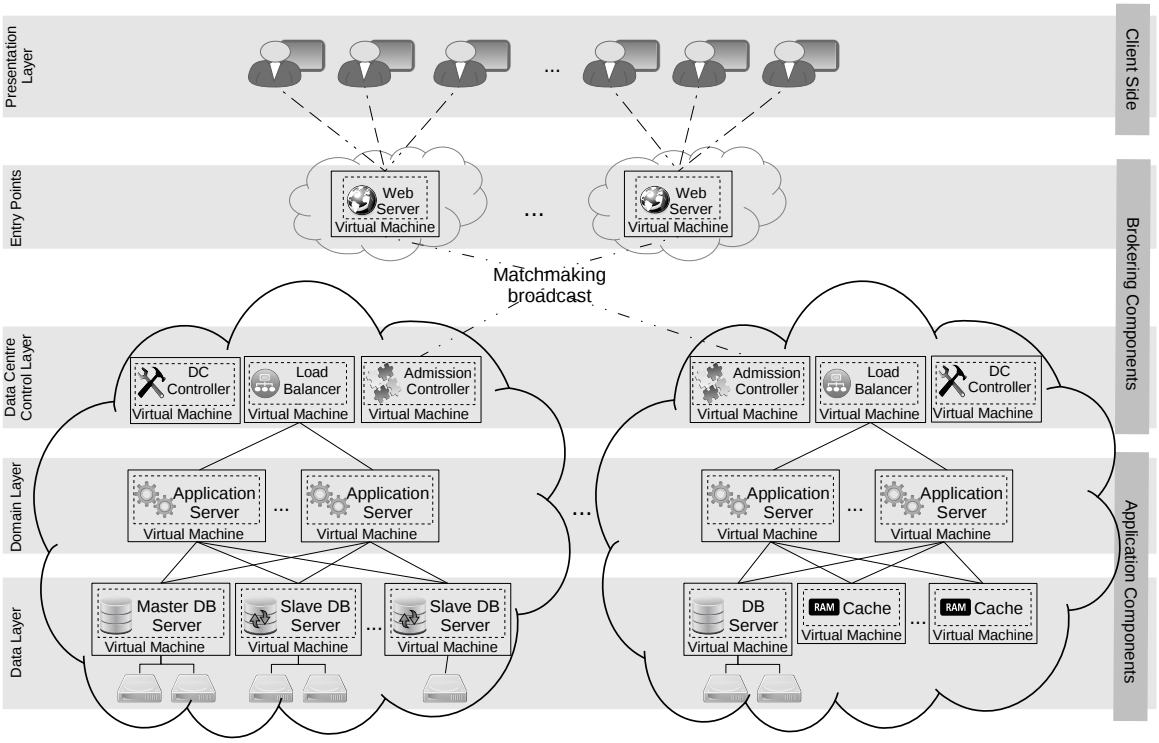


Figure 4.1: Overall layered architecture. The Brokering components mange the system's provisioning and workload distribution, while a standard 3-Tier software stack serves the end users.

the *Entry Point*. We emphasise that the cross-cloud interactions between *Entry Points* and *Admission Controllers* happen only once, immediately after user arrival, and hence do not result in further communication delay as the user is being served.

At first glance, an *Entry Point* can be likened to a standard load balancer, as it redirects users to serving data centres. However, standard load balancers redirect each user request, while *Entry Points* redirect users only upon arrival. Furthermore, *Entry Points* collaborate with the *Admission Controllers* to implement cloud selection respecting legislative, data location, cost, and QoE requirements, which is not implemented in standard load balancers.

In each data centre the *Data Centre Control Layer* consists of three VMs:

- **Admission Controller** — decides whether a user can be served within this data centre and provides an estimation of the potential cost for serving him/her. Upon request, it provides feedback to the *Entry Point* VMs to facilitate their choice of a data centre for a user.

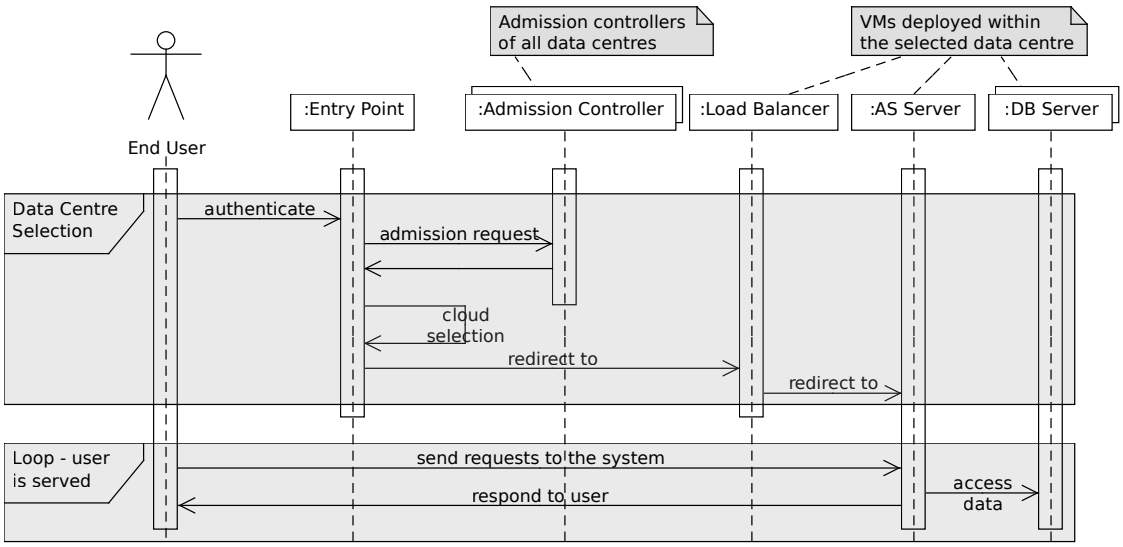


Figure 4.2: Component Interaction. Cloud site selection happens once, upon the user arrival. Subsequent requests are handled within the selected data centre.

- **Load Balancer** — the standard *Load Balancer* component from the 3-Tier reference architecture. We consider it as a logical part of the *Data Centre Control Layer*, since it redirects requests to the application servers.

- **DC Controller** — responsible for observing the performance utilisation of the running AS servers, and reactively shutting down or starting AS VM instances to meet resource demands at minimal cost.

In principle, the *DC Controller* and the *Load Balancer* VMs may be replaced by services like Amazon Auto Scaling [5] and AWS Elastic Load Balancer (ELB) [11]. Nevertheless, not all cloud providers have such services. Even if a provider offers auto scaling services, it is often not possible to monitor custom performance metrics — e.g. number of live application sessions. Moreover, in Section 4.5 we introduce novel algorithms for load balancing and autoscaling which in conjunction reduce cost and the probability of server overload. These are not implemented by current cloud providers and hence, for generality, we will consider the usage of separate VMs for these purposes.

4.4.2 Component Interaction

Figure 4.2 depicts the interaction between components upon user arrival. In the first phase, the brokering components select an appropriate cloud site for the user, based on his/her identity. As a first step, the user authenticates to one of the *Entry Points*. At this point, the *Entry Point* has the user's identity and geographical location (extracted from the IP address). As a second step, the *Entry Point* broadcasts the user's identifier to the *Admission Controllers* of all data centres. We call this step *matchmaking broadcast*.

There are no restrictions on the location of the *Entry Points*. Ideally, they should be positioned in a way which minimises the network latency effects during the *matchmaking broadcast*. One reasonable approach is to deploy each entry point in one of the used clouds, given that the clouds are already selected in a way which serves the expected user base with adequate latency.

Within each data centre, the *Admission Controller* checks if the persistent data for this user is present. Additionally, each *Admission Controller* implements application specific logic to determine which users can be served in the data centre based on the regulatory requirements. The *Admission Controllers* respond to the *Entry Point* whether the user's data is present and if they are allowed (in terms of legislation and regulations) to serve the user. In the response, they also include information about the costs within the data centre.

Based on the *Admission Controllers'* responses, the *Entry Point* selects the data centre to serve the user and redirects him/her to the *Load Balancer* deployed within it. The *Entry Point* filters all clouds which have the user's data and are eligible to serve him/her. If there is more than one such cloud, the *Entry Point* selects the most suitable with respect to network latency and pricing. If no cloud meets the data location and legislative requirements the user is denied service.

After a data centre is selected, the user is served by the AS and DB servers within the chosen cloud as prescribed by the standard 3-Tier architecture. He or she does not have any further interaction with the brokering components. Hence, we consider that AS servers deployed in a data centre can only access DB servers in the same data centre. This is a reasonable constraint, as often there is no SLA about the network speed and latency between data centres, and thus cross-cloud data access can lead to performance issues.

Furthermore, transferring persistent data across the public Internet may be a breach of the legislation and policy requirements of many applications.

4.5 Provisioning and Workload Management

4.5.1 Scalability Within a Cloud

At present, the practices for load balancing among a dynamic number of VMs in a cloud environment and among a fixed number of physical servers are the same — e.g. round robin or some of its adaptations. When using physical servers, one usually tries to distribute the load so that the servers are equally loaded and all sessions are served equally well. In a cloud environment, if the number of AS VMs is insufficient, new ones can be provisioned dynamically. Similarly, if there are more than enough allocated AS VMs, some of them could be stopped to reduce costs. If the load of a stateful application is equally distributed among underutilised VMs, then no VM can be stopped without failing the sessions served there.

This is not an impediment for stateless applications, as sessions are not bound to servers and hence VMs can be stopped without causing service disruption. Thus, standard load balancing techniques like weighted round robin or "least connection" can be effective. However, in the case of stateful sessions, in order to stop an AS VM one needs to make sure it does not serve any sessions. One approach is to transfer all sessions from an AS VM to another one before shut-down. However, this is not straightforward, as active sessions together with their states need to be transferred without service interruption. A better approach is to balance the incoming workload in a way, which consolidates the sessions in as few servers as possible without violating the QoS requirements. This results in a maximum number of stoppable (i.e. not serving any sessions) servers. Intuitively, if the load balancer packs as many sessions as possible (without causing overload) to a few servers, then the number of stoppable servers (not serving any sessions) will be maximal.

This idea is implemented in Algorithm 3. It defines a *sticky load balancing* policy, and thus after the first session's request is assigned to a server, all successive ones are assigned to it as well. It takes as input the newly arrived session s_i, the list of already deployed AS

ALGORITHM 3: Load Balancing Algorithm.

input : s_i, th_{cpu}, th_{ram}, VM_{as}

```
1  sortDescendinglyByCPUUtilisation(VM_as);
2  hostVM ⟵ last element of VM_as;
3  for vm_i ∈ VM_as do
4      vm_cpu ⟵ CPU utilisation of vm_i;
5      vm_ram ⟵ RAM utilisation of vm_i;
6      if vm_cpu < th_cpu and vm_ram < th_ram and !networkBuffersOverloaded() then
7          hostVM ⟵ vm_i;
8          break;
9      end
10 end
11 assignSessionTo(s, hostVM)
```

servers VM_{as} and two ratio numbers in the interval $(0, 1)$ — the CPU and RAM thresholds th_{cpu} and th_{ram}. As a first step in the algorithm, we sort the available AS VMs in a descending order with respect to their CPU utilisation. Then we assign the incoming session to the first VM in the list, whose CPU and RAM utilisations are below th_{cpu} and th_{ram} respectively and whose input and output TCP network buffers/queues are not becoming overloaded. These buffer sizes are denoted by the Recv-Q and Send-Q values returned by the *netstat* command. To simplify the algorithm's definition we have extracted this logic in a new boolean function $networkBuffersOverloaded()$. It simply checks if there is a TCP socket for which any of the ratios of the Recv-Q and Send-Q values to the maximum capacities of those queues is greater than 0.9. If there is no such server, the session is assigned to the least utilised one (line 2). An obvious postcondition of the algorithm is that a newly arrived session is assigned to the most utilised in terms of CPU server, whose CPU and RAM utilisations are under the thresholds. If there is no such server — the one with the least utilised CPU is used. By increasing the thresholds, we can achieve better consolidation of sessions at the expense of a higher risk of CPU or RAM contention, which may result in lower response time. On the contrary, when the thresholds are lower, the overall number of underutilised VMs will be higher. Therefore reasonable values for these thresholds are the autoscaling triggers (used by our autoscaling algorithm), which define if a server is overloaded.

The *DC controller* is responsible for adjusting the number of AS VMs accordingly. This implementation of the *Load Balancer* (Algorithm 3) allows the *DC controller* to stop AS VMs which serve no sessions. The *DC controller* is also responsible for instantiating

new AS VMs when needed. Algorithm 4 details how this can be done when using on-demand VM instances. This algorithm is periodically executed every Δ seconds to ensure the provisioned resources match the demand at all times. The input parameters of the algorithm are:

- $\mathbf{t_{cur}}$ — the current time of the algorithm call;
- $\mathbf{tgr_{cpu}}$ — CPU trigger ratio in the interval $(0, 1)$;
- $\mathbf{tgr_{ram}}$ — RAM trigger ratio in the interval $(0, 1)$;
- $\mathbf{VM_{as}}$ — list of currently deployed AS VMs;
- $\mathbf{n}$ — number of over-provisioned AS VMs to cope with sudden peaks in demand;
- $\mathbf{\Delta}$ — time period between algorithm repetitions.

If an AS VM's CPU or RAM utilisation exceeds respectively tgr_{cpu} and tgr_{ram} or some of its input/output TCP network buffers is becoming overloaded, we call this server overloaded. In the beginning of Algorithm 4 (lines 1-11), we inspect the statuses of all available AS VMs and note if they are overloaded or free (i.e. not serving any sessions).

In an online application, the resource demand can raise unexpectedly in the time periods between two subsequent executions of the scaling algorithm. Moreover, booting and setting up new AS VMs is not instantaneous and can take up to a few minutes depending on the underlying infrastructure. Hence, resources cannot be provisioned instantly in response to the increased workload. If the workload spike is significant, this can result in server overload and performance degradation. One solution is to over-provision AS VMs, so that unexpected workload spikes can be handled. The n input parameter of the algorithm denotes exactly that — how many AS VMs should be over-provisioned to cope with unexpected demand.

As a postcondition of the algorithm execution, there should be at least $n + 1$ free AS VMs if all other AS VMs are overloaded, or n otherwise. For example, in the special case when $n = 0$, one AS VM is provisioned only if all others are overloaded. This is ensured by lines 16-24 of the algorithm.

Similarly, to avoid charges some over-provisioned VMs should be stopped whenever their number exceeds n. However, it is not beneficial to terminate a running VM ahead of

ALGORITHM 4: Scale Up/Down Algorithm.

input : t_{cur}, tgr_{cpu}, tgr_{ram}, VM_{as}, n, Δ

1 $nOverloaded \longleftarrow 0$;
2 $listFreeVms \longleftarrow$ empty list;
3 **for** $vm \in VM_{as}$; // Inspect the status of all AS VMs
4 **do**
5 $vm_{cpu} \longleftarrow$ CPU utilisation of vm;
6 $vm_{ram} \longleftarrow$ RAM utilisation of vm;
7 **if** $vm_{cpu} \geq tgr_{cpu}$ *or* $vm_{ram} \geq tgr_{ram}$ *or networkBuffersOverloaded()* **then**
8 $nOverloaded \longleftarrow nOverloaded + 1$;
9 **else if** vm_i *serves no sessions* **then**
10 $listFreeVms.add(vm)$;
11 **end**
12 **end**
13 $nFree \longleftarrow$ length of $listFreeVms$;
14 $nAS \longleftarrow$ length of VM_{as};
15 $allOverloaded \longleftarrow nOverloaded + nFree = nAS$ and $nOverloaded > 0$;
16 **if** $nFree \leq n$; // Provision more VMs
17 **then**
18 $nVmsToStart \longleftarrow 0$;
19 **if** $allOverloaded$ **then**
20 $nVmsToStart \longleftarrow n - nFree + 1$;
21 **else**
22 $nVmsToStart \longleftarrow n - nFree$
23 **end**
24 launch $nVmsToStart$ AS VMs
25 **else**
26 $nVmsToStop \longleftarrow 0$; // Release VMs
27 **if** $allOverloaded$ **then**
28 $nVmsToStop \longleftarrow nFree - n$;
29 **else**
30 $nVmsToStop \longleftarrow nFree - n + 1$
31 **end**
32 sortAscendinglyByBillingTime($listFreeVms$);
33 **for** $i = 1$ **to** $nVmsToStop$ **do**
34 $billTime \longleftarrow$ billing time of $listFreeVms[i]$;
35 **if** $billTime - t_{cur} < \Delta$ **then**
36 terminate $listFreeVms[i]$;
37 **else**
38 break
39 **end**
40 **end**
41 **end**

its next billing time. It is better to keep it running until its billing time, in order to reuse it if resources are needed again. This is ensured by lines 25-40 of the algorithm. Firstly, we sort the free VMs in ascending order with respect to their next billing time. Next,

we iterate through the excessively allocated VMs and terminate only those for which the next billing time is earlier than the next algorithm execution time.

Lastly, in the previous discussion we assumed that the application is stateful — i.e. it maintains contextual user information in the memory of the AS server. For scalability reasons, many applications are stateless, or they store session state in an external in-memory cache like Amazon ElastiCache [6]. Algorithm 4 can handle such applications as well by considering each AS server to be assigned 0 sessions at all times. This is reflective of the main characteristic of stateless applications that each request can be served on a different server, as no session state is kept in the servers' memory. Consequently, in the algorithm all AS servers which are not overloaded will be considered free (lines 9-11) and will be viable for termination. Therefore, our approach encompasses both stateful and stateless applications.

4.5.2 Data Centre Selection

We can largely classify the requirements for data centre selection as constraints and objectives. Constraints should not be violated under any circumstances. In this chapter, we consider the following constraints: (i) users should be served in data centres which are compliant with the regulatory requirements and (ii) users should be served in data centres containing their data in the application's data layer. The system should prefer to deny service than to violate these constraints. In contrast, the system can continue to serve a user even if an objective is not optimised. We consider the following objectives: (i) cost minimisation and (ii) latency minimisation. In other words, upon a user arrival the *Entry Point* extracts the data centres which satisfy the constraints and selects the most suitable among them in terms of latency and cost.

While cost minimisation is a natural goal of cloud clients, it should not be pursued at the expense of end user Quality of Experience (QoE) and hence we must balance between the two objectives. The maximum acceptable latency between users and the data centres can be considered as a part of the application's SLA. Thus, we can choose the optimal data centre in terms of cost, whose network latency is less than the predefined in the SLA one.

Algorithm 5 implements this idea and details the data centre selection procedure. The

ALGORITHM 5: Cloud Site Selection Algorithm.

input : $users$, $timeout$, $clouds$, $latency_{SLA}$

// Broadcast users' data to admission controllers
for $c_i \in clouds$ **do**
    $ac_i \longleftarrow$ IP address of c_i's *Admission Controller*;
    send to ac_i users' identifier;
end
wait $timeout$ seconds or until all clouds respond;
for $u_i \in users$ **do**
    $clouds_{accept} \longleftarrow$ clouds eligible to serve u_i;
    sortAscendinglyByPrice($clouds_{accept}$);
    $selectedCloud \longleftarrow$ null;
    $selectedLatency \longleftarrow +\infty$;
    for $c_i \in clouds_{accept}$ **do**
        $latency \longleftarrow$ latency between u and c_i;
        if $latency < latency_{SLA}$ **then**
            $selectedCloud \longleftarrow c_i$;
            break;
        else if $selectedLatency > latency$ **then**
            $selectedCloud \longleftarrow c_i$;
            $selectedLatency \longleftarrow latency$;
        end
    end
    if $selectedCloud = null$ **then**
        Deny Service;
    else
        $lb \longleftarrow$ IP of *Load Balancer* in $selectedCloud$;
        redirect u to lb;
    end
end

algorithm selects clouds for multiple users at once. Hence, users arriving at the system at approximately the same time can be dispatched to serving clouds in batch thus avoiding excessive cross cloud communication. The input parameters of the algorithm are:

- **users** — identifiers of the users, for which the *entry point* should select a cloud.
- **timeout** — period after which if a data centre's *admission controller* has not responded it is discarded.
- **clouds** — a list of the used data centres. For each of them, we can obtain the IP addresses of the *Admission Controller* and the *load balancer*.
- $\mathbf{latency_{SLA}}$ — SLA for the network latency between a user and the serving data

centre.

In the beginning of the algorithm (lines 1-4), the *Entry Point* asynchronously broadcasts all users' identifiers to the clouds' *Admission Controllers*. After that, the *Entry Point* waits until all contacted *Admission Controllers* respond or the timeout period elapses (line 5). At this stage, unresponsive clouds whose *admissions controllers* fail to respond within the timeout are discarded.

For each user, the response of the clouds' *Admission Controllers* includes: (i) a boolean value, whether the cloud is eligible to serve the user and (ii) an estimation of the cost for serving a user. Based on this input, for every user the *Entry Point* retains the clouds eligible to serve him/her (line 7). If no eligible cloud is present the user is denied service. Otherwise, the cloud which has the smallest cost and provides latency below the SLA requirement is selected (lines 9-20). If there is no eligible cloud meeting the network latency SLA — the one with the lowest network latency to the user is selected (lines 16-19).

Note that the decision if a cloud site is eligible for a given user is application specific. For some applications with no additional privacy, security and legislation requirements all clouds may be eligible for all users. In others, certain users will have to be served within a specific legislative domain or within certified data centres based on their nationality. We assume that this application specific eligibility logic is implemented in the *Admission Controllers* by application developers.

Algorithm 5 uses the network latency between the end user and the prospective serving data centres. Hence, the *Entry Point* needs to evaluate the latencies between them based on their IP addresses. By latency we denote only the network latency. We do not try to estimate the entire response time, consisting of network and server delays. We argue this simplification does not reduce the generality of our approach, because for an interactive 3-Tier application the server delays should be small and similar in all clouds provided there is no significant resource contention. In this case, the variable part of the overall delay is the network latency. In Algorithm 4 we make sure the domain layer scales horizontally and enough resources are present at all times. If contention indeed occurs within a cloud site, because of either non-scalable DB layer or inappropriate choice of parameters for Algorithm 4, then we provide a back-off mechanism through the cost es-

timation, as discussed below. Hence, we minimise the probability of resource contention within the Multi-Cloud set-up and therefore servers' delays should be small and similar in all cloud sites.

An approximation of the network latency can be achieved in two steps. Firstly we can identify the geographical locations (longitude and latitude) of the user and a given cloud based on their IP addresses. For this we use the GeoLite [80] database, mapping IP addresses to geospatial coordinates. As a second step, we compute the latency between a user and a cloud based on the extracted coordinates by using the PingER [138] service. PingER is an end-to-end Internet performance measurement (IEPM) project, which constantly records the network metrics between more than 300 hosts positioned worldwide. The geospatial coordinates of each host are provided. To approximate the latency between a user and a cloud, we select the 3 pairs of PingER hosts that are closest to the user and the cloud respectively, and define the latency as a weighted sum of the 3 latencies between the hosts in these 3 pairs. The weights are defined proportionally to the proximity of the hosts to the user and the cloud. To compute the distance between the geospatial positions we use the well known Vincenty's formulae. The data from both GeoLite and PingER can be downloaded and used offline. If latest up-to-date Internet performance data is needed it can be periodically downloaded and updated automatically.

The last missing piece of information is the cost evaluation for serving a user by a cloud (used in line 8), performed by the *Admission Controllers*. The difficulty here is to define a unified cost evaluation for different clouds, with different pricing policies and different VM types and performance.

Firstly, if the application's infrastructure within a data centre is overloaded and it should not accept further users, it returns $+\infty$ as a cost estimation. One reason for an overload may be a lack of scalability in the DB layer. As discussed, it may not scale horizontally and given significant workload can be easily overloaded. Another reason may be a bottleneck in the data centre infrastructure — for example an internal network congestion may threaten to slow down the application's inter-tier communication. This is a less likely case, as Lloyd et al. [103] have demonstrated that for multi-tier applications the impact of the VMs' network traffic is typically negligible in comparison to the servers' performance. Internal network bottlenecks could be easily detected in the case

of a private data centre. In a public data centre obtaining such information may be more difficult, as the cloud provider would have to expose such internal performance data to its clients. By returning $+\infty$ cost to the *Entry Point* the *Admission Controller* ensures that users are sent to this cloud only as a last resort. Hence, *Admission Controllers* use the cost as a back-off mechanism.

As a first step in session cost estimation, we define $p(vm_i)$ to be the price per minute of a virtual machine vm_i. For cloud providers that charge for longer intervals (e.g. an hour like Amazon AWS) we compute this value by dividing by the number of minutes in a charge period. For each virtual machine vm_i, based on its current utilisation and the number of currently served sessions, we can approximate how many sessions $f(vm_i)$ it will be able to serve if fully utilised — see Eq. 4.1.

$$f(vm_i) = \frac{numSessions(vm_i)}{max(util_{cpu}(vm_i), util_{ram}(vm_i))} \tag{4.1}$$

Therefore, the term $p(vm_i)/f(vm_i)$ is representative of the session cost per minute in vm_i. To achieve better estimation, we average the cost estimations of all AS servers V, which currently serve sessions in the cloud. If no sessions are served in the cloud we use the last successful estimation for this data centre, or 0 if there has not been such. Eq. 4.2 summarises the previous discussion:

$$session\ cost\ per\ minute = \begin{cases} +\infty & \text{if } overloaded \\ previous\ estimation & \text{if } V = \emptyset \\ \frac{\sum_{vm_i \in V} p(vm_i)/f(vm_i)}{|V|} & \text{otherwise} \end{cases} \tag{4.2}$$

In the above discussion, for each VM vm_i we used only the price per minute $p(vm_i)$, number of sessions, and its CPU and memory utilisations. Therefore, our cost evaluation can be used even if the types of the VMs are different, as long as we can evaluate these characteristics. It is also worthwhile noting that the above cost estimation strategy is a heuristic forecast of the future cost incurred by a user, as we can not know in advance how long the user will use the system, what exactly will be his/her actions, etc.

4.5.3 Fault Tolerance

Within the given architecture, a data centre outage can be seamlessly overcome by incorporating time-outs in the *Entry Points*. If an *Admission Controller* will not reply timely to the *matchmaking broadcast*, the *Entry Point* does not consider its respective cloud.

Within a cloud, the *DC controller* manages how the AS VMs are instantiated and stopped in order to meet QoS requirements with minimal costs. Doing so, it also monitors the AS VMs in the data centre and restarts them upon failure. In this setting, it is obvious that a failure of the *DC Controller* would disable the fault tolerance and scalability of the architecture. Hence, the *Admission Controller* and the *Load Balancer* run background threads that check the status of the *DC controller* and restart it upon failure.

VMs can take up to a few minutes to boot. A failure of the *Load Balancer* and the *Admission Controller* would mean that no users can be served in this data centre during such an outage. Thus, applications requiring high availability can have multiple *Load Balancers* and *Admission Controllers* working in parallel to achieve resilience against such failure.

4.6 Performance Evaluation

Our approach to application provisioning and workload distribution is generic and testing it with all possible middleware technologies, workloads, cloud offerings, and data centre locations is an extremely laborious task. In this section, we demonstrate how under typical workload and set-up our approach meets imperative requirements like legislation compliance with only minimal losses in terms of latency and cost.

To validate our work, we use the CloudSim discrete event simulator [46], which has been used in both industry and academia for performance evaluation of cloud environments and applications. To represent 3-Tier applications in a Multi-Cloud, we use the latest CloudSim extensions from Chapter 3.

Table 4.1: Simulation parameters.

Parameter	Value	Component/Algorithm
th_{CPU}	0.7	Load balancer (AS Server Selection)
th_{RAM}	0.7	Load balancer (AS Server Selection)
tgr_{CPU}	0.7	DC Controller (Auto Scaling)
tgr_{RAM}	0.7	DC Controller (Auto Scaling)
n	1	DC Controller (Auto Scaling)
Δ	10 sec	DC Controller (Auto Scaling)
$latency_{SLA}$	30 ms	Entry point (Cloud Selection)

4.6.1 Experiment Setting

In our experimental set-up, we simulate 4 cloud data centres. We model the first two of them with the characteristics of Amazon EC2. We position one of them in Dublin, Ireland and the other one in New York, US. Later, we call these data centres *DC-EU-E* and *DC-US-E* respectively. We assign the VMs from these data centres IP address from Dublin and New York respectively, which are extracted from GeoLite. All VMs we allocate in these data centres have the performance characteristics and the price of EC2 *m1.small* instances with Linux in the respective AWS regions. We model the VM start-up times based on the empirical performance study by Mao and Humphrey [107]. Just like in Amazon EC2, the on-demand VM billing in *DC-EU-E* and *DC-US-E* is done per hour.

To demonstrate the usage of heterogeneous cloud resources from multiple cloud providers, we model the other two data centres after Google Compute Engine. We position them in Hamina, Finland and Dalles, US as these are actual locations of Google data centres and we assign all their VMs IP addresses from these locations. We call these data centres *DC-EU-G* and *DC-US-G*. All VM characteristics and prices are modelled after the *n1-standard-1-d* VM type in the respective locations. As Google Compute Engine is a new cloud offering, there is no statistical analysis of its VM booting times. Thus in our simulation we consider the start-up time of a *n1-standard-1-d* VM to be the same as the one of an EC2 *m1.small* VM. Like in Google Compute Engine, VMs in *DC-EU-G* and *DC-US-G* are billed in 1 minute increments and all VMs are charged for 10 minutes at least.

In our simulation, we deploy the aforementioned 3-Tier architecture and brokering components, as described in the previous sections. We model one *Entry Point* VM in each data centre. To demonstrate how our approach handles resource contention in the data

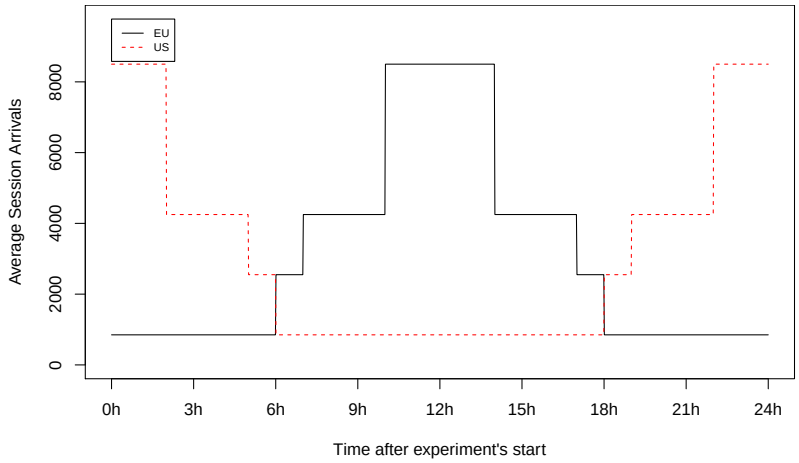


Figure 4.3: User arrival frequencies per minute in the entry points of the EU data centres (*DC-EU-E*, *DC-EU-G*) and the US ones (*DC-US-E*, *DC-US-G*).

layer, in the experiments we assume that in each cloud the DB layer is static (i.e. not scalable) and consists of 2 DB servers, holding equal in size database shards. Table 4.1 summarises the values of the algorithms' parameters that we use in the simulation.

4.6.2 Experimental Application and Workload

We base our experimental workload on the Rice University Bidding System (RUBiS) benchmarking environment [149], [16]. RUBiS implements an e-commerce dynamic website similar to eBay.com and follows the 3-Tier architectural pattern having AS and DB servers.

The RUBiS workload consists of sessions, each of which consists of a string of user requests. We have deployed in a local virtualised environment the PHP version of RUBiS with a standard non-clustered MySql database in the backend and we run a test with 100 concurrent sessions. During the test we monitor how the performance utilisations (in terms of CPU, RAM and disk) of the servers change over time as a result of the executed workload. Based on that, we define the performance utilisations over time of a typical RUBiS session. We will not describe the exact procedure for extracting a session performance model, as Chapter 3 already detailed it and demonstrated experimentally the validity of the extracted model. We use this derived session performance model for our simulation.

The number of incoming sessions over a short time period can be well modelled with

Poisson distribution with a constant mean λ [48], [145]. However, over larger time periods the frequencies of user arrivals can change and are rarely constant. Hence, the number of session arrivals over time can be represented as a Poisson distribution over a frequency function of time $\lambda(t)$, which represents the variations in session arrival frequencies.

Our experiment has a duration of 24 hours with workload which is more intensive during working hours and lower otherwise. We model the user arrival frequencies in the *Entry Points* of the two European (EU) data centres to be the same. The arrival frequencies in the US data centres are the same as those in the EU ones, only "shifted" with 12 hours to represent the time zone difference. Figure 4.3 depicts how the arrival frequencies per minute (i.e. $\lambda(t)$) in the entry points of the European and the US data centres change over time.

In our simulation, each user/session is assigned an IP address, which as explained previously can be used to approximate the user's physical location and the latencies to the candidate data centres. The GeoLite [80] database provides IP ranges for every country. In the simulation, whenever we model the arrival of a user in a US data centre, we take a random US IP from GeoLite. Similarly, all users arriving in the EU data centres are assigned random IP addresses from EU countries.

To demonstrate how our system handles regulatory requirements, we introduce an additional legislative constraint. In the simulation, we assign a citizenship to each user and impose the requirement that a user with US citizenship should be served in a US data centre and a EU citizen should be served in the EU. As discussed, we implement this logic in the *Admission Controllers*. We assign US citizenship to 10% of the users arriving in the EU entry points, and EU citizenship to 10% of the users arriving in the US clouds. Furthermore, in our simulation the data of all EU citizens is replicated in both EU data centres, and the data of all US citizens is replicated in both US cloud sites. Therefore, a EU or US citizen can be served in any EU or US data centre respectively.

4.6.3 Baseline Approach

We compare our approach to a baseline method which uses the standard industry practices. More specifically, we have implemented a baseline simulation, which distributes

the incoming users to the data centres that can serve them with the lowest latency, similarly to the Route 53 LBR service [8]. Following the design of the AWS Elastic Load Balancer [11], within each data centre we implement sticky load balancing that assigns new sessions to running AS servers following the Round-robin algorithm. Lastly, in our baseline simulation we implement automatic autoscaling following the design of AWS AustoSale [5]. More specifically, if all AS servers within a data centre reach CPU utilisation of more than 80% — a new AS server is started. If an AS server reaches CPU utilisation below 10% it is stopped. We have also implemented a *cool down* period of 2.5 minutes. Just like in AWS AustoSale, we do not allow for two consequent autoscaling actions to happen within a period shorter than the *cool down* period.

4.6.4 Results

Figure 4.4 depicts the number of served sessions in each data centre over time. In the diagram, a session is classified as *failed* if some server handling it failed (e.g. due to out of memory error). A session is *rejected* if it is assigned to a data centre which is not eligible to serve it. In our simulation, this happens if a US citizen is assigned to a European data centre or vice versa. Otherwise, a session is considered *served*.

From Figure 4.4 we can see that the baseline approach redirects much fewer sessions to data centres *DC-EU-G* and *DC-US-G* in comparison to the others. This is because of the location of these data centres and the end users. As described, in our experiment all IP addresses within EU and US are likely to be used as sources of sessions with the same probability. As to the GeoLite database, and the PingER service, there are much more IP addresses located nearby and with lower latency to Dublin, Ireland and New York, US than to Hamina, Finland and Dalles, Oregon, US. Therefore, the baseline approach redirects the majority of incoming sessions to these data centres. As the data layer cannot scale up, this leads to resource contention during peak workload periods (10h-14h in the EU data centres and 22h-24h, 0h-2h in the US). This in turn causes congestion in the DB servers resulting in slowdown in session serving. As a result, the number of concurrently served sessions is increased significantly, causing AS servers keeping in memory the sessions' states to fail with "out of memory" errors (in the case of *DC-US-E*) or to degrade response time (in the case of *DC-EU-E*).

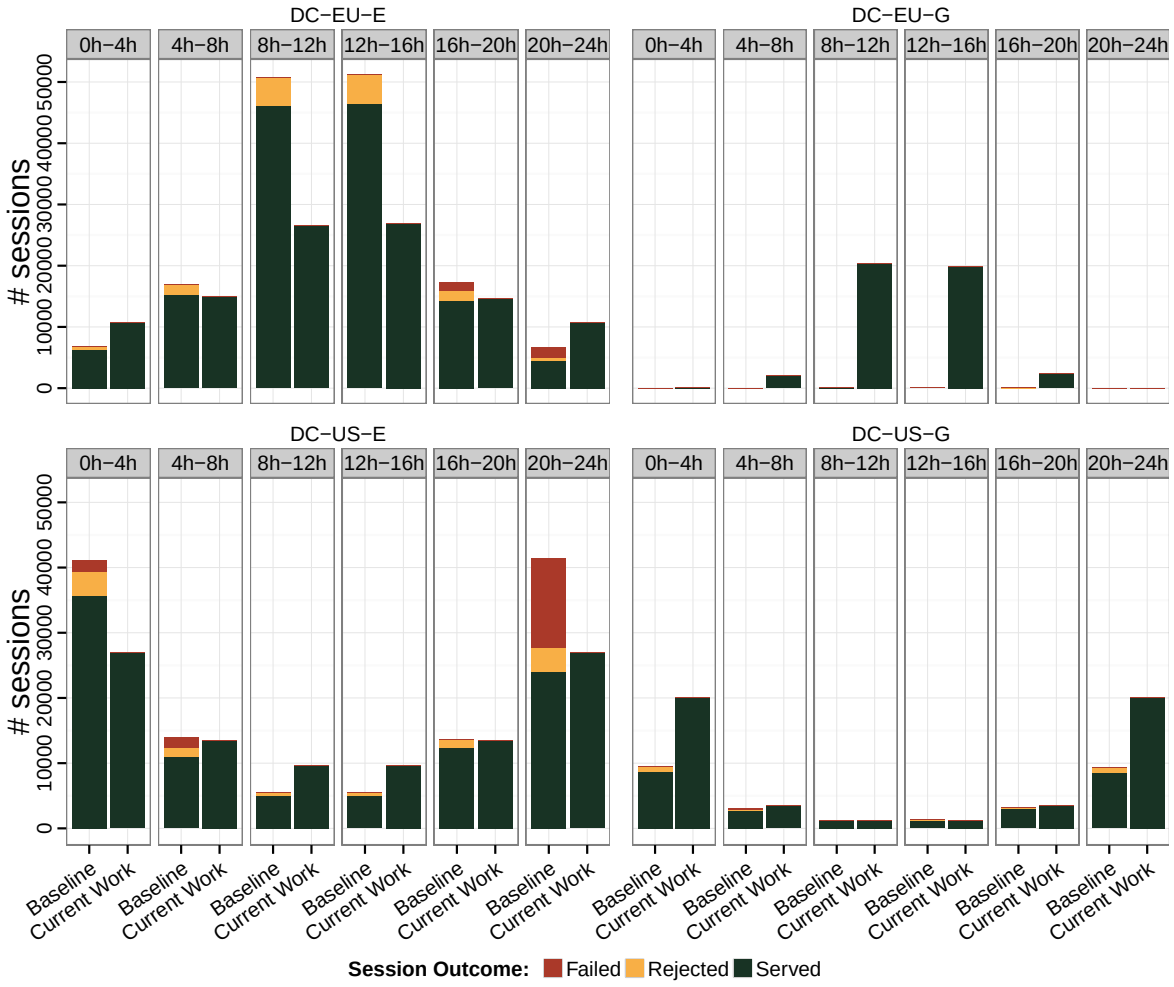


Figure 4.4: Session outcome over time.

Another reason for session failure in the baseline approach is the autoscaling, which terminates AS servers with low utilisation even if they serve sessions. This is visible in the case of *DC-EU-E* during the 16h-24h period and in *DC-US-E* during the 0h-4h period, when the scaling down causes several session failures, as sessions are stateful. Our approach terminates servers only if they do not serve any sessions, and thus reduces the number of session failures for stateful applications. Consequently, the overall rate of session failures in the baseline is approximately 7%.

In contrast to the baseline approach, during the workload peak periods our method redirects many sessions to *DC-EU-G* and *DC-US-G*, even though they may not be optimal in terms of cost or latency. This is because the cost of a data centre is evaluated as $+\infty$ if it is overloaded (see Eq. 4.2) and this is used by the cloud selection Algorithm 5. As a

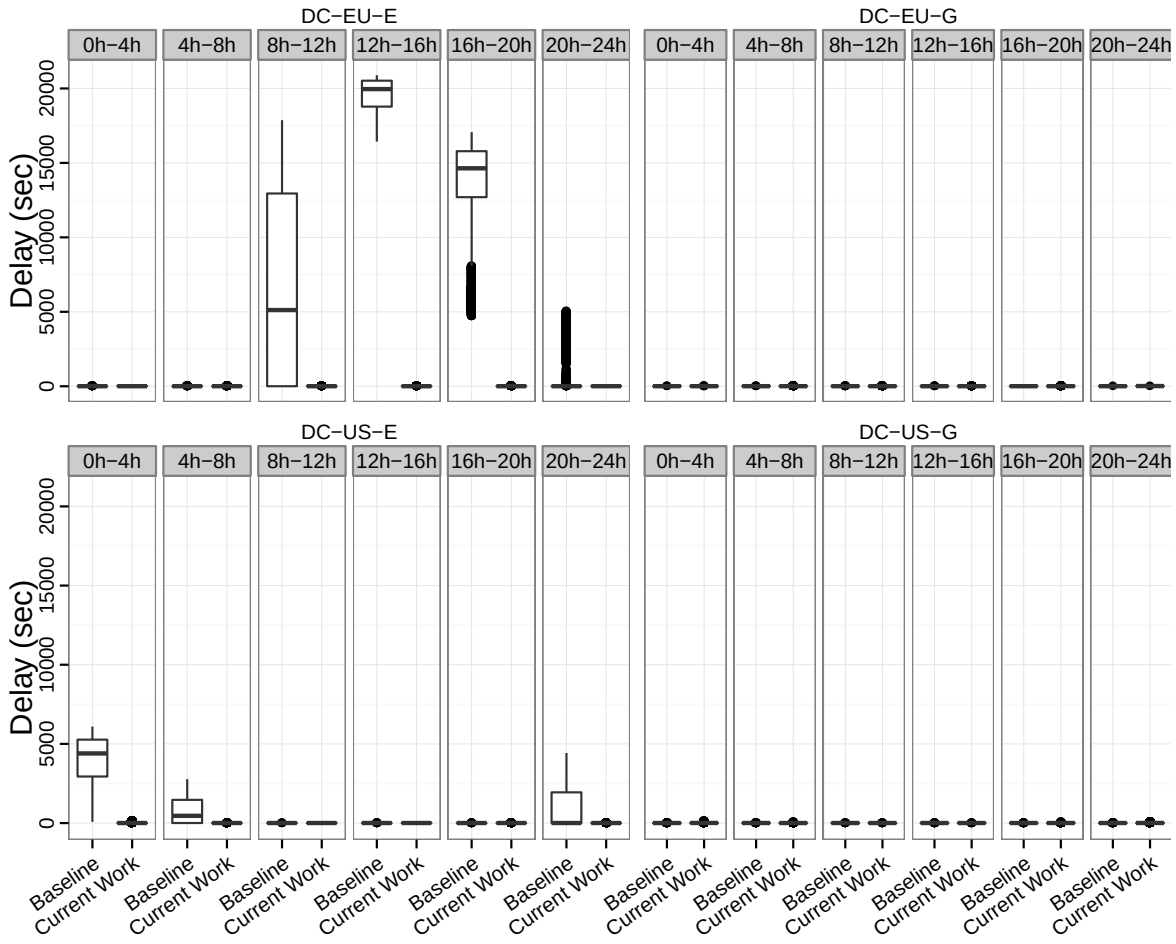


Figure 4.5: Session delays in seconds.

result, our approach minimises session failure by diverting users from overloaded data centres to alternative ones. During off-peak hours, our approach also redirects most users to *DC-EU-E* and *DC-US-E*, which as discussed is optimal in terms of latency.

Furthermore, the baseline approach does not consider the stated regulatory requirements during the cloud selection stage and only uses the latency as selection criteria. Thus, about 10% of the incoming sessions are redirected to ineligible clouds and are rejected. In contrast, our approach takes this into consideration, and redirects users to eligible data centres, even if this means sub-optimality in terms of latency and cost.

A session delay is defined as the sum of the latency delay and the execution delay. The latency delay is the time lost in network transfer between a user and the cloud during a session. Execution delay is the time lost due to resource contention (e.g. CPU preemption) on the server side. The session delay is a measurement of the end user experience.

The simulation environment allows us to measure execution delay, and we can compute the latency delay based on the latency and the average number of interactions/requests during a session.

Figure 4.5 depicts the session delays of the users served in the European and US data centres. The delays in *DC-EU-E* and *DC-US-E* during the peak workload periods of the baseline approach significantly exceed the ones of our approach. Similarly, to the session failures, this is caused by the resource contention in the DB layer. In contrast, all delays in *DC-EU-G* and *DC-US-G* are insignificant since the baseline approach redirects very few users there and therefore resource contention is small. The delays in *DC-US-E* are smaller than those in *DC-EU-E* as the failures there were much more and therefore fewer sessions were actually measured (see Figure 4.4). In our approach, the DB layer contentions are mitigated, as users are redirected to alternative data centres whenever the data layer in a given cloud is overloaded. Moreover, the overall session delays in our approach are less than 10s at all times in all data centres, showing that the effect of selecting a cloud with less than optimal latency in some cases is small.

Figure 4.6 shows the distributions of the achieved latencies between clients and clouds by the baseline and our approaches. The average baseline latency is lower with approximately 10ms than the one in our approach because the baseline method greedily selects the data centre with the lowest latency. Also, our approach honours the stated regulatory requirements and hence 10% of the users are served overseas, which contributes to the increased average latency. Still, in our approach the mean, median, and the interquartile range of the latencies are below the stated SLA of 30ms thus providing for adequate QoE. End users experience the network latency through application response delays. In our approach the average network delay is higher, but the execution delay is much lower, resulting in lower overall delay (see Figure 4.5) and better QoE.

Lastly, the overall cost incurred by our approach is about 28% more than the cost of the baseline. First, this can be attributed to the number of rejected and failed (approx. 17%) sessions in the baseline experiment. Since the baseline approach served much fewer sessions, it needed fewer servers and therefore its cost is lower. Also in order to prevent failure, our approach redirects many sessions to more expensive data centres thus resulting in increased overall cost.

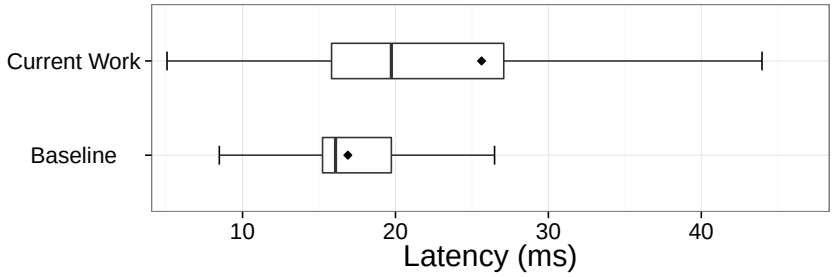


Figure 4.6: Network latency between users and serving data centres in milliseconds. The mean is denoted with a rhombus.

4.7 Summary

In this chapter, we have introduced a novel approach for adaptive resource provisioning and workload distribution of 3-Tier applications across clouds. It encompasses all aspects of resource management and workload redirection, including: (i) cloud selection, (ii) load balancing, and (iii) autoscaling. We introduced new architectural components and algorithms that ensure imperative requirements like regulation compliance and high availability are not violated without sacrificing too much cost and end-user QoE. To validate our approach, we have performed simulations with realistic cloud data centre settings, VM types, costs, and network characteristics, derived from a real-life benchmarking application, cloud providers and Internet monitoring service. We have compared our approach to a baseline method which follows current industrial best practices. Results show that our approach is a significant improvement over the baseline in terms of the achieved availability, accumulative session delay, and regulatory compliance, while maintaining acceptably low cost and latency between users and serving data centres.

In this chapter, we considered that all VMs from the domain layer are of a predefined type. Most cloud providers offer virtual machines of different types in terms of performance capacity and cost. This opens an opportunity to further improve the cost and performance efficiency of the system, by dynamically selecting appropriate VM types when scaling up. This is precisely the goal of the next chapter, in which we propose a method for automated VM type selection during autoscaling. It uses a set of machine learning approaches and heuristics to "learn" in real time the application performance utilisation patterns and to estimate the actual capacity of all VM types.

Chapter 5

Dynamic Selection of Virtual Machines for Application Servers

Autoscaling is a hallmark of cloud computing because it allows flexible just-in-time allocation and release of computational resources in response to dynamic and often unpredictable workloads. This is especially important for 3-Tier web applications whose workload is time dependent and prone to flash crowds. In this chapter, we focus on the application layer. Reactive autoscaling policies of the type "Instantiate a new Virtual Machine (VM) when the average server CPU utilisation reaches X%" have been used successfully since the dawn of cloud computing. But which VM type is the most suitable for the specific application at the moment remains an open question. In this chapter, we propose an approach for dynamic VM type selection. It uses a combination of online machine learning techniques, works in real time, and adapts to changes in the users' workload patterns, application changes as well as middleware upgrades and reconfigurations. We have developed a prototype, which we tested with the CloudStone benchmark deployed on AWS EC2. Results show that our method quickly adapts to workload changes and reduces the total cost compared to the industry standard approach.

5.1 Introduction

ORGANISATIONS relying on fixed size private infrastructures often realise it can not match their dynamic needs, thus frequently being either under or overutilised. In contrast, in a cloud environment one can automatically acquire or release resources as they are needed — a distinctive characteristic known as *autoscaling*. This is especially important for large scale web applications, since the number of users fluctuates over time and is prone to flash crowds as a result of marketing campaigns and

This chapter is based on the following publication: ***Nikolay Grozev*** *and Rajkumar Buyya, "Dynamic Selection of Virtual Machines for Application Servers in Cloud Environments", Technical Report CLOUDS-TR-2016-1, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, February 5, 2016.*

product releases.

A user interacts with the presentation layer, which redirects the requests to an Application Server (AS) which in turn can access the data layer. The presentation layer is executed on the client's side (e.g. in a browser) and thus scalability is not an issue. Scaling the DB layer is a notorious challenge, since system architects have to balance between consistency, availability and partition tolerance following the results of the CAP theorem [40, 41]. This field has already been well explored (Cattel surveys more than 20 related projects [50]). Furthermore, Google has published about their new database which scales within and across data centres without violating transaction consistency [60]. Hence, data layer scaling is beyond the scope of our work.

In general, autoscaling the Application Servers (AS) is comparatively straightforward. In an Infrastructure as a Service (IaaS) cloud environment, the AS VMs are deployed "behind" a load balancer which redirects the incoming requests among them. Whenever the servers' capacity is insufficient, one or several new AS VMs are provisioned and associated with the load balancer and the DB layer — see Figure 5.1.

But what should be the type of the new AS VM? Most major cloud providers like Amazon EC2 and Google Compute Engine offer a predefined set of VM types with different performance capacities and prices. Currently, system engineers "hardcode" preselected VM types in the autoscaling rules based on their intuition or at best on historical performance observations. However, user workload characteristics vary over time leading to constantly evolving AS capacity requirements. For example, the proportion of browsing, bidding and buying requests in an e-commerce system can change significantly during a holiday season, which can change the server utilisation patterns. Middleware and operating system updates and reconfigurations can lead to changes in the utilisation patterns as well [56]. This can also happen as a result of releasing new application features or updates.

Moreover, VM performance can vary significantly over time because of other VMs collocated on the same physical host causing resource contentions [63, 152, 160]. Hence, even VM instances of the same type can perform very differently. From the viewpoint of the cloud's client this can not be predicted.

To illustrate better, let us consider a large scale web application with hundreds of

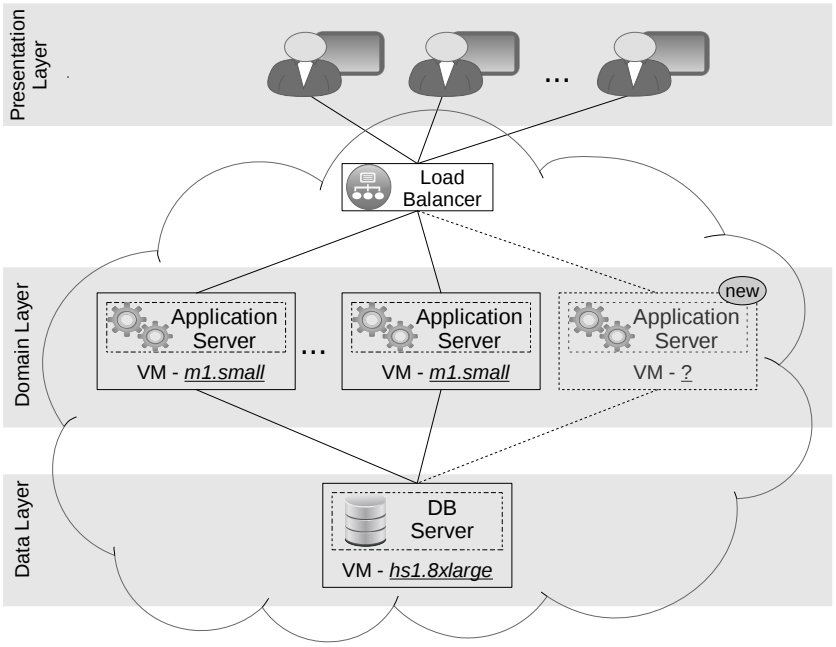


Figure 5.1: A 3-Tier application in Cloud. Whenever the autoscaling conditions are activated, a new application server should be provisioned. In this cjapter we select the optimal VM type for the purpose.

dedicated AS VMs. Its engineers can analyse historical performance data to specify the most appropriate VM type in the autoscaling rules. However, they will have to reconsider their choice every time a new feature or a system upgrade is deployed. They will also have to constantly monitor for workload pattern changes and to react by adjusting the austoscaling rules. Given that VM performance capacities also vary over time, the job of selecting the most suitable VM type becomes practically unmanageable. This can result in significant financial losses, because of using suboptimal VMs.

To address this, the key **contributions** of this chapter are (i) a machine learning approach which continuously learns the application's resource requirements and (ii) a dynamic VM type selection (DVTS) algorithm, which selects a VM type for new AS VMs. Since both workload specifics and VM performance vary over time, we propose an online approach, which learns the application's behaviour and the typical VM performance capacities in real time. It relieves system maintainers from having to manually reconfigure the autoscaling rules.

The rest of the chapter is organised as follows: In Section 5.2, we describe the related works. Section 5.3 provides a succinct overview of our approach. Section 5.4 discusses

the machine learning approaches we employ to "learn" the application's requirements in real time. Section 5.5 describes how to select an optimal VM type. Section 5.6 details the architecture of our prototype and the benchmark we use for evaluation. Section 5.7 describes our experiments and results. Finally, Section 5.8 summarises the findings.

5.2 Related Work

There have been significant efforts in developing horizontal autoscaling methods for cloud enabled applications. Lorido-Botran et al. classify autoscaling policies as *reactive* and *predictive* or *proactive* [104]. The most widely adopted *reactive* approaches are based on threshold rules for performance metrics (e.g. CPU and RAM utilisation). For each such characteristic the system administrator provides a lower and upper threshold values. Resources are provisioned whenever an upper threshold is exceeded. Similarly, if a lower threshold is reached resources are released. How much resources are acquired or released when a threshold is reached is specified in user defined autoscaling rules. There are different "flavours" of threshold based approaches. For example in Amazon Auto Scaling [5] one would typically use the average metrics from the virtual server farm, while RightScale [144] provides a voting scheme, where thresholds are considered per VM and an autoscaling action is taken if the majority of the VMs "agree" on it. Combinations and extensions of both of these techniques have also been proposed [51, 57, 153]. *Predictive* or *proactive* approaches try to predict demand changes in order to allocate or deallocate resources. Multiple methods using approaches like reinforcement learning [30, 66], queuing theory [4], and Kalman filters [78] to name a few have been proposed.

Our work is complementary to all these approaches. They indicate at what time resources should be provisioned, but do not select the resource type. Our approach selects the best resource (i.e. VM type) once it has been decided that the system should scale up horizontally.

Fernandez et al. propose a system for autoscaling web applications in clouds [72]. They monitor the performance of different VM types to infer their capacities. Our approach to this is different, as we inspect the available to each VM CPU capacity and

measure the amount of "stolen" CPU instructions by the hypervisor from within the VM itself. This allows us to normalise the VMs' resource capacities to a common scale, which we use to compare them and for further analysis. Furthermore, their approach relies on a workload predictor, while ours is usable even in the case of purely reactive autoscaling.

Singh et al. use k-means clustering to analyse the workload mix (i.e. the different type of sessions) and then use a queueing model to determine each server's suitability [154]. However, they do not consider the performance variability of virtual machines, which we take into account. Also, they do not select the type of resource (e.g. VM) to provision and assume there is only one type, while this is precisely the focus of this chapter.

A part of our work is concerned with automated detection of application behaviour changes through a Hierarchical Temporal Memory (HTM) model. Similar work has been carried out by Cherkasova et al. [56], who propose a regression based anomaly detection approach. However, they analyse only the CPU utilisation. Moreover, they consider that a set of user transactions' types is known beforehand. In contrast, our approach considers RAM as well and does not require application specific information like transaction types. Tan et al. propose the PREPARE performance anomaly detection system [159]. However, their approach can not be used by a cloud client, as it is built on top of the Xen virtual machine manager to which external clients have no access.

Another part of our method is concerned with automatic selection of the *learning rate* and *momentum* of an Artificial Neural Network (ANN). There is a significant amount of literature in this area as surveyed by Moreira and Fiesler [114]. However, the works they overview are applicable for static data sets and have not been applied to learning from streaming online data whose patterns can vary over time. Moreover, they only consider how the intermediate parameters of the backpropagation algorithm vary and do not use additional domain specific logic. Although our approach is inspired by the work of Vogl et al. [167] as it modifies the *learning rate* and *momentum* based on the prediction error, we go further and we modify them also based on the *anomaly score* as reported by the Hierarchical Temporal Memory (HTM) models.

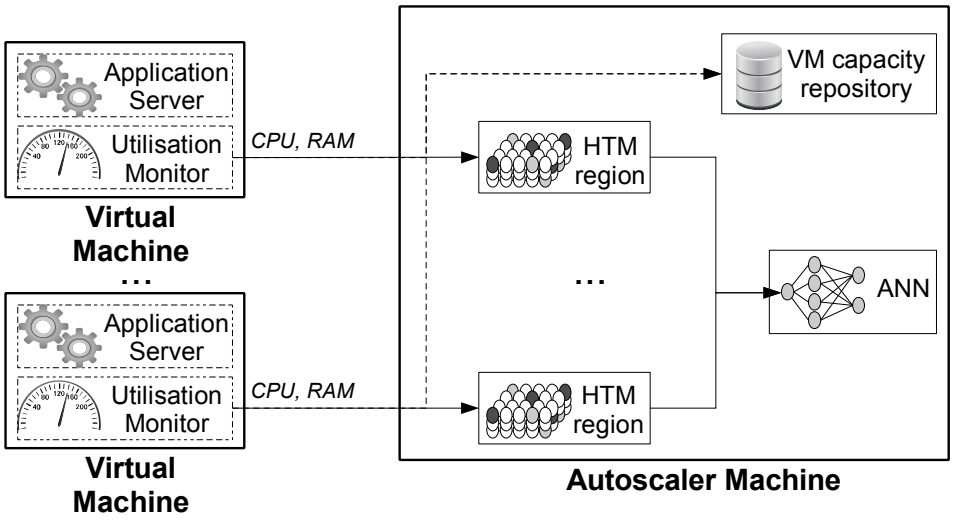


Figure 5.2: System components and their interaction.

5.3 Method Overview

Figure 5.2 depicts an overview of our machine learning approach and how the system components interact. Within each AS VM we install a monitoring program which periodically records utilisation metrics. These measurements are transferred to an *autoscaling component*, which can be hosted either in a cloud VM or on-premises. It is responsible for (i) monitoring AS VMs' performance (ii) updating machine learning models of the application behaviour, and (iii) autoscaling.

Within each AS VM the *utilisation monitors* report statistics about the CPU, RAM, disk and network card utilisations and the number of currently served users. These records are transferred every 5 seconds to the *autoscaling* component, where they are normalised, as different VMs have different de facto resource capacities. In the machine learning approaches, we only consider the CPU and RAM utilisations, as disk and network utilisations of AS VMs are typically small [103].

For each AS VM the *autoscaler* maintains a separate single-region Hierarchical Temporal Memory (HTM) model [86], which is discussed in a later section. In essence, we use HTMs to detect changes in the application behaviour of each AS VM. We prefer HTM to other regression based anomaly detection approaches, as it can detect anomalies on a stream of multiple parameters (e.g. CPU and RAM). Whenever monitoring data is retrieved from an AS VM, the *autoscaler* trains its HTM with the received number of users,

CPU and RAM utilisations and outputs an *anomaly score* defining how "unexpected" the data is.

As a next step, we use these utilisation measurements to train a 3-Tier artificial neural network (ANN) about the relationship between the number of served users and resource consumptions. We choose to use an ANN because of its suitability for online data streams. Other simpler "sliding window" approaches operate only on a portion of the data stream. As a system's utilisation patterns can remain the same for long time intervals, these approaches' accuracies can be very limited. On the contrary, an ANN does not operate on a fixed time window and is more adept with changes in the incoming data stream, as we will detail in a later section.

There is only one ANN and training samples from all AS VMs are used to train it. In essence, the ANN represents a continuously updated regression model, which given a number of users predicts the needed resources to serve them within a single VM without causing resource contentions. Thus, we need to filter all training samples, which were taken during anomalous conditions (e.g. insufficient CPU or RAM capacity causing intensive context switching or disk swapping respectively). Such samples are not indicative of the relationship between number of users and the resource requirements in the absence of resource contentions. Furthermore, we use the *anomaly score* of each training sample (extracted from HTM) to determine the respective *learning speed* and *momentum* parameters of the back propagation algorithm so that the ANN adapts quickly to changes in the utilisation patterns.

Training the ANN and the HTMs happens online from the stream of VM measurements in parallel with the running application. Simultaneously, we also maintain a *VM capacity repository* of the latest VM capacity measurements. When a new VM is needed by the autoscaling component, we use this repository to infer the potential performance capacity of all VM types. At that time the ANN is already trained adequately and given the predicted performance capacities can be used to infer how many users each VM type could serve simultaneously. Based on that we select the VM type, with minimal cost to number of users ratio.

5.4 Learning Application Behaviour

5.4.1 Resource Monitoring

To measure VM performance utilisation, we use the *SAR*, *mpstat*, *vmstat*, and *netstat* Linux monitoring tools. We use the mpstat *%idle* metric to measure the percentage of time during which the CPU was idle. The *%steal* metric describes the percentage of "stolen" CPU cycles by a hypervisor (i.e. the proportion of time the CPU was not available to the VM) and can be used to evaluate the actual VM CPU capacity. Similarly, SAR provides the *%util* and *%ifutil* metrics as indicative of the disk's and network card's utilisations.

Measuring the RAM utilisation is more complex as (i) operating systems keep in memory cached copies of recently accessed disk sectors in order to reduce disk access and (ii) many middleware technologies do not release memory until garbage collection, as discussed in Chapter 3. Hence, using the *vmstat* RAM utilisation metrics can be an overestimation of the actual memory consumption. Thus, we use the *"active memory" vmstat* metric to measure memory consumption instead. It denotes the amount of recently used memory, which is unlikely to be claimed for other purposes.

Lastly, we need to evaluate the number of concurrently served users in an AS VM. This could be extracted from the AS middleware, but that would mean writing specific code for each type of middleware. Moreover, some proprietary solutions may not expose this information. Therefore, we use the number of distinct IP addresses with which the server has an active TCP socket, which can be obtained through the *netstat* command. Typically, the AS VM is dedicated to running the AS and does not have other outgoing connections except for the connection to the persistence layer. Therefore, the number of addresses with active TCP sockets is a good measure of the number of currently served users.

5.4.2 Normalisation and Capacity Estimation

Before proceeding to train the machine learning approaches, we need to normalise the measurements which have different "scales", as the VMs have different RAM sizes and CPUs with different frequencies. Moreover, the actual CPU capacities within a single VM

vary over time as a result of the dynamic collocation of other VMs on the same host.

As a first step in normalising the CPU load, we need to evaluate the actual CPU capacity available to each VM. This can be extracted from the */proc/cpuinfo* Linux kernel file. If the VM has n cores, */proc/cpuinfo* will list meta information about the physical CPU cores serving the VM including their frequencies $fr_1, ...fr_n$. The sum of these frequencies is the maximal processing capacity the VM can get, provided the hypervisor does not "steal" any processing time. Using the *%steal* mpstat parameter we can actually see what percentage of CPU operations have been taken away by the hypervisor. Subtracting this percentage from the sum of frequencies gives us the actual VM CPU capacity at the time of measurement. To normalise we further divide by the maximal CPU core frequency fr_{max} multiplied by the maximal number of cores n_{max_cores} of all considered VMs in the cloud provider. This is a measure of the maximal VM CPU capacity one can obtain from the considered VM types. As clouds are made of commodity hardware, we will consider $fr_{max} = 3.5GHZ$. This is formalised in Eq. 5.1.

$$cpuCapacityNorm = \frac{(100 - \%steal) \sum_{i=0}^{n} fr_i}{100\ n_{max_cores}\ fr_{max}} \tag{5.1}$$

Having computed the VM CPU capacity, we store it into the *VM capacity repository*, so we can use it later on to infer the capacities of future VMs. Each repository record has the following fields:

- **time** — a time stamp of the capacity estimation;
- **vm-type** — an identifier of the VM type - e.g. "m1.small";
- **vm-id** — a unique identifier of the VM instance - e.g. its IP or elastic DNS address;
- **cpuCapacityNorm** — the computed CPU capacity.

If we further subtract the *%idle* percentage from the capacity we will get the actual CPU load given in Eq. 5.2.

$$cpuLoadNorm = \frac{(100 - \%idle - \%steal) \sum_{i=0}^{n} fr_i}{100\ n_{max_cores}\ fr_{max}} \tag{5.2}$$

Normalising the RAM load and capacity is easier, as they do not fluctuate like the CPU capacity. We divide the *active memory* by the maximal amount of memory RAM_{max} in all considered virtual machine types in the cloud — see Eq. 5.3.

$$ramLoadNorm = \frac{active_memory}{RAM_{max}} \tag{5.3}$$

Whenever a new AS VM is needed, we have to estimate the CPU and RAM capacities of all available VM types based on the *capacity repository* and their performance definitions provided by the provider. The normalised RAM capacity of a VM type is straightforward to estimate as we just need to divide the capacity in the provider's specification by RAM_{max}. To estimate the CPU capacity of a VM type we use the mean of the last 10 entries' capacities for this type in the *capacity repository*. If there are no entries for this VM type in the repository (i.e. no VM of this type has been instantiated) we can heuristically extrapolate the CPU capacity from the capacities of the other VM types. Typically, IaaS providers specify an estimation of each VM type's CPU capacity — e.g. Google Compute Engine Units (GCEU) in Google Compute Engine or Elastic Compute Units (ECU) in AWS. Hence, given an unknown VM type vmt we can extrapolate its normalised CPU capacity as:

$$cpuCapacity(vmt) = \frac{1}{|V|} \sum_{vmt_i \in V} \frac{cpuCapacity(vmt_i) \times cpuSpec(vmt_i)}{cpuSpec(vmt)} \tag{5.4}$$

where V is the set of VM types present in the *capacity repository* and whose CPU capacity can be determined from previous measurements. $cpuSpec(vmt_i)$ defines the cloud provider's estimation of a VM type's capacity — e.g. number of GCEUs or ECUs.

5.4.3 Anomaly Detection Through HTM

The Hierarchical Temporal Memory (HTM) model is inspired by the structure and organisation of the neocortex. It has been developed and commercialised by the Grok company [83] (formerly Numenta [119]), following the concepts from Jeff Hawkins' book "On Intelligence" [87]. Hawkins builds upon the seminal work of Mountcastle [115] that the neocortex is predominantly uniform in structure and function even in regions handling different sensory inputs — e.g. visual, auditory, and touch. The HTM model tries to

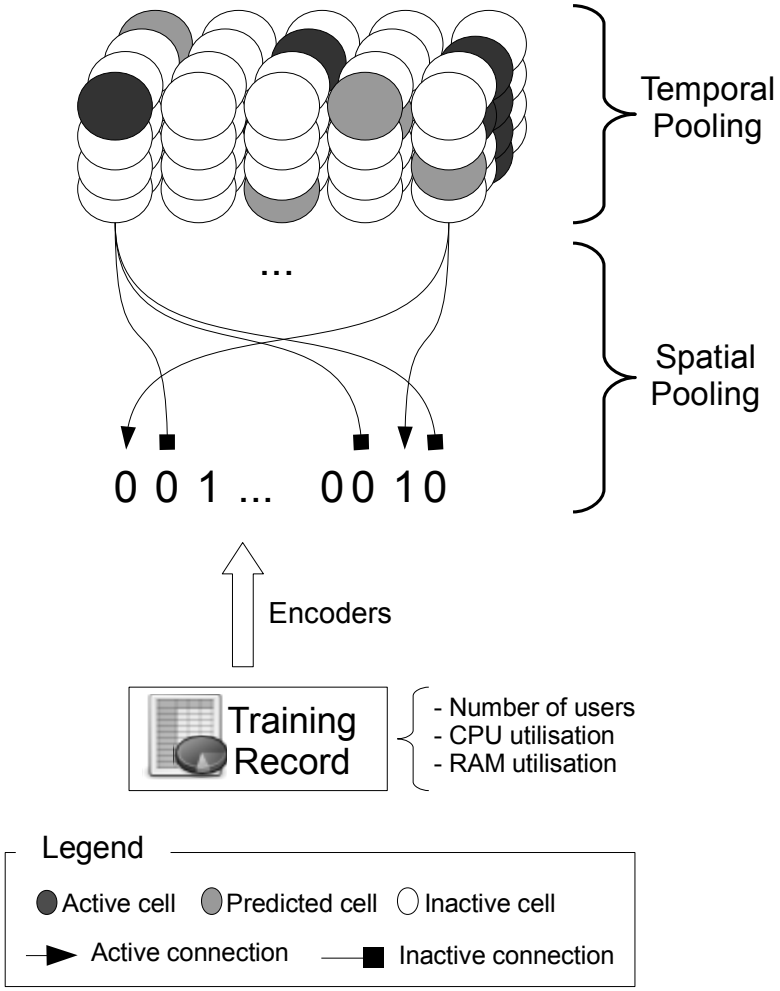


Figure 5.3: HTM region structure.

mimic this structure in a computational model. There are several differences compared to the biological structure of the neocortex in order to be computationally viable as described in the implementation white paper [86]. Grok's implementation is available as an open source project called NuPIC [119]. In this section, we provide only a brief overview of HTM to introduce the reader to this concept. The interested reader is referred to the official documentation [86].

HTMs consist of one or several stacked regions. During inference, input arrives into the lowest region, whose output serves as input to the successive one and so forth until the topmost region outputs the final result. The purpose of a region is to convert noisy input sequences to more stable abstract representations. In this chapter, we use single-region HTMs and we will focus on them in the rest of the section.

A HTM region consists of columns of cells, which are most often arranged in a three dimensional grid — see Figure 5.3. Each cell can be in one of three possible states: (i) active form feed forward input, (ii) active from lateral input (i.e. predicted), or (iii) inactive. Conceptually, active cells represent the state of the last input and predicted cells represent the likely state after future inputs. A HTM region receives as input a number of bits. Special *encoders* are used to convert input objects into bitwise representations, so that objects which are "close" in the sense of the target domain have similar bit representations. Upon receiving new binary input the HTM changes the states of the columns based on several rules summarised below.

As a first step, the HTM has to decide which columns' cells will be activated for a given input — an algorithm known as *Spatial Pooling*. It nullifies most of the 1 bits, so that only a small percentage (by default 2%) are active. Each column is connected with a fixed sized (by default 50% of the input length) random subset of input bits called the *potential pool*. Each column's connection to an input bit has a ratio number in the range [0,1] associated with it known as the *permanence*. HTM automatically adjusts the *permanence* value of a connection after a new input record arrives, so that input positions whose value have been 0 or 1 and are members of the *potential pool* of a selected column are decreased or increased respectively. Connections with *permanences* above a predefined thresholds are considered active. Given an input, for each column the HTM defines its *overlap score* as the number of active bits with active connections. Having computed this for every column, HTM selects a fixed sized (by default 2%) set of columns with the highest *overlap score*, so that no two columns within a predefined radius are active.

As a second step, HTM decides which cells within these columns to activate. This is called *Temporal Pooling*. Within each of the selected columns the HTM activates only the cells which are in *predicted* state. If there are no cells in predicted state within a column, then all of its cells are activated, which is also known as *bursting*.

Next, the HTM makes a prediction of what its future state will be — i.e. which cells should be in predicted state. The main idea is that when a cell activates it establishes connections to the cells which were previously active. Each such connection is assigned a weight number. Over time, if the two nodes of a connection become active in sequence again, this connection is strengthened, i.e. the weight is increased. Otherwise, the con-

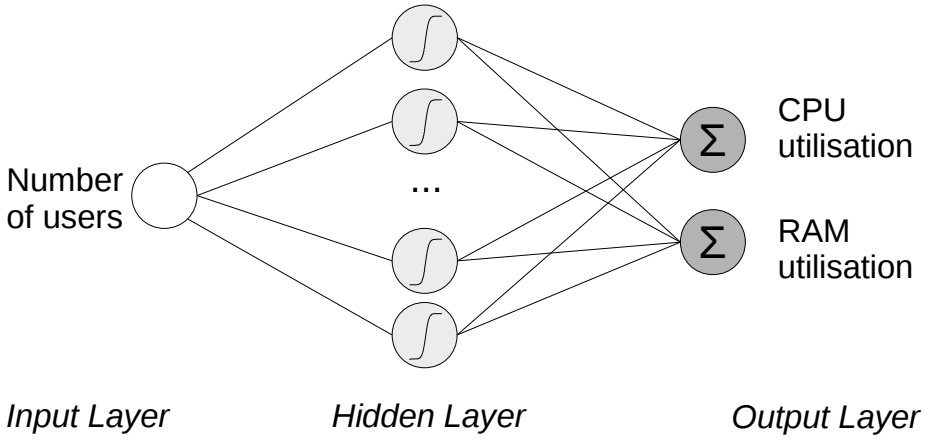


Figure 5.4: ANN topology.

nection slowly decays, i.e. the weight is gradually decreased. Once a cell becomes active, all non-active cells having connections to it with weights above a certain threshold are assigned the predicted state. This is analogous to how synapses form and decay between neurons' dendrites in the neocortex in response to learning patterns.

The presence of predicted cell columns allows a HTM to predict what will be its likely state in terms of active cells after the next input. Moreover, it also allows for the detection of anomalies. For example, if just a few predicted states become active this is a sign that the current input has not been expected. Thus, the *anomaly_score* is defined as the proportion of active spatial pooler columns that were incorrectly predicted and is in the range $[0, 1]$.

In our environment, every 5 seconds we feed each HTM with a time stamp, the number of users and the CPU and RAM utilisations of the respective VM. We use the standard NuPIC scalar and date encoders to convert the input to binary input. As a result we get an *anomaly score* denoting how expected the input is, in the light of the previously described algorithms.

5.4.4 ANN Training

Figure 5.4 depicts the topology of the artificial neural network (ANN). It has one input — the number of users. The hidden layer has 250 neurons with the sigmoid activation function. The output layer has two output nodes with linear activation functions, which

predict the normalised CPU and RAM utilisations within an AS VM.

Once a VM's measurements are received and normalised and the *anomaly score* is computed by the respective HTM region, the ANN can be trained. As discussed, we need to filter out the VM measurements which are not representative of normal, contention free application execution, in order to model the relationship between number of users and resource utilisations. We filter all VM measurements in which the CPU, RAM, hard disk, or network card utilisations are above a certain threshold (e.g. 70%). Similarly, we filter measurements with negligible load — i.e. less than 25 users or less than 10% CPU utilisation. We also ignore measurements from periods during which the number of users has changed significantly — e.g. in the beginning of the period there were 100 users and at the end there were 200. Such performance observations are not indicative of an actual relationship between number of users and resource utilisations. Thus, we ignore measurements for which the number of users is less than 50% or more than 150% of the average of the previous 3 measured numbers of users from the same VM.

Since we are training the ANN with streaming data, we need to make sure it is not overfitted to the latest training samples. For example, if we have constant workload for a few hours we will be receiving very similar training samples in the ANN during this period. Hence, the ANN can become overfitted for such samples and lose its fitness for the previous ones. To avoid this problem, we filter out measurements/training samples, which are already well predicted. More specifically, if a VM measurement is already predicted with a *root mean square error* (RMSE) less than 0.01 it is filtered out and the ANN is not trained with it. We call this value $rmse^{pre}$ because it is obtained for each training sample before the ANN is trained with it. It is computed as per Eq. 5.5, where $output_i$ and $expected_i$ are the values of the output neurons and the expected values respectively.

$$rmse^{pre} = \sqrt{\sum(output_i - expected_i)^2} \tag{5.5}$$

With each measurement, which is not filtered out, we perform one or several iterations/epochs of the back-propagation algorithm with the number of users as input and the normalised CPU and RAM utilisations as expected output. The back-propagation algorithm has two important parameters — the *learning rate* and the *momentum*. In essence, the *learning rate* is a ratio number in the interval $(0, 1)$ which defines the amount of weight

update in the direction of the gradient descent for each training sample [114]. For each weight update, the *momentum* term defines what proportion of the previous weight update should be added to it. It is also a ratio number in the interval $(0, 1)$. Using a *momentum* the neural network becomes more resilient to oscillations in the training data by "damping" the optimisation procedure [114].

For our training environment, we need a low *learning rate* and a high *momentum*, as there are many oscillations in the incoming VM measurements. We select the *learning rate* to be $lr = 0.001$ and the *momentum* $m = 0.9$. We call these values the *ideal parameters*, as these are the values we would like to use once the ANN is close to convergence. However, the low *learning rate* and high *momentum* result in slow convergence in the initial stages, meaning that the ANN may not be well trained before it is used. Furthermore, if the workload pattern changes, the ANN may need a large number of training samples and thus time until it is tuned appropriately. Hence, the actual *learning rate* and *momentum* must be defined dynamically.

One approach to resolve this is to start with a high *learning rate* and low *momentum* and then respectively decrease/increase them to the desired values [114,167]. This allows the back-propagation algorithm to converge more rapidly during the initial steps of the training. We define these parameters in the initial stages using the asymptotic properties of the sigmoid function, given in Eq. 5.6.

$$s(x) = \frac{1}{1 - e^{-x}} \tag{5.6}$$

As we need to start with a high *learning rate* and then decrease it gradually to lr, we could define the learning rate lr_k for the k-th training sample as $s(-k)$. However, the sigmoid function decreases too steeply for negative integer parameters and as a result the learning rate is higher than lr for just a few training samples. To solve this, we use the square root of k instead and thus our first approximation of the *learning rate* is:

$$lr_k^{(1)} = max(lr, s(-\sqrt{k})) \tag{5.7}$$

As a result $lr_k^{(1)}$ gradually decreases as more training samples arrive. Figure 5.5 depicts how it changes over time.

We also need to ensure that it increases in case unusual training data signalling a workload change arrives and thus we need to elaborate $lr_k^{(1)}$. For this we keep a record of the last 10 samples' *anomaly scores* and errors (i.e. $rmse^{pre}$). The higher the latest anomaly scores, the more "unexpected" the samples are and therefore the *learning rate* must be increased. Similarly, the higher the sample's $rmse^{pre}$ compared to the previous errors, the less fit for it the ANN is and thus the *learning rate* must be increased as well. Thus, our second elaborated approximation of the *learning rate* is:

$$lr_k^{(2)} = lr_k^{(1)}\ max(1, \frac{rmse_k^{pre}}{\overline{rmse}}) \prod_{i=0}^{9} 2s(an_{k-i}) \tag{5.8}$$

where an_k and $rmse_k^{pre}$ are the *anomaly score* and the error of the k-th sample and $\overline{rmse}$ is the average error of the last 10 samples. Note that we use the sigmoid function for the anomaly scores to diminish the effect of low values.

In some cases the *learning rate* can become too big in the initial training iterations, which will in fact hamper the convergence. To overcome this problem, for each sample k we run a training iteration with $lr_k^{(2)}$, compute its RMSE $rmse_k^{post}$ and then revert the results of this iteration. By comparing $rmse_k^{pre}$ and $rmse_k^{post}$ we can see if training with this $lr_k^{(2)}$ will contribute to the convergence [167]. If not, we use the ideal parameter lr instead. Thus we finally define the *learning rate* parameter lr_k in Eq. 5.9:

$$lr_k = \begin{cases} lr_k^{(2)} & \text{if } rmse_k^{pre} > rmse_k^{post} \\ lr & \text{otherwise} \end{cases} \tag{5.9}$$

Similarly, we have to gradually increase the *momentum* as we decrease the *learning rate* until the ideal *momentum* is reached. If a workload change is present we need to decrease the *momentum* in order to increase the learning speed. Hence, we can just use the ratio of the ideal learning rate lr to the current one as shown in Eq. 5.10.

$$m_k = min(m, \frac{lr}{lr_k^{(2)}}) \tag{5.10}$$

Figure 5.5 depicts how the *learning rate* and *momentum* change during the initial training stages, given there are no anomalies, accuracy losses and $\forall k : rmse_k^{pre} > rmse_k^{post}$ — i.e. when $\forall k : lr_k^{(1)} = lr_k^{(2)} = lr_k$. Figure 5.7 shows the actual lr_k given realistic workload.

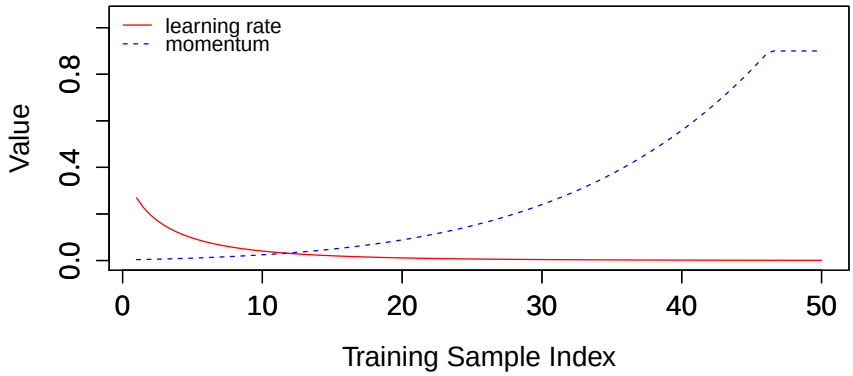


Figure 5.5: The $lr_k^{(1)}$ approximation of the *learning rate* and the respective momentum during the initial ANN training stages.

Furthermore, to speed up convergence it is beneficial to run multiple *epochs* (i.e. repeated training iterations) with the first incoming samples and with samples taken after a workload change. The ideal *learning rate* lr and its approximation $lr_k^{(2)}$ already embody this information and we could simply use their ratio. However, $\frac{lr_k^{(2)}}{lr}$ can easily exceed 300 given $lr = 0.001$, resulting in over-training with particular samples. Hence, we take the logarithm of it as in Eq. 5.11:

$$e_k = \left\lfloor 1 + ln(\frac{lr_k^{(2)}}{lr}) \right\rfloor \tag{5.11}$$

5.5 Virtual Machine Type Selection

When a new VM has to be provisioned the ANN should be already trained so that we can estimate the relationship between number of users and CPU and RAM requirements. The procedure is formalised in Algorithm 6. We loop over all VM types VT (line 3) and for each one we estimate its normalised CPU and RAM capacity based on the *capacity repository* as explained earlier (lines 5-6). The VM cost per time unit (e.g. hour in AWS or minute in Google Compute Engine) is obtained from the provider's specification (line 7).

Next, we approximate the number of users that a VM of this type is expected to be able to serve (lines 10-18). We iteratively increase n by Δ starting from $minU$, which is the minimal number of users we have encountered while training the neural network. We use the procedure *predict* (defined separately in Algorithm 7) to estimate the normalised

ALGORITHM 6: Dynamic VM Type Selection (DVTS).

```
   input : VT, ann, Δ, minU, maxU
1  bestVmt ⟵ null;
2  bestCost ⟵ 0;
3  for vmt ∈ VT ;                                   // Inspect all VM types
4  do
5      cpuCapacity ⟵ vmt's norm. CPU capacity ;
6      ramCapacity ⟵ vmt's norm. RAM capacity;
7      vmtCost ⟵ vmt's cost per time unit;
8      userCapacity ⟵ 0;
9      n ⟵ minU;
10     while True ;                                 // Find how many users it can take
11     do
12         cpu, ram ⟵ predict(ann, n, minU, maxU);
13         if cpu < cpuCapacity and ram < ramCapacity then
14             userCapacity ⟵ n;
15         else
16             break;
17         end
18         n ⟵ n + Δ;
19     end
       // Approximate the cost for a user per time unit
20     userCost ⟵ vmtCost / userCapacity;
       // Find the cheapest VM type
21     if userCost < bestCost then
22         bestCost ⟵ userCost;
23         bestVmt ⟵ vmt;
24     end
25 end
26 return bestVmt;
```

CPU and RAM demands that each of these values of n would cause. We do so until the CPU or RAM demands exceed the capacity of the inspected VM type. Hence, we use the previous value of n as an estimation of the number of users a VM of that type can accommodate. Finally, we select the VM type with the lowest cost to number of users ratio (lines 20-23).

Algorithm 7 describes how to predict the normalised utilisations caused by n concurrent users. If n is less than the maximum number of users $maxU$ we trained the ANN with, then we can just use the ANN's prediction (line 5). However, if n is greater than $maxU$ the ANN may not predict accurately. For example, if we have used a single *small* VM to train the ANN, and then we try to predict the capacity of a *large* VM, n can become much larger than the entries of the training data and the regression model may be inac-

ALGORITHM 7: Resource Utilisation Estimation.

```
   input : ann, n, minU, maxU
1  cpu ⟵ 0;
2  ram ⟵ 0;
3  if n < maxUsers ;                                  // If within range - use ANN
4  then
5  |  cpu, ram ⟵ ann.run(n);
6  else
   |  // If outside range - extrapolate
7  |  minRam, minCPU ⟵ ann.run(minU);
8  |  maxRam, maxCPU ⟵ ann.run(maxU);
9  |  cpuPerUser ⟵ (maxCPU−minCPU)/(maxU−minU);
10 |  ramPerUser ⟵ (maxRam−minRam)/(maxU−minU);
11 |  cpu ⟵ maxCPU + cpuPerUser(n − maxU);
12 |  ram ⟵ maxCPU + ramPerUser(n − maxU);
13 end
14 return cpu, ram;
```

curate. Thus, we extrapolate the CPU and RAM requirements (lines 7-11) based on the range of values we trained the ANN with and the performance model we have proposed in Chapter 3.

5.6 Benchmark and Prototype

There are two main approaches for experimental validation of a distributed system's performance — through a simulation or a prototype. Discrete event simulators like CloudSim [46] have been used throughout industry and academia to quickly evaluate scheduling and provisioning approaches for large scale cloud infrastructure without having to pay for expensive test beds. Unfortunately, such simulators work on a simplified cloud performance model and do not represent realistic VM performance variability, which is essential for testing our system. Moreover, simulations can be quite inaccurate when the simulated system serves resource demanding workloads, as they do not consider aspects like CPU caching, disk data caching in RAM and garbage collection, as we discussed in Chapter 3. Therefore, we test our method through a prototype and a standard benchmark deployed in a public cloud environment.

We validate our approach with the CloudStone [59, 155] web benchmark deployed in Amazon AWS. It follows the standard 3-Tier architecture. By default, CloudStone

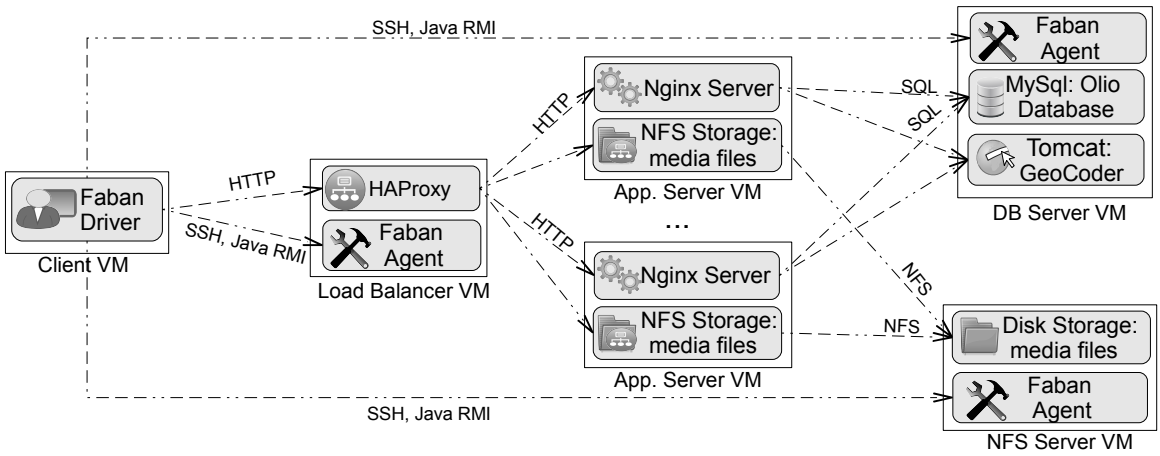


Figure 5.6: CloudStone benchmark's extended topology.

is not scalable, meaning that it can only use a single AS. Thus, we had to extend it to accommodate multiple servers. Our installation scripts and configurations are available as open source code. The interested readers can refer to our online documentation and installation instructions.[1]

The benchmark deployment topology is depicted in Figure 5.6. CloudStone uses the *Faban* harness to manage the runs and to emulate users. The *faban driver*, which is deployed in the client VM communicates with the *faban agents* deployed in other VMs to start or stop tests. It also emulates the incoming user requests to the application. These requests arrive at a HAProxy *load balancer* which distributes them across one or many application servers (AS). CloudStone is based on the Olio application, which is a PHP social network website deployed in a Nginx server. In the beginning we start with a single AS "behind" the *load balancer*. When a new AS VM is provisioned we associate it with the *load balancer*. We update its weighted round robin policy, so that incoming request are distributed among the AS VMs proportionally to their declared CPU capacity (i.e. ECU).

The persistent layer is hosted in a MySql server deployed within a separate DB VM. CloudStone has two additional components — (i) a geocoding service called *GeoCoder*, hosted in an Apache Tomcat server, and (ii) a shared *file storage* hosting media files. They are both required by all application servers. We have deployed the geocoding service in the DB VM. The file storage is deployed in a Network File System (NFS) server on a

[1] http://nikolaygrozev.wordpress.com/2014/06/02/advanced-automated-cloudstone-setup-in-ubuntu-vms-part-2/

separate VM with 1TB EBS storage, which is mounted from each AS VM.

We use "m3.medium" VMs for the client, load balancer and DB server and "m1.small" for the NFS server. The types of the AS VMs are defined differently for each experiment. All VMs run 64bit Ubuntu Linux 14.04.

Our prototype of an autoscaling component is hosted on an on-premises physical machine and implements the previously discussed algorithms and approaches. It uses the JClouds [21] multi-cloud library to provision resources, and thus can be used in other clouds as well. We use the NuPIC [120] and FANN [71] libraries to implement HTM and ANN respectively. We ignore the first 110 *anomaly scores* reported from the HTM, as we observed that these results are inaccurate (i.e. always 1 or 0) until it receives initial training. Whenever a new AS VM is provisioned we initialise it with a deep copy of the HTM of the first AS VM, which is the most trained one. The monitoring programs deployed within each VM are implemented as bash scripts, and are accessed by the autoscaling component through SSH. Our implementation of Algorithm 7 uses $\Delta = 5$.

Previously we discussed that the number of current users could be approximated by counting the distinct IP addresses to which there is an active TCP session. However, in CloudStone all users are emulated from the same client VM and thus have the same source IP address. Thus, we use the number of recently modified web server session files instead.

Our autoscaling component implementation follows the Amazon Auto Scaling [5] approach and provisions a new AS VM once the average CPU or RAM utilisations of the server farm reaches 70% for more than 10 seconds. Hence, we ensure that in all experiments the AS VMs are not overloaded. Thus, even if there are SLA violations, they are caused either by the network or the DB layer, and the AS layer does not contribute to them. We also implement a *cool down* period of 10 minutes.

5.7 Validation

In our experiments, we consider three VM types: *m1.small*, *m1.medium*, and *m3.medium*. Table 5.1 summarises their cost and declared capacities in the Sydney AWS region which we use.

Table 5.1: AWS VM type definitions.

VM type	ECU	RAM	Cost per hour
m1.small	1	1.7GB	$0.058
m1.medium	2	3.75GB	$0.117
m3.medium	3	3.75GB	$0.098

In all experiments, we use the same workload. We start by emulating 30 users and each 6 minutes we increase the total number of users with 10 until 400 users are reached. To achieve this we run a sequence of CloudStone benchmarks, each having 1 minute ramp-up and 5 minutes steady state execution time. Given CloudStone's start-up and shut-down times, this amounts to more than 5 hours per experiment. The goal is to gradually increase the number of users, thus causing the system to scale up multiple times.

To test our approach when a workload characteristic changes, we "inject" a change 3.5 hours after each experiment's start. To do so, we manipulate the *utilisation monitors* to report higher values. More specifically they increase the reported CPU utilisations with 10% and the reported RAM utilisation with 1GB plus 2MB for every currently served user.

We implement one experiment, which is initialised with a *m1.small* AS VM and each new VM's type is chosen based on our method (DVTS). We also execute 3 baseline experiments, each of which statically selects the same VM type whenever a new VM is needed, analogously to the standard AWS Auto Scaling rules.

First, we investigate the behaviour of DVTS before the workload change. It continuously trains one HTM for the first AS VM and the ANN. In the initial stages the ANN *learning rate* and *momentum* decrease and increase respectively to facilitate faster training. For example, the *learning rate* lr_k (defined in Eq. 5.9) during the initial stages is depicted in Fig 5.7. It shows how lr_k drastically reduces as the ANN improves its accuracy after only a few tens of training samples. Once the AS VM gets overloaded we select a new VM type. At this point we only have information about *m1.small* in the *capacity repository* and therefore we infer the other CPU capacities based on Eq. 5.4. Finally, using Algorithm 6 we select *m3.medium* as the type for the second VM.

After the new VM is instantiated, the autoscaling component starts its monitoring.

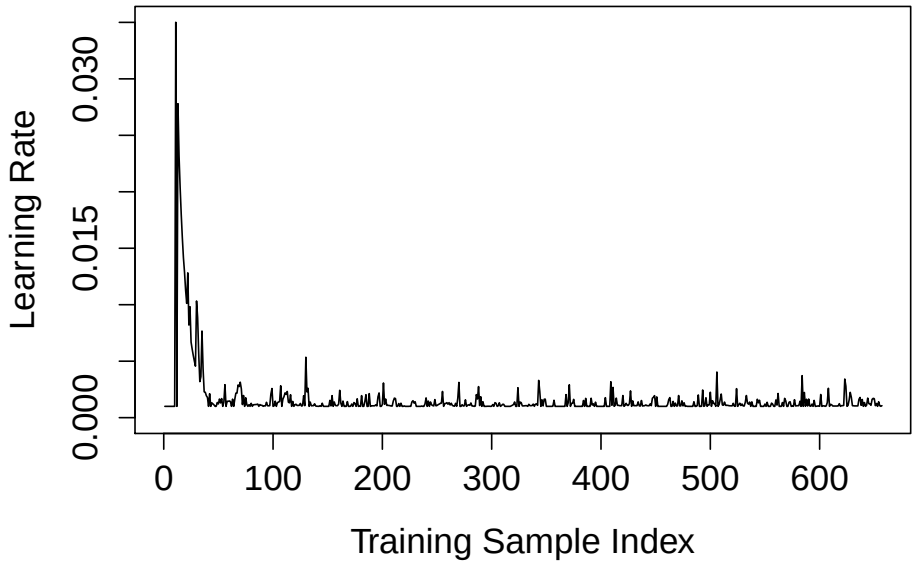


Figure 5.7: Learning rate lr_k during initial stages of training the ANN.

It trains the ANN and a new dedicated HTM with its measurements. It also updates the *capacity repository* with the CPU capacity of the new VM. Surprisingly, we observe that on average its CPU capacity is about 35% better than the one of the *m1.small* VM, even though according to the specification *m3.medium* has 3 ECUs and *m1.small* has 1. Therefore, the previous extrapolation of *m3.medium*'s capacity has been an overestimation. Hence, when a new VM is needed again, the algorithm selects *m1.small* again.

3.5 hours after the start of the experiment the workload change is injected. This is reflected in the HTMs' anomaly scores an_k and the ANN's errors. Consequently, the *learning rate* lr_k, the *momentum* m_k and the *epochs* e_k also change to speed up the learning process as per equations 5.9, 5.10, and 5.11 and as a result the ANN adapts quickly to the workload change. As discussed, for each sample we compute its error (RMSE-pre) before updating the ANN. Figure 5.8 depicts how these errors increase when the change is injected and decrease afterwards as the ANN adapts timely.

Eventually the load increases enough so the system needs to scale up again. Due to the injected change, the workload has become much more memory intensive, which is reflected in the ANN's prediction. Hence, *m1.small* can serve just a few users, given it has only 1.7GB RAM. At that point the CPU capacity of *m1.medium* is inferred from the capacities of *m1.small* and *m3.medium* as per Eq. 5.4, since it has not been used before. Consequently, Algorithm 6 selects *m1.medium* for the 4th VM just before the experiment

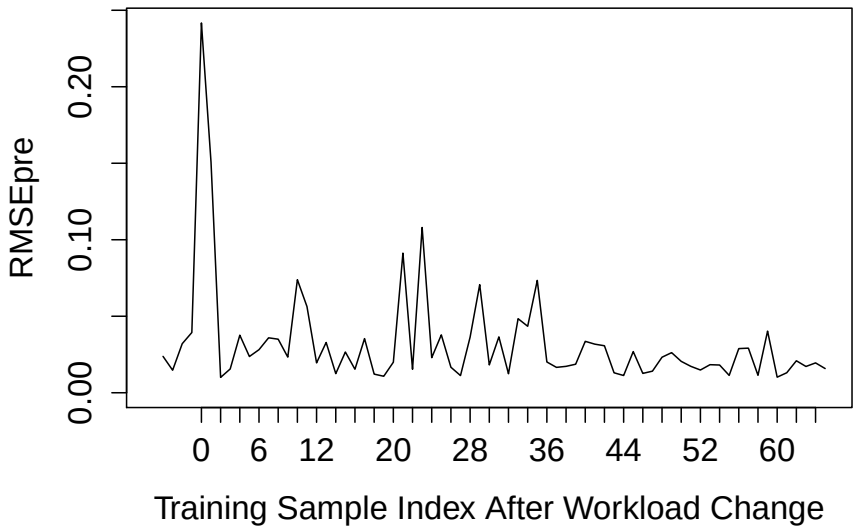


Figure 5.8: RMSE-pre in the presence of a workload change. The 0 index corresponds to the first sample after the workload change.

completes.

For each experiment, Figure 5.9 depicts the timelines of the allocated VMs and the total experiment costs. For each VM the type and cost are specified to the right. Our selection policy is listed as *DVTS*. The baseline policy which statically selects *m1.small* allocates 8 new VMs after the workload change as *m1.small* can serve just a few users under the new workload. In fact, if there was no *cool down* period in the autoscaling, this baseline would have exceeded the AWS limit of allowed number of VM instances before the end of the experiment. The baselines which select *m1.medium* and *m3.medium* fail to make use of *m1.small* instances before the change injection, which offers better performance for money.

Admittedly, in the beginning DVTS did a misstep with the selection of *m3.medium*, because it started with an empty *capacity repository* and had to populate it and infer CPU capacities "on the go". This could have been avoided by prepopulating the *capacity repository* with test or historical data. We could expect that such inaccuracies are avoided at later stages, once more capacity and training data is present. Still, our approach outperformed all baselines in terms of incurred costs with more than 20% even though its effectiveness was hampered by the lack of contextual data in the initial stages.

Our experiments tested DVTS and the baselines with a workload, which is lower than what is observed in some applications. While our tests did not allocate more than 12 VMs

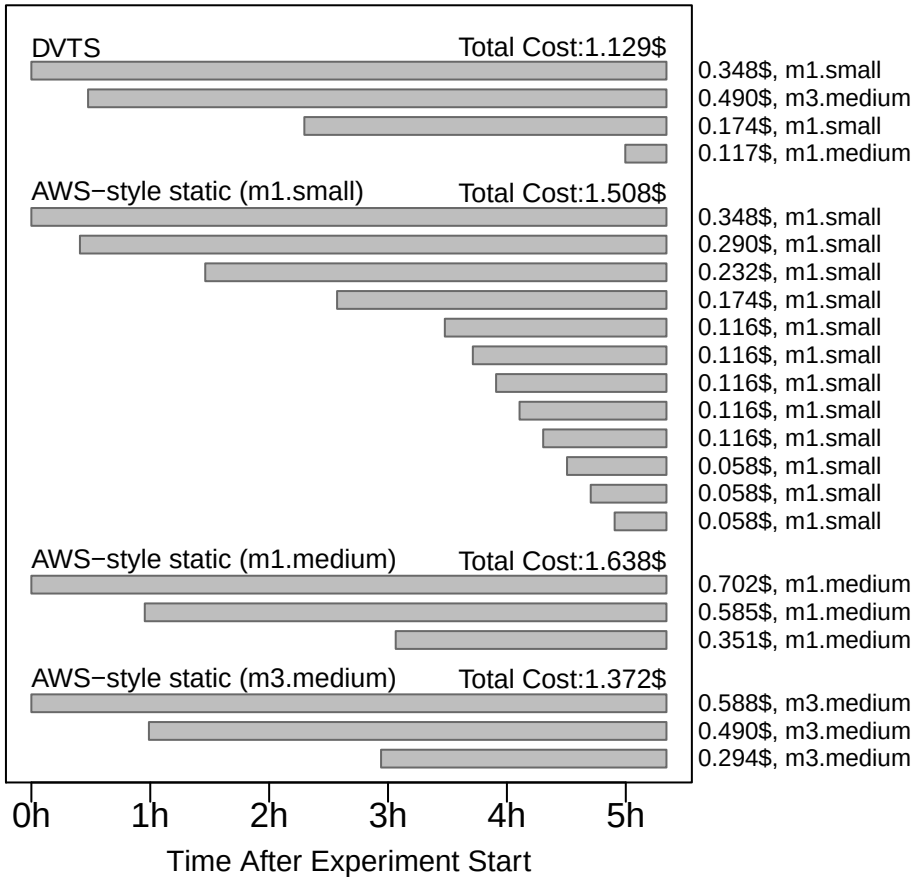


Figure 5.9: Timelines and costs of all VMs grouped by experiments. DVTS is our approach. The AWS-style policies are the baselines, which statically select a predefined VM type.

(in the baseline experiment, which statically allocates *m1.small*) many real world systems allocate hundreds or even thousands of servers. We argue that in such cases, DVTS will perform better than demonstrated, as there will be much more training data and thus the VM types' capacity estimations will be determined more accurately and the machine learning approaches will converge faster. As discussed, this would allow some initial missteps of DVTS to be avoided. Moreover, as the number of AS VMs grows, so does the cost inefficiency caused by the wastage of allocated resources, which can be reduced by DVTS.

Finally, the response times in the DVTS experiment and all baseline experiments were equivalent. All experiments scale up once the AS VMs' utilisations exceed the predefined thresholds, and thus never become overloaded enough to cause response delays. The load balancer is equally utilised in all experiments, as it serves the same number of

users, although it redirects them differently among the AS VMs. Similarly, the DB layer is equally utilised, as it always serves all users from all AS VMs.

5.8 Summary

In this chapter, we have introduced an approach for VM type selection when autoscaling application servers. It uses a combination of heuristics and machine learning approaches to "learn" the application's performance characteristics and to adapt to workload changes in real time. To validate our work, we have developed a prototype, extended the CloudStone benchmark, and executed experiments in AWS EC2. We have made improvements to ensure our machine learning techniques train quickly and are usable in real time. Also, we have introduced heuristics to approximate VM resource capacities and workload resource requirements even if there is no readily usable data, thus making our approach useful given only partial knowledge. Results show that our approach can adapt timely to workload changes and can decrease the cost compared to typical static selection policies.

In the next chapter, we consider the problem of workload redirection across clouds in more details. We introduce a framework that redirects incoming users to cloud sites considering the potential network latencies and the stated regulatory constraints.

Chapter 6

Regulations and Latency Aware Load Distribution

In this chapter, we introduce a flexible framework that allows interactive web applications to utilise a Multi-Cloud environment. It redirects users to suitable cloud sites considering jurisdictional and latency constraints. Regulatory requirements are specified via a flexible and simple domain specific model, which is then interpreted by a rule inference engine. We conducted an experimental evaluation of the proposed system using services of 10 cloud sites/data centres located in 5 continents and offered by two cloud providers, namely Amazon and NeCTAR. The results show that our approach minimises network latency, is fault tolerant, and meets all stated regulatory requirements with negligible performance overhead.

6.1 Introduction

ENTERPRISES that operate across national borders have to consider a plethora of legislative and regulatory constraints with respect to privacy, data access control, security, etc. Government and academic reports have outlined the complexities of building regulatory compliant cloud services and emphasised the need for engineering approaches enabling such compliance [27, 39, 132, 133]. As an example, the Data Protection Directive of the European Union (EU) forbids the transfer of personal data to non-member countries, unless an adequate level of protection is ensured [70]. Similar laws exist in many other jurisdictions as well [39]. Other legislations, like the Regulation of Investigatory Powers Act (RIPA) [128] in the UK, allow governments to access or monitor cloud clients data and operations thus raising privacy concerns for some cloud services.

This chapter is based on the following publication: ***Nikolay Grozev*** *and Rajkumar Buyya, "Regulations and Latency Aware Load Distribution of Web Applications in Multi-Clouds", The Journal of Supercomputing, Springer (Under Review), 2015.*

Businesses operating in multiple legislative domains have to consider many and often conflicting regulatory requirements and thus no single cloud site will suffice.

To address these problems, the usage of multiple cloud sites (i.e. a Multi-Cloud) has gained significant interest. If an application has specific requirements as to where (e.g. in which jurisdiction) a user is served, this user can be redirected accordingly to one of the employed sites. As discussed previously, Multi-Clouds facilitate the development of more responsive web based interactive applications. Multiple geographically distributed cloud sites can serve users worldwide with low network latency and thus provide adequate QoE. Naturally, this raises the question of how these two objectives can be achieved in conjunction, without adding significant overhead to the system.

Chapter 4 already proposed mechanisms for efficient 3-Tier application brokering in multiple clouds, which consider these aspects among others. However, the discussion of these characteristics was mostly conceptual and we evaluated the system performance through discrete event simulations. This chapter is much more technical in nature and demonstrates how these ideas can be implemented with modern distributed computing and rule inference technologies. We develop a prototype, which we use for evaluation and detail the technical aspects of our implementation and optimisation techniques, which allow our approach to scale adequately under heavy load. Also, we explore in details how regulatory requirements can be flexibly specified with minimal efforts in a domain specific model and efficiently interpreted at runtime. Finally, we have extended our network latency estimation approach to achieve better accuracy and scalability.

In this chapter, we propose a system for load distribution of web facing interactive cloud applications. We detail its underlying design and algorithms, describe the technical aspects of our prototype, and experiment with it in real life cloud environments. More specifically, our key **contributions** are: (i) design and implementation of architectural components for regulations and latency aware user redirection in a Multi-Cloud environment and (ii) a domain specific approach for defining regulatory constrains.

The rest of the chapter is organised as follows: In Section 6.2 we provide an overview of the related works and compare them to ours. In Section 6.3 we outline the architectural components and their interaction model. Section 6.4 presents a detailed description of all system components' implementation in terms of used technologies, algorithms, and

configurations. It also proposes a domain specific approach for defining regulatory constrains. Our experimental settings and results are discussed in Section 6.5. In the final Section 6.6, we summarise our findings.

6.2 Related Work

Our approach encompasses load distribution and resource selection in a Mult-Cloud and considers the regulatory requirements. We have already reviewed the related works in these areas in Chapters 2 and 4. In essence, while Multi-Cloud projects like OPTIMIS [73], Contrail [49], mOSAIC [137], and MODAClouds [22] can dynamically select and provision resources in multiple clouds, we also address the problem of workload redirection. Furthermore, they do not consider the geographical locations of the cloud sites they select from, while we do in order to implement regulations aware load distribution. At present, load distribution technologies like AWS Route 53 [8] and AWS Elastic Load Balancer (ELB) [11] only consider the network latency or the utilisation of the underlying infrastructure. We also consider the regulatory requirements and the end user's identity.

Content Delivery Networks (CDN) have some resemblance with Route 53 and our approach because they efficiently deliver web content hosted in multiple data centres to users distributed worldwide. However, a CDN only delivers static web content, while we direct users to cloud sites where they are interactively served with dynamic web content.

Mont et al. proposed an approach allowing end users to specify their own data protection rules [113]. They also introduced a method for service providers to enforce such rules by using sticky policies. Their work was later prototyped by Mowbray et al. [116]. This model is not applicable when a cloud provider does not collaborate to enforce the stated requirements, while our approach is cloud agnostic. Furthermore, our method works in a Multi-Cloud environment, while theirs is applicable only within a single cloud site.

Several formalisms for specifying access policies have been proposed. Two notable examples are OASIS XACML [122] and EPAL [1]. Our approach uses a rule based inference engine to infer the suitability of a cloud site for a user. Therefore, we found it convenient to express the access policies directly in the declarative language of the rule engine. As a future work we are planning to investigate how XACML and EPAL specifi-

cations can be transformed to such engine rules.

To dispatch a user to an appropriate cloud site, we first need to authenticate him/her, which can be done in several ways. The OpenID protocol [125] allows a service provider to authenticate an end user through an external service. The OAuth [123] protocol allows a service provider to gain authorised access to protected user resources hosted on another service without explicitly providing credentials (e.g. user name/password). Alike OpenID, OAuth can be used for authentication as well, if the accessed resources are only user metadata (e.g. user name). More and more web services are adopting OpenID and OAuth. However, many others still rely on custom authentication (e.g. user name/password). Since we aim for generality, we decided not to couple our approach with any of these specifications. Instead, our approach works with user tokens, which are unique ids and can be extracted from OpenID, OAuth or via custom authentication.

6.3 System Architecture

Our approach to cloud site selection (depicted on Figure 6.1) extends the aforementioned industry standard approach by replacing the DNS service with an *Entry Point* component. Similarly, it redirects users among a set of cloud locations. Furthermore, within each target cloud site we deploy an additional *Admission Controller* component. *Entry Points* communicate with the *Admission Controllers* to determine the suitability of the respective cloud sites and then redirect the user accordingly. Apart from these two new components, there are no other changes to the industry standard approach. Therefore, in each cloud the system can be implemented in accordance with the reference 3-Tier approach. In other words, what we propose is a layer "on top" of the standard 3-Tier architecture, which performs the user redirection to the individual cloud sites.

An end user arrives at an *Entry Point*. There may be one or more *Entry Points* whose purpose is to redirect each user to an appropriate cloud site. Before doing so, the user must authenticate so the regulatory requirements can be considered. Authentication is application specific and is provided by the application developers. It could be OAuth or OpenID authentication with an external service, or a custom username/password implementation. Once the authentication is complete, the *Authentication Server* passes a unique

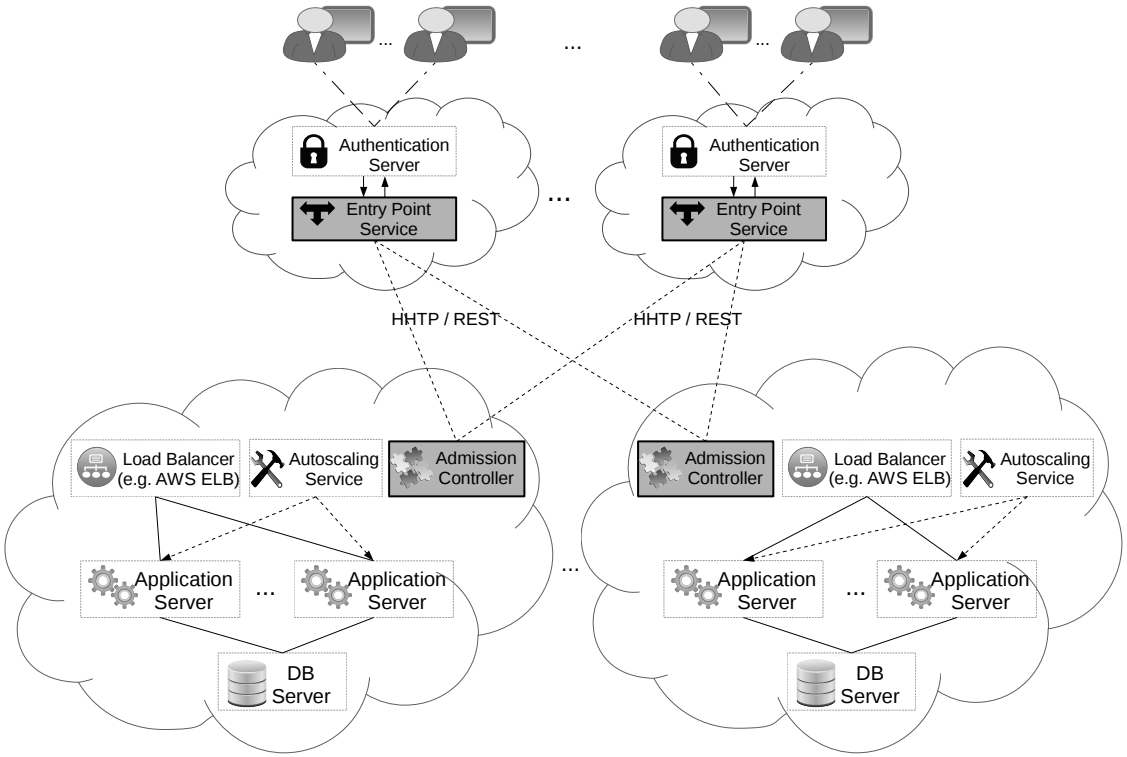


Figure 6.1: Overall Architecture: *Entry Point* and *Admission Controller* components replace the DNS based cloud site selection.

user identifier (e.g. retrieved from an OAuth server or just a user name) to the *Entry Point*.

The *Entry Point* can be called in two ways. Since it is developed in Java, it can be put on the java classpath of the *Authentication Server*. It can also be accessed as a RESTful web service in case the *Authentication Server* is not developed in Java. An *Entry Point* is deployed in a separate VM. Depending on its implementation, the authentication service could be deployed within the same VM or separately.

In each target cloud site there is an *Admission Controller* which exposes a RESTful web service API and is deployed in a standalone web server. The *Entry Point* sends a request to each cloud site's *Admission Controller* to determine which sites are eligible in terms of regulations to serve the incoming users. We have developed a domain specific rule based model (detailed in Section 6.4.2) allowing for succinct regulations requirements specification. Based on this information and estimations of the network latency between the end user and each cloud site, the *Entry Point* decides where to redirect the user to.

Figure 6.2 depicts this flow.

Once the user is redirected to a cloud site, he/she is served there and has no further interaction with the *Entry Points* and the *Admission Controllers*. In other words, the delay resulting from the interactions between the *Entry Point* and the *Admission Controller* is tangible to the user only during the initial redirection process and does not impede the overall QoE. We argue that existing web-facing systems, which use OAuth 2.0 authentication with external services (e.g. Google, Facebook or LinkedIn) already serve users with one-off initial delays, and thus this is acceptable.

Although placing the *Admission Controllers* in all cloud sites contributes to the one-off delay, it allows them to closely monitor the resource performance and utilisation within each cloud site. This information can be fed back to the *Entry Points* to implement sophisticated redirection policies. We are planning to investigate this in our future work and thus we aim for a flexible architecture.

6.4 Design and Implementation

6.4.1 Entry Points

The function of each *Entry Point* is to redirect each incoming user to an appropriate cloud site. After a user is redirected, he/she has no further interaction with the *Entry Point*. Thus, it introduces a one-off delay upon user arrival and does not affect the subsequent user interactions.

Each *Entry Point* is developed in Java and is deployed into a single Java archive (i.e. *jar*) file. It can be used from another Java application by adding the jar file to its classpath. To simplify the integration with the *Authentication Service*, the jar exposes a singleton façade class called *EntryPoint*, which for a given user determines the serving cloud site.

The *Entry Point* can be executed separately as a standalone application as well. In this case, an embedded *jetty* web server is started, which uses the *Jersey* RESTful Web Services framework to expose a REST web service. This is useful when the client code is not easily integrated with Java. Details on how this is achieved can be found in our online

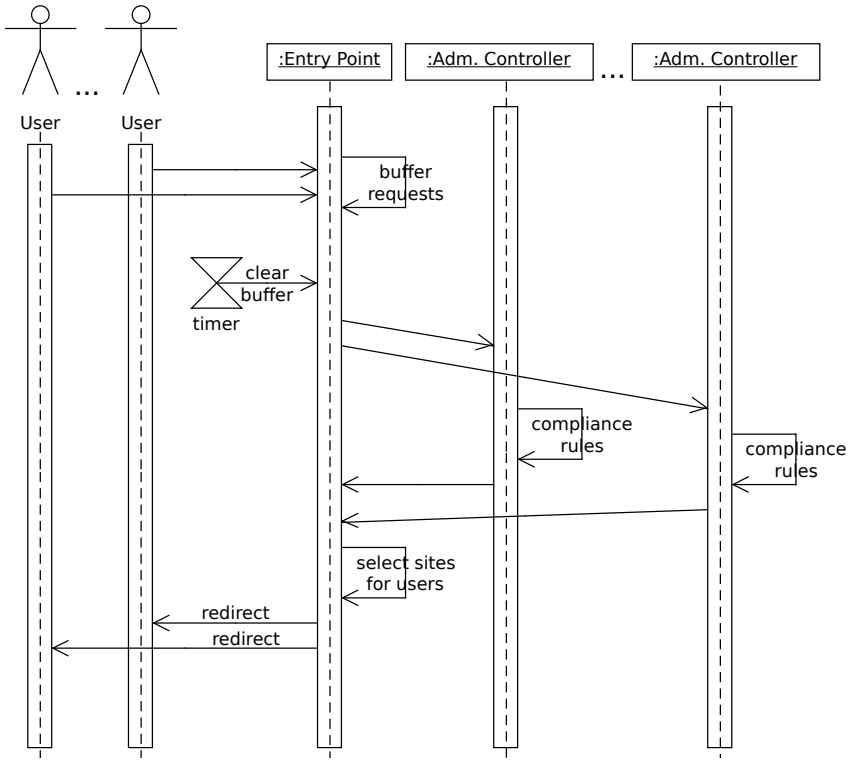


Figure 6.2: *Entry Point — Admission Controller* interaction.

documentation [1]. Client code can call this service by providing the user identification token and IP address and receive as a result the address of the *Load Balancer* in the selected cloud site. Behind the scenes, the RESTful web service simply calls the aforementioned *Entry Point* façade.

Figure 6.2 illustrates the interactions between the *Entry Point*, the users, and the *Admission Controllers* which is discussed hereafter. For brevity it omits the user authentication.

To avoid excessive network communication, the *Entry Point* sends the incoming user requests to the *Admission Controllers* in batch rather than one by one. The *Entry Point* aggregates the user requests over a time period (typically a few seconds) and sends them to the *Admission Controllers* at once. This is transparent from the caller's perspective, as the procedure call is blocked until the result for the respective user request is complete. Furthermore, requests are broadcasted to the *Admission Controllers* asynchronously. A separate thread from a cached thread pool is used for each *Admission Controller* in order

[1] https://nikolaygrozev.wordpress.com/2014/10/16/rest-with-embedded-jetty-and-jersey-in-a-single-jar-step-by-step/

to reduce the overall response time.

We emphasise that the delays during the initial end user redirection are one-off. After the user is redirected to a cloud site, they only communicate with its load balancer. There are no additional delays in the user's interaction with the application. Furthermore, in Section 6.5 we demonstrate experimentally that these initial delays are negligible.

In the case of a cloud site failure, the respective *Admission Controller* will not respond to the *Entry Point* requests. Hence, the *Entry Points* are preconfigured with deadlines and if a cloud site does not respond within the deadline it is not considered. If however, this *Admission Controller* "reappears" online — the *Entry Point* will start considering it again. For this purpose, each *Entry Point* is configured with a time period parameter, so that it can check if failed *Admission Controllers* have reappeared online. Once the responses from the *Admission Controllers* have been received, the *Entry Point* selects the eligible cloud site, which offers the lowest network latency.

We developed a utility which estimates the network latency between an end user and a cloud site. Given two arbitrary IP address (e.g. of the user and the load balancer in a cloud site) it estimates the expected network latency between them. We approach this problem in two steps. Firstly, we use the MaxMind's GeoLite [80] database to determine the location (i.e. longitude and latitude) of each IP address. The latest GeoLite database is freely available and can be downloaded in a proprietary format. MaxMind also provides an open source Java library to read and query the proprietary database, which we use.

MaxMind's GeoLite is updated once every few months and thus it may not resolve some IP addresses' respective locations correctly. This can often happen for IP addresses of cloud VMs, as a cloud provider can dynamically reassign IPs from the same range to machines in different data locations. Experimenting with GeoLite we found that while end users' IP addresses are mostly resolved accurately, the estimated location of AWS EC2 VMs based on their public IP address can be very incorrect. For example, a VM in the Tokyo AWS region can be resolved as located in the US. Fortunately, cloud providers like Amazon AWS provide up to date information about their IP ranges and the respective geographical locations [9]. In case an IP address falls within any of these ranges we can use it to override the respective value of GeoLite.

Secondly, once the coordinates are established, we use PingER which is an Internet

Performance Monitoring Service (IEPM) [138]. It records network characteristics (e.g. round trip time) between hundreds of nodes distributed worldwide which periodically ping each other. The reported performance metrics can be averaged per time interval (e.g. year, month, day or hour) and can be downloaded in tab separated values (TSV) files through a public web service. The coordinates (longitude and latitude) of each node are provided as well. We use Vincenty's formulae to compute the distance between any two geospatial positions. To estimate the latency between two IP addresses, whose positions have already been resolved, we use the network latencies between the 3 pairs of PingER nodes that are geometrically closest to them.

More formally, for each pair of PingER nodes (n_{i1}, n_{i2}) we define its distance to the locations of the target pair of IP addresses (t_1, t_2) as:

$$distance(n_{i1}, n_{i2}, t_1, t_2) \quad = \quad \min\{v(n_{i1}, t_1) + v(n_{i2}, t_2), v(n_{i1}, t_2) + v(n_{i2}, t_1)\} \quad (6.1)$$

where v is the well known Vincenty's function for computing the distance between two geographical locations specified by their coordinates. Then we choose the 3 pairs of nodes $(n_{11}, n_{12}), (n_{21}, n_{22}), (n_{31}, n_{32})$, which minimise the *distance* function for the targeted t_1 and t_2. Hence, we can approximate the Internet latency between t_1 and t_2 as the following weighted sum:

$$latency(t_1, t_2) = (\sum \frac{d_{min}.l_i}{d_i}) / (\sum \frac{d_{min}}{d_i}) \quad (6.2)$$

where d_i is a shorthand for $distance(n_{i1}, n_{i2}, t_1, t_2)$, d_{min} is the smallest distance among d_1, d_2, d_3 and l_i is the PingER latency between n_{i1} and n_{i2}. Eq. 6.2 essentially defines a weighted sum of the network latencies between the 3 geometrically closest pairs of nodes. The weights are defined proportionally to each pair's nodes combined distance to the target locations.

To improve the performance of the network latency estimation we keep in-memory *least-recently-used* (LRU) caches of (i) the mappings between IP addresses and coordinates, (ii) the distances between geospatial locations, and (iii) the already resolved latencies. To improve the cache-hit probability, we round all coordinates (latitude and latency) to 2 digits after the decimal sign when comparing with or inserting into the caches. This ensures the network latency estimation for users coming from the same region (e.g. the

same city) will not be recomputed every time. We use the Google Guava [82] cache implementation as it is thread safe, and limit all cache sizes to 10 000 entries.

Finally, in the previous discussion we mentioned several configuration parameters used by the *Entry Point*. These are provided in two configuration files. The first one is a coma separated values (CSV) file which lists all cloud sites and the addresses of their *Admission Controllers* and *Load Balancers*, from which the load balancer chooses when a user arrives. The second file is a property file, and provides (i) the time between the batch requests to the *Admission Controllers* and (ii) the deadline length to wait for response from the *Admission Controllers*. These files must be provided before the *Entry Point Service* is used, but can also be updated and reloaded dynamically to change its behaviour — e.g. to add a new cloud site.

6.4.2 Admission Controllers

Each *Admission Controller* is deployed in a single jar file and is executed within an embedded jetty server hosting a RESTful web service. The service takes as input a list of user identifiers, and returns the responses as a list of JSON formatted objects. The *Entry Points* communicate with the *Admission Controllers* through this web service, even if they are deployed in the same VM.

The *Admission Controller* must determine the eligibility of the cloud site for each of the users. To achieve this, first we must resolve the user's metadata — e.g. data about their citizenships and privacy requirements. Obviously this resolution is application specific. For example, one application would use OAuth to retrieve such data, while another would just execute a query within its database. Since we aim for generality, our *Admission Controller* component provides a "hook" for resolving user metadata. We define a simple interface called *IUserResolver*, which has a single method for resolving a user's metadata. Application developers should implement this interface and provide the name of their class as an input parameter to the *Admission controller*.

We also define two simple classes representing a user and a cloud site. The *User* class has just 3 fields: (i) user id, (ii) set of citizenships, and (iii) tags. The *tags* field can aggregate miscellaneous additional meta-information for the user — e.g. if he/she is a government employee. *Admission Controllers* uses the previously discussed user resolver to

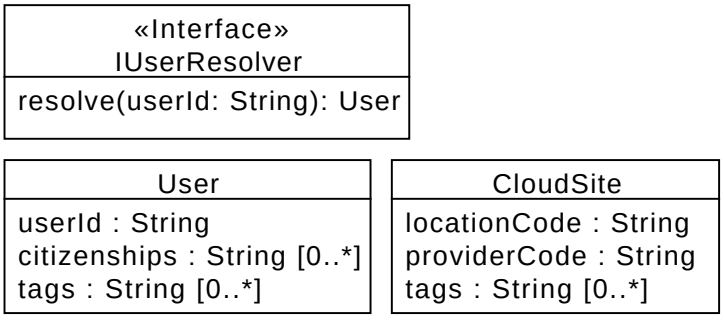


Figure 6.3: Domain model classes used for cloud site to user matching.

instantiate this class based on a user id. The *CloudSite* class represents meta-information about the present cloud site. It has fields representing: (i) the geographical location (e.g. country or state code) (ii) the cloud provider (e.g. AWS), and (iii) a set of tags. As before, we use tags to represent miscellaneous additional information — e.g. if the cloud is certified to serve US government agencies. A JSON file containing the respective marshalled *CloudSite* instance is given as a parameter to every *Admission Controller*.

The *User*, *IUserResolver*, and *CloudSite* types (depicted on Figure 6.3) define the simple interface, which we expose to application programmers.

Given a user and a cloud site, the *Admission Controller* must infer if the user can be served there. An example might help picture what the reasoning logic should be. Let us consider an application which needs to ensure that EU citizens are served within the EU, and that government officials must be served in specifically certified cloud sites. Given a user id, we can resolve the user's meta-information with the provided *IUserResolver*. Let us assume this returns a user with French citizenship, who is also a government official (denoted by a tag in the *User* instances). Let us also assume the corresponding cloud site (represented with a *CloudSite* instance) is located in Germany and the provider is AWS. We may also know that all AWS cloud sites comply with the government certification in question. In this case we should be able to infer that the cloud site is eligible to serve the client.

This type of automated reasoning based on predefined facts and rules is a hallmark application of rule based engines and this is exactly how we approach this problem. In our implementation of the *Admission Controller* we use the Drools rules management sys-

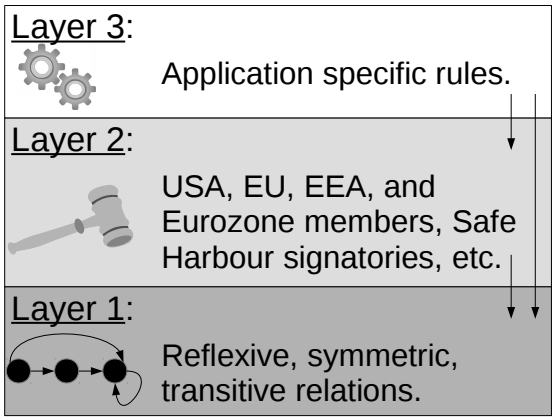


Figure 6.4: Layers of DRL rules. Each layer can use the rules from the underneath layers.

tem [143]. Drools uses the ReteOO algorithm, which implements well known performance enhancements of the famous Rete algorithm. Drools uses a domain specific language called Drools Rule Language (DRL) to define rules and facts. It integrates well with existing Java code and facts in DRL are just Java objects. This allows the declarative programming approach prescribed by DRL to also make use of the polymorphic features of Java. For example, if a rule is activated for any fact of a specific Java type, it will also be activated for any instance of its subtypes. We will make use of this feature to allow more flexible specification of regulatory rules.

Specifying compliance rules in a declarative DRL form rather than procedurally is beneficial in terms of maintainability and flexibility. However, this still needs to be done on a per-application basis. Many applications would share common rules and facts — e.g. about which countries participate in the U.S.-EU Safe Harbour agreement. It would be beneficial if we specify such common rules so they can be reused by applications. Hence, we divide the rules in 3 layers/modules depicted in Figure 6.4. The rules from each layer can only use the ones from the layers below.

In Layer 1, we model the concepts of reflexive, symmetric, and transitive binary relations. We introduce 3 Java interfaces to represent these concepts. They all extend from a common *IRelation* interface which has a left and a right hand side (*lhs* and *rhs*) objects — see Figure 6.5. Layer 1 includes DRL rules, which produce new *IRelation* instances in the Drools working memory based on the reflexive, symmetric and transitive algebraic rules. For example, if we have an instance of *ISymmetricRelation*, whose $lhs = a$ and $rhs = b$, the

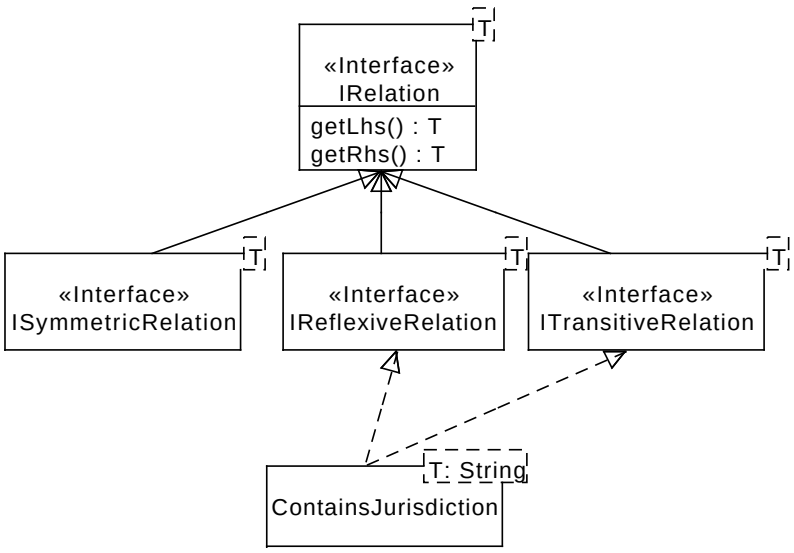


Figure 6.5: Interfaces for the relations from Layer 1 and the *ContainsJurisdiction* relation from Layer 2.

respective rule in Layer 1 will be activated and will insert in working memory a relation of the same type, whose $lhs = b$ and $rhs = a$.

To illustrate how this is useful, let us consider how we can implement a relation of jurisdictional containment. We can define a relation *ContainsJurisdiction*, which implements *IReflexiveRelation* and *ITransitiveRelation*. Then we can use it to specify the facts that Germany is within the Eurozone, which in turn is in the EU. The rules in Layer 1 will automatically fire and populate the Drools working memory with the appropriate *ContainsJurisdiction* instances denoting that Germany is in the EU and itself. This simplifies significantly the development of rules using jurisdictions. Figure 6.5 depicts the aforementioned interfaces and class.

Layer 2 specifies common facts and rules about the regulatory domain and cloud providers. In fact, the aforementioned *ContainsJurisdiction* relation is defined and extensively used in Layer 2, to specify which countries are in the Eurozone, EU and The European Economic Area (EEA), which countries have signed the US-EU Safe Harbour agreement and so on. In essence, this layer contains the expert knowledge about the regulatory domain in the form of DRL rules.

Finally, using the domain rules from Layer 2 we can implement the actual admission

control policies in Layer 3, which are application specific. We define a class *Admission-Denied*, which has a single property — the unique user id. When the *Admission Controller* starts the DRL rules, it inserts into the working memory the meta-information of one or more users and the cloud site as instances of the aforementioned types. Based on this, the rules from Layer 3, should insert an *AdmissionDenied* instance in working memory for every user who is not eligible for the specified cloud site.

```
1 rule "US government officials - in US only"
2 when
3     User($id: userId, tags contains "US-GOV")
4     CloudSite($lc : locationCode)
5     not(ContainsJurisdiction(lhs == "USA", rhs == $lc))
6 then
7     insert (new AdmissionDenied($id));
8 end
```

Listing 6.1: Sample Layer 3 admission control rule.

As an example, Listing 6.1 demonstrates how we can restrict the redirection of US government officials to cloud sites outside of the US. Government officials are recognised by a tag in the respective *User* instance.

In Drools, one can specify the *salience* of each rule. This is an integer used by the Drools engine when deciding the order of rule execution. The rules in Layer 1 are given highest salience (200) and the rules from Layer 2 are given lower salience (100). The rules in Layer 3 get the lowest salience of 0. This causes the rules from the lower layers to execute with higher priority (i.e. earlier if possible). Thus, when the higher layer rules are run the underlying knowledge base is already populated.

The logical separation of rules into layers helps in the change management as well. The base rules from Layer 1 should not change for any applications. Rules from Layer 2 should rarely change — for example, when a new country joins the EU, or when new regulation is enforced. The rules from Layer 3 may change frequently and are application specific. Drools allows developers to specify rules in separate files. We use this feature to separate the layers in 3 different files. Application developers need to provide only the file for Layer 3.

6.5 Performance Evaluation

To evaluate our approach we perform two experiments in a heterogeneous Multi-Cloud environment. We employ the following cloud sites:

- AWS Region in Oregon, US;
- AWS Region in Northern California, US;
- AWS Region in Northern Virginia, US;
- AWS Region in São Paulo, Brasil;
- AWS Region in Dublin, Ireland;
- AWS Region in Frankfurt, Germany;
- AWS Region in Tokyo, Japan;
- AWS Region in Singapore;
- NeCTAR data centre in Perth, Australia;
- NeCTAR data centre in Melbourne, Australia.

In both experiments we deploy *Admission Controllers* and *Entry Points* in multiple cloud sites around the world. Then we use other geographically distributed cloud sites to emulate incoming users. We record multiple performance characteristics to demonstrate the viability of our approach under strenuous load.

6.5.1 Experiment 1 — Large Scale Deployment

The goal of our first experiment is to show that our workload distribution system (i) adds negligible delay when users access the system, (ii) honours all regulatory requirements, and (iii) can handle significant workloads without requiring a lot of resources.

In each of 4 different cloud sites we deploy an *Entry Point* and an *Admission Controller* in a shared VM. We use *m3.medium* instances in the N. Virginia, Frankfurt, and Tokyo AWS regions and a *m2.medium* instance in the Melbourne data centre of the NeCTAR Research Cloud. To emulate users from across the globe we use *m3.medium* instances

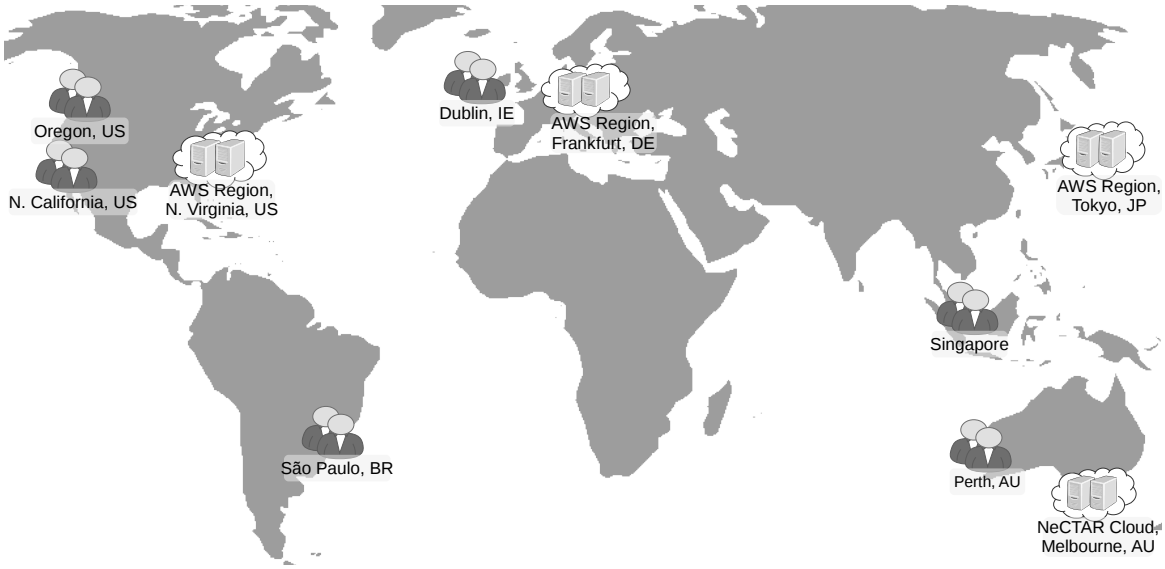


Figure 6.6: Experiment 1 — 4 cloud sites host the *Entry Points* and *Admission Controllers*; 6 cloud sites are used to emulate incoming users.

provisioned in the Oregon, N. California, São Paulo, Dublin, and Singapore AWS regions and a *m2.medium* instance in the Western Australian site of NeCTAR near Perth. Figure 6.6 depicts the locations we use for Experiment 1. All VMs run Ubuntu 14.04 and we have increased their maximal open files and socket connection limits, so they can handle more connections.

The first experiment has been conducted for the duration of 24 hours. We emulate 2 users per second from each of the 6 cloud sites we use to represent clients. This amounts to more than 1 million emulated users during the experiment altogether. Each emulated user connects randomly to one of the 4 *Entry Points* and provides a random user id. In the *Entry Points*, we need to resolve this id to a *User* instance. We do so by implementing a custom *IUserResolver* instance. Given a user id it creates a *User* instance by randomly assigning his/her citizenship to one of the following countries: Germany, USA, Australia, and Canada. To every third user with US citizenship, we assign the tag "US-GOV" denoting that they are government officials. To every second user we assign the "PCI-DSS" tag. The Payment Card Industry Data Security Standard (PCI DSS) formalises procedures for reducing credit card frauds [131]. Tagging users with "PCI-DSS" means they are allowed to work with credit cards data within the system.

We configure each *Admission Controller* with the geographical location and the

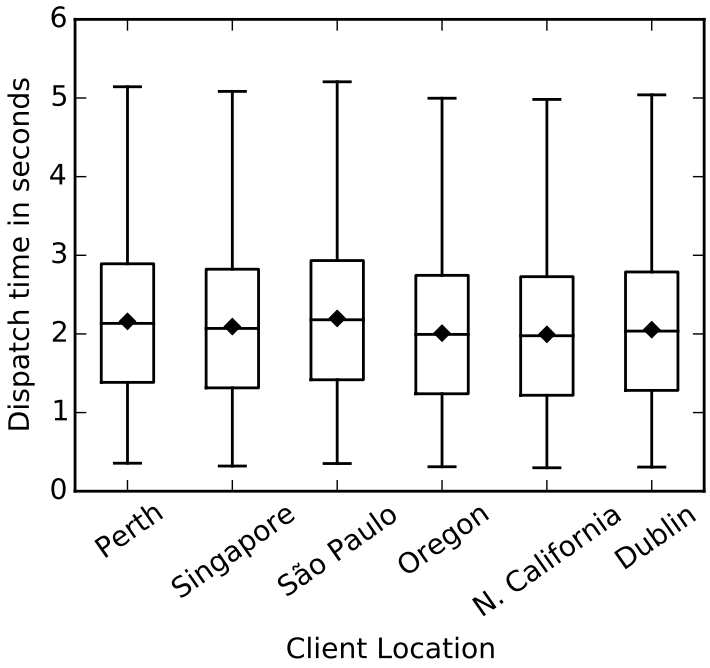


Figure 6.7: Dispatch times of users to their serving cloud sites, grouped by location of the emulated users. Mean values are denoted with a rhombus.

provider name of the respective cloud site. All AWS cloud sites are marked with the "PCI-DSS" tag, as AWS is PCI DSS compliant [10]. The NeCTAR cloud sites are not tagged as "PCI-DSS", as it is lacking official compliance.

We configure the *Admission Controllers* with the following regulatory requirements expressed as Layer 3 DRL rules:

- Users having access to credit card data (i.e. tagged with "PCI-DSS") should be served in PCI DSS compliant cloud sites.
- EU citizens should be served within the EU.
- US government officials should be served within the US.

Figure 6.7 shows the distributions of dispatch times of the emulated end users grouped by their location. By "dispatch time" we denote the time from the user emitting a request to an *Entry Point* until he/she is redirected to an appropriate cloud site — i.e. the duration of the procedure depicted in Figure 6.2. This delay is experienced just once, before the user is redirected to the destination cloud site. Regardless of the user

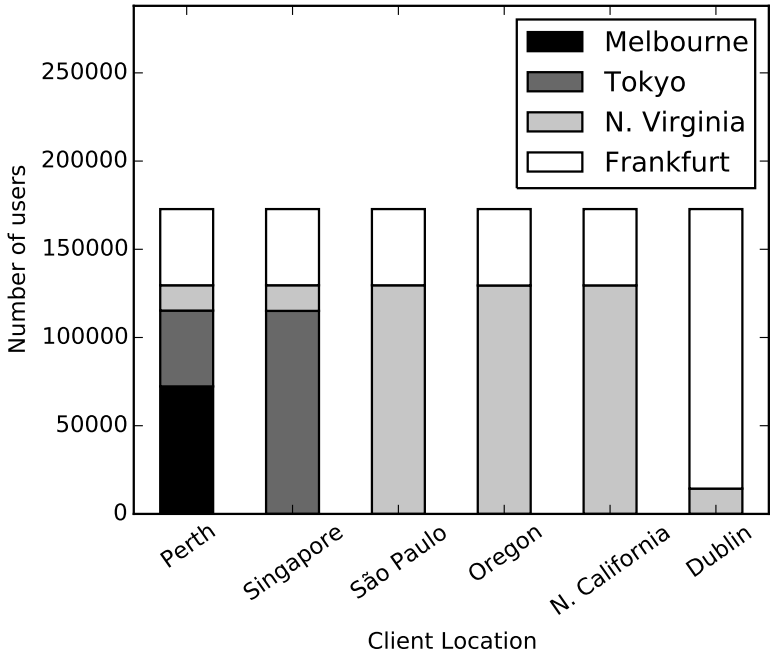


Figure 6.8: Number of users redirected to each cloud site grouped by location.

Table 6.1: Network latencies between clients and cloud sites in milliseconds.

	N.Virginia	Frankfurt	Melbourne	Tokyo
Perth	137.0	180.0	**20.0**	124.0
Singapore	121.0	131.0	69.0	**44.0**
São Paulo	**61.0**	106.0	224.0	140.0
Oregon	**38.0**	73.0	88.0	45.0
N.Cal.	**41.0**	84.0	84.0	52.0
Dublin	39.0	**11.0**	159.0	108.0

location the mean and median dispatch times are around 2 seconds and the majority of values are less than 3 seconds. The mean and median for the users from São Paulo are slightly higher (with approximately 0.18 seconds) because of the higher latencies to all other cloud sites.

Table 6.1 lists the actual network latencies between the emulated clients and the cloud sites. The best latencies are highlighted. These values are obtained by performing 50 measurements between each pair and then taking the median. Ideally, for a given user our approach should prefer the cloud site with the lowest latency unless there are specific regulatory requirements for him/her.

Figure 6.8 shows the proportion of clients from each location dispatched to the different cloud sites. As per Table 6.1, the cloud site in North Virginia offers the best latency

to the clients in São Paulo, Oregon and California, and thus 75% of the users from these locations are redirected there. The other 25% are redirected to Frankfurt in accordance with the regulatory requirements, as they are German citizens. Most users from Dublin are redirected to the Frankfurt site, except for those (approx. 8.3%) who are tagged with "US-GOV" and must be served in the US. Likewise, most users from Singapore were redirected to Tokyo, except for EU citizens and US government officials.

Similarly, 25% of the users from Perth were directed to Frankfurt, and 8.3% to N. Virginia in order to meet the aforementioned requirements. However, only half of the rest were directed to the Melbourne site, although it offers the lowest latency. This is because the NeCTAR cloud is not PCI DSS compliant and thus users working with credit cards have to be served elsewhere. Such users are directed to the Tokyo AWS cloud site, as it offers the lowest latency from all PCI DSS compliant sites.

6.5.2 Experiment 2 — Fault Tolerance

The goal of our second experiment is to show how the system behaves when a cloud site fails. In this experiment we deploy a single *Entry Point* in a separate *m3.medium* VM in the São Paulo AWS site. Also, we deploy two *Admission Controllers* in *m3.medium* instances in the N. Virginia and Frankfurt sites. Alike the previous experiment we emulate users from VMs in the Dublin and Oregon data centres. The frequencies with which users are started is the same as in Experiment 1. However, in this experiment we do not configure any regulatory rules — our goal is to test failure tolerance. The length of Experiment 2 is 3 hours.

To simulate failure, every 20 minutes we switch off one of the *Admission Controllers* for 10 minutes, after which we start it again. We switch off the two *Admission Controllers* in turns. Figure 6.9 shows the proportion of the overall clients directed to each cloud site during the 10 minute periods. As per Table 6.1 the Frankfurt site offers better latency for users in Dublin and the N.Virginia site for those in Oregon. Hence, during the 10 minute periods when both *Admission Controllers* were present, the two cloud sites receive about equal number of users, as we emulate equal number of users from Oregon and Dublin. During the periods when one of the *Admission Controllers* is not present, all users are directed to the other one.

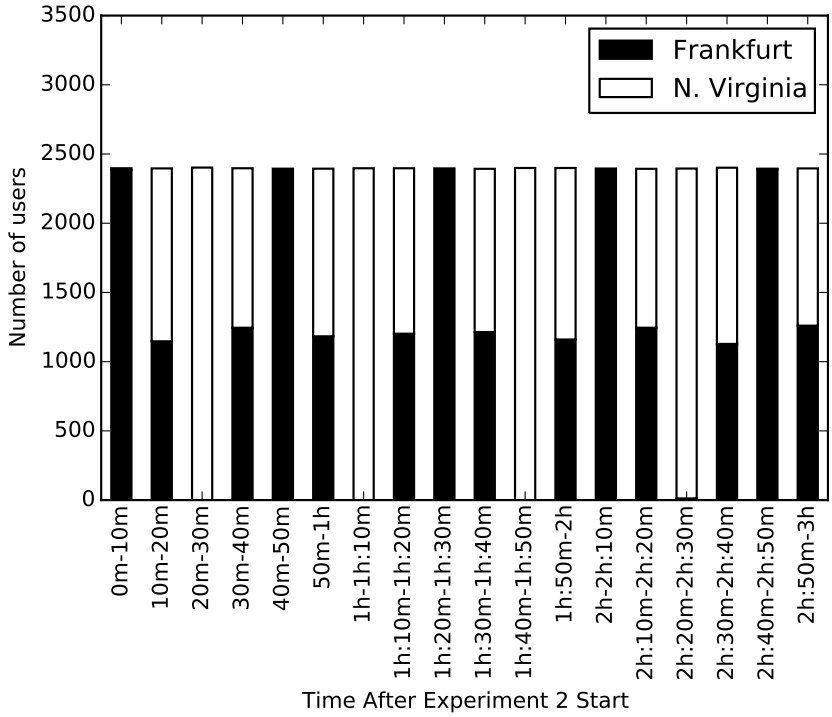


Figure 6.9: Experiment 2 — Number of users directed to each cloud site.

From Figure 6.9 we can see that during the 10 minute periods after an *Admission Controller* is started again it can receive slightly less than the expected 50% of users. In our experiments we have configured the *Entry Points* to check if disconnected *Admission Controllers* have come back online every 1 minute — this is a configurable parameter. Hence, the *Entry Point* can be oblivious that an *Admission Controller* is back online for up to 1 minute, causing all incoming users to be redirected to the alternative cloud site.

Figure 6.10 depicts the dispatch time during each 30 minutes of Experiment 2. It shows that during the experiment the mean and median dispatch times remain below 3 seconds. We have configured the *Entry Point* to connect to the *Admission Controllers* with a timeout of 10 seconds. This parameter is configurable as well. If the connection fails, the respective cloud site is marked as failed and is not used again until it reappears back online, as explained previously. Hence, users arriving at the *Entry Point* immediately after an *Admission Controller* has been shut down can experience a dispatch delay of up to 14 seconds, which is nearly 10 seconds more than the 3rd quartile. This is represented by the outliers in Figure 6.10.

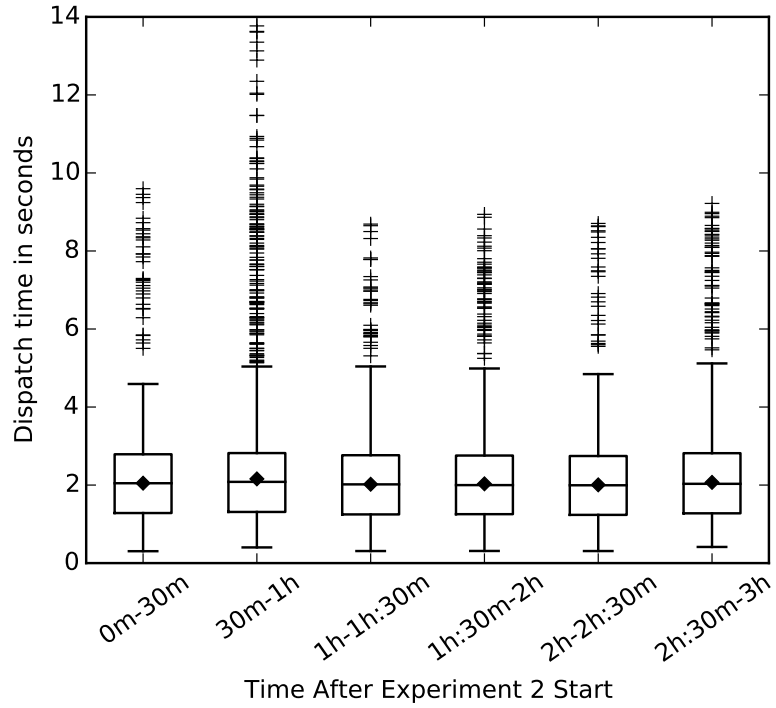


Figure 6.10: Experiment 2 — Dispatch times over time.

6.6 Summary

In this chapter, we have introduced an approach for distributing end users of an interactive web application across multiple cloud sites. We described a rule based domain specific model, which allows regulatory requirements to be specified with minimal efforts. Our system ensures that these requirements are honoured. We provide extension points or "hooks" that allow developers to use their custom authentication and user details resolution mechanisms. Furthermore, our approach estimates the potential network latencies to each cloud sites and chooses accordingly. We deployed our prototype in multiple cloud sites worldwide and carried extensive experiments showing that indeed it is fault tolerant, has negligible performance overhead, minimises latency, and meets the stated regulatory requirements.

Chapter 7
Conclusions and Future Directions

This chapter summarises the research work on Multi-Cloud 3-Tier application brokering presented in this thesis. It highlights the main contributions and outlines a number of promising avenues for future work in the area.

7.1 Summary and Discussion

CLOUD computing is a disruptive model of renting and using IT resources on demand. The advent of cloud services, which are widely accessible on demand through the Internet, has brought us closer than ever to the long held dream of computing as a utility. Renting a dynamic pool of preconfigured IT resources as a service, instead of maintaining dedicated in-house infrastructure, proves to be attractive for many organisations. It allows them to focus on their core businesses and to respond to dynamic and unpredictable workloads.

Using services hosted in a single cloud site poses several challenges like service unavailability, regulatory compliance, network latency between end users and cloud sites, and vendor lock-in, to name a few. These issues are of special importance for large scale web-facing applications serving users worldwide. The 3-Tier architectural pattern is the predominant design approach for building such applications. The use of a Multi-Cloud is seen as a viable approach for resolving the aforementioned issues. This thesis aims to fill the void in the related literature in this field by proposing and investigating novel resource provisioning and workload management techniques (a.k.a. brokering) for 3-Tier applications in a Multi-Cloud. In particular, Chapter 1 described the thesis objectives in more details and outlined how they are addressed in the individual chapters. It also presented a synopsis of the research area, the thesis topic, and motivation.

To position the thesis contributions with respect to the existing body of knowledge, Chapter 2 surveyed the related work in the area of application brokering across clouds. It reviewed, compared, and classified more than 20 academic and industry projects. We identified that most projects do not allow location aware resource provisioning, and thus can not guarantee that the application regulatory requirements will be met. While most projects provide resource provisioning capabilities, none of them manages or balances the incoming workload. The approaches proposed in this thesis encompassed both resource provisioning and workload distribution in conjunction.

Chapter 2 also dealt with the terminological ambiguity in the area, as some terms in this emerging research field are used interchangeably in parts of the literature. In this chapter, we clearly defined and delineated these terms to set the stage for the following chapters. For example, the distinction between Inter-Cloud federations and Multi-Clouds is pivotal for positioning our work in the related literature. An Inter-Cloud federation is achieved when a number of cloud providers voluntarily interconnect their cloud sites to share resources among each other. Federations require that cloud providers cooperate, which is often not the case as they compete with each other, and thus very few industry developments exist. The term Multi-Cloud denotes the usage of multiple clouds by clients or services acting on their behalf without relying on any underlying cooperation between the cloud providers. Presently, this is a much more realistic scenario and consequently there are a number of readily available Multi-Cloud offerings. However, current Multi-Cloud services and libraries do not even provide transparent resource provisioning — they only define unified interfaces for managing resources in multiple clouds. Nevertheless, presently they are the most mature and widely adopted Multi-Cloud technologies and are instrumental in the development of flexible and efficient Multi-Cloud systems.

To facilitate the performance analysis and the development of new brokering techniques, Chapter 3 proposed a performance model for 3-Tier applications in one and multiple clouds. It also introduced a probabilistic approach for modelling time-varying incoming workloads. Based on these models, we have also developed a discrete event performance simulator, which can be used to evaluate systems' behaviour without conducting long running experiments on expensive infrastructure. To evaluate our work, we

modelled the RUBiS benchmarking application and executed simulations with it in one and multiple clouds. Then, we performed actual runs of RUBiS using the same workload and deployment environment as the ones from the simulations and compared the results. We showed that our model approximates well the observed performance when no major resource contention is observed.

Chapter 4 proposed an overall architecture for Multi-Cloud resource provisioning and load distribution for 3-Tier applications. It is generic and can accommodate different policies. In this chapter, we also described provisioning and workload redirection policies, which in combination heuristically optimise the cost and response delays without violating the predefined legislative and regulatory requirements. For this purpose, we developed a method for approximating the potential network latency between any two public IP addresses by using publicly available geolocation and Internet performance monitoring data sets and services. We evaluated our architecture and policies with the simulation environment proposed in Chapter 3. We showed that our approach meets all regulatory requirements, and achieves better fault tolerance and cost efficiency with minimal penalty in terms of network latency compared to the industry best practices.

Chapter 5 presented a policy for VM selection, which can be "plugged" within the aforementioned architecture. We proposed a heuristic approach to dynamically estimate the actual CPU and memory capacities of all available VM types based on real time performance measurements. Our method uses these estimations and a combination of machine learning techniques to dynamically select the appropriate VM type when the domain layer needs to scale up. It works in real time, detects and adapts to performance changes. We performed experiments with the CloudStone benchmarking application in AWS comparing our approach to the industry standard as a baseline. Our method observed more than 20% improvement in the incurred cost than the baseline while the response times of the two remained similar.

Chapter 6 proposed a framework for user redirection across clouds and is also "pluggable" within the aforementioned architecture. It introduced a domain specific rule based model for flexibly defining regulatory requirements. These rules are thereafter interpreted by a rule inference engine at runtime. We also implemented the framework and detailed the architectural design and how different web and concurrent programming

technologies are used. Lastly, we extended our previous network latency estimation utility for the case when the same public IP address has been used in multiple cloud sites at different times.

To evaluate our approach, we deployed our system in several cloud sites in different legislative domains and belonging to 2 cloud providers and performed extensive experiments by emulating incoming users with various characteristics. Our results showed that the framework offers failure transparency to the end user, always honours the stated regulatory requirements, and chooses the optimal cloud site locations for users in terms of network latency.

7.2 Future Directions

This thesis encompassed all aspects of Multi-Cloud 3-Tier application brokering — from workload distribution across clouds to efficient resource management in a single cloud. Nevertheless, in each of the considered subfields there are promising unexplored pathways for future work.

7.2.1 Models for Asynchronous I/O Bound Applications

Chapter 3 introduced a performance model allowing the representation of I/O operations alongside CPU instructions. We also extended the CloudSim simulator accordingly. In this thesis, we used this basic functionality only to model synchronous access to the data layer of a 3-Tier system. However, the presented performance model and simulator are not coupled with the 3-Tier architectural pattern. They can be used as building blocks for simulators representing asynchronous batch data processing applications, which can not be modelled with current simulation environments. Such simulators can foster the research in workload scheduling and resource provisioning of I/O bound applications, just as current discrete event performance simulators have furthered the work on computationally intensive applications.

7.2.2 Provisioning Techniques Using A Mixture of VM Pricing Models

Cloud providers offer resources like Virtual Machines (VMs) via different purchasing models. For example, clients can buy *on demand* VMs and pay for them on a fixed rate, or they can purchase *reserved* instances for long term use on discounted prices. Using reserved VMs can be beneficial when clients know the expected workloads well in advance. Since this is rarely true for dynamic web-facing 3-Tier applications, for generality in this thesis we considered *on demand* VMs only. Nevertheless, if clients have an approximate application specific knowledge of the expected incoming workload, they could allocate a pool of *reserved* instances and compensate any additional demand with *on demand* VMs. This can result in significant savings for cloud clients. The idea could be further extended by developing methods for approximate estimation of the future demand and size of the pool of reserved resources based on historical data.

Another interesting approach would be to use *spot* instances to reduce the cost even further. *Spot* VMs are usually much cheaper than *reserved* and *on demand* instances, but can be forcefully terminated by the cloud provider at any time. Therefore, interactive applications using *spot* instances must be fault tolerant and completely stateless, so that nothing is lost if VMs are forcefully stopped. Furthermore, losing spot VMs can dramatically reduce the overall computational power dedicated to the application, thus deteriorating the response times. New monitoring and autoscaling approaches that can timely and transparently replace terminated spot VMs with *on demand* instances are needed to ensure adequate end user experience at all times. New algorithms using a mixture of *reserved*, *on demand*, and *spot* instances and dynamically deciding on the spot price to bid hold the potential to significantly reduce cloud clients' costs.

7.2.3 Dynamic Replacement of Application Server VMs

In Chapter 5, we discussed how to select a suitable VM when the domain layer needs more resources. It detects changes in the incurred performance utilisation patterns and the capacities of the underlying resources in real time and adapts the VM type selection accordingly. The efficiency of the approach can be increased by periodically replacing all running VMs with more suitable ones in terms of cost and performance. However,

replacing all VMs and migrating their state may be an expensive operation. Most cloud providers charge for VMs per a predefined time unit (e.g. an hour). Switching off a VM before its charge period has expired would not be beneficial. Furthermore, VMs have different expected "*boot times*", which depend on the VM type, operating system, and image size among other factors. This should also be accounted for when planning the VM replacement. New methods for deciding when it is beneficial to migrate, considering all the above factors, are needed.

7.2.4 VM Type Selection In Private Clouds

The goal of the VM selection method proposed in Chapter 5 was to reduce the cost for serving end users. This is a typical objective when using public cloud offerings. However, in private or community clouds, where clients do not pay for the used resources, other objective like energy efficiency or resource waste minimisation are more suitable. Hence, combining our algorithms and machine learning techniques with energy consumption and performance models is viable future work.

7.2.5 Regulatory Requirements Specification Using Industry Standards

In Chapter 6, we proposed an approach for specifying regulatory requirements in the Drools Rule Language (DRL). This was useful, as DRL rules can be directly interpreted by the Drools rule inference engine. However, standards like OASIS XACM and EPAL have gained significant attention, as they are specifically tailored for security and access policies. Investigating how to automatically translate security policies expressed in such standards to DRL rules is essential.

7.2.6 Generalisation to Multi-Tier Applications

This thesis focuses on interactive applications following the 3-Tier architectural pattern. This approach is indeed the prevalent way to build interactive systems. Nevertheless, several architectural approaches have extended the standard 3-Tier model by further splitting the domain layer in multiple parts. When these sublayers are deployed separately, the resulting system becomes Multi-Tier. Investigating how the brokering meth-

ods from this thesis can be generalised to encompass interactive Multi-Tier applications is an important and promising future direction. Furthermore, the performance model and simulator from Chapter 3 could be extended for a Multi-Tier scenario to foster the research efforts in the area.

7.3 Final Remarks

Cloud computing has become a widely adopted model for dynamically provisioning and using IT resources for interactive web-facing applications. The use of a Multi-Cloud environment for such applications, which is explored in this thesis, holds the potential to resolve many of the challenges observed when a single cloud is used. Research work, such as the presented in this thesis, is pivotal for the technical advancement in this novel area. It is instrumental in the development of the next generation middleware technologies, which will make viable the development of efficient and regulatory compliant Multi-Cloud interactive applications.

Bibliography

[1] (2015, Jul. 18) Enterprise privacy authorization language. [Online]. Available: http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/index.html

[2] A. Aarsten, D. Brugali, and G. Menga, "Patterns for three-tier client/server applications," in *Proceedings of the 3rd Conference on the Pattern Languages of Programs (PLoP)*, Monticello, Illinois, USA, 1996.

[3] B. Adler, "Building scalable applications in the cloud," RightScale, Report, 2011.

[4] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Proceedings of the Network Operations and Management Symposium (NOMS)*, Maui, Hawaii, USA, 2012, pp. 204–212.

[5] Amazon. (2015, Jul. 18) Amazon auto scaling. [Online]. Available: http://aws.amazon.com/autoscaling/

[6] Amazon. (2015, Jul. 18) Amazon elasticache. [Online]. Available: http://aws.amazon.com/elasticache/

[7] Amazon. (2015, Jul. 18) Amazon relational database service (amazon RDS). [Online]. Available: http://aws.amazon.com/rds/

[8] Amazon. (2015, Jul. 18) Amazon Route 53. [Online]. Available: http://aws.amazon.com/route53/

[9] Amazon. (2015, Jul. 18) AWS IP address ranges. [Online]. Available: http://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html

[10] Amazon. (2015, Jul. 18) AWS PCI DSS level 1 FAQs. [Online]. Available: http://aws.amazon.com/compliance/pci-dss-level-1-faqs/

[11] Amazon. (2015, Jul. 18) Elastic load balancing. [Online]. Available: http://aws.amazon.com/elasticloadbalancing/

[12] Amazon. (2015, Jul. 18) High performance computing (HPC) on AWS. [Online]. Available: http://aws.amazon.com/hpc/

[13] Amazon. (2015, Jul. 18) Public sector case studies. [Online]. Available: http://aws.amazon.com/government-education/case-studies/

[14] Amazon. (2015, Jul. 18) Summary of the Amazon EC2 and Amazon RDS service disruption. [Online]. Available: http://aws.amazon.com/message/65648/

[15] Amazon. (2015, Jul. 18) Summary of the AWS service event in the US east region. [Online]. Available: http://aws.amazon.com/message/67457/

[16] C. Amza, A. Chanda, A. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite, "Specification and implementation of dynamic web site benchmarks," in *Proceedings of the IEEE International Workshop on Workload Characterization (WWC-5)*, Austin, Texas, USA, 2002, pp. 3–13.

[17] Apache Foundation. (2015, Jul. 18) Apache Delta Cloud. [Online]. Available: http://deltacloud.apache.org/

[18] Apache Foundation. (2015, Jul. 18) Apache Libcloud. [Online]. Available: http://libcloud.apache.org/

[19] Apache Foundation. (2015, Jul. 18) Apache Nuvem. [Online]. Available: http://wiki.apache.org/incubator/Nuvem

[20] Apache Foundation. (2015, Jul. 18) Apache™ Hadoop™! [Online]. Available: http://hadoop.apache.org/

[21] Apache Foundation. (2015, Jul. 18) JClouds. [Online]. Available: http://jclouds.apache.org/

[22] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C. Nechifor, D. Petcu, A. Gericke, and C. Sheridan, "MODAClouds: A model-driven approach for the design and execution of applications on multiple clouds," in *Proceedings of the 4th Workshop on Modeling in Software Engineering (MISE)*, Zurich, Switzerland, 2012, pp. 50–56.

[23] Arjuna, "Arjuna Agility^{TM}:removing the barriers to business agility," University of Zurich, Department of Informatics, Tech. Rep., 2010.

[24] Arjuna Agility. (2015, Jul. 18) Arjuna. [Online]. Available: http://www.arjuna.com/agility

[25] Arjuna Agility. (2015, Jul. 18) What is federation. [Online]. Available: http://www.arjuna.com/what-is-federation

[26] 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," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[27] Australian Government, Department of Communications, "Cloud computing regulatory stock take," Australian Government, Department of Communications, Report Version 1, 2014.

[28] A. Avetisyan, R. Campbell, I. Gupta, M. Heath, S. Ko, G. Ganger, M. Kozuch, D. O'Hallaron, M. Kunze, T. Kwan, K. Lai, M. Lyons, D. Milojicic, H. Y. Lee, Y. C. Soh, N. K. Ming, J.-Y. Luke, and H. Namgoong, "Open Cirrus: A global cloud computing testbed," *Computer*, vol. 43, no. 4, pp. 35–43, 2010.

[29] A. Barker and J. Van Hemert, "Scientific workflow: A survey and research directions," in *Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics (PPAM)*, Gdansk, Poland, 2008, pp. 746–753.

[30] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[31] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "A taxonomy and survey of energy-efficient data centers and Cloud computing systems," *Advances in Computers, M. Zelkowitz (ed.)*, vol. 82, pp. 47–111, 2011.

[32] D. Bernstein and Y. Demchenko, "The ieee intercloud testbed – creating the global cloud of clouds," in *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, vol. 2, Bristol, United Kingdom, 2013, pp. 45–50.

[33] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the intercloud - protocols and formats for cloud computing interoperability," in *Proceedings of the 4th International Conference on Internet and Web Applications and Services (ICIW)*, Venice/Mestre, Italy, 2009, pp. 328–336.

[34] D. Bernstein and D. Vij, "Intercloud directory and exchange protocol detail using XMPP and RDF," in *Proceedings of the 6th World Congress on Services (SERVICES-1)*, Miami, Florida, USA, 2010, pp. 431–438.

[35] D. Bernstein, D. Vij, and S. Diamond, "An intercloud cloud computing economy-technology, governance, and market blueprints," in *Proceedings of the 1st Annual SRII Global Conference (SRII)*, San Jose, California, USA, 2011, pp. 293–299.

[36] D. Bernstein and D. Vij, "Simple storage replication protocol SSRP for intercloud," in *Proceedings of the 2nd International Conference on Emerging Network Intelligence (EMERGING)*, Florence, Italy, 2010, pp. 30–37.

[37] D. Bernstein and D. Vij, "Using semantic web ontology for intercloud directories and exchanges," in *Proceedings of the 11th International Conference on Internet Computing (ICOMP)*, Las Vegas Nevada, USA, 2010, pp. 18–24.

[38] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *Proceedings of the 3rd International Conference on Cloud Computing (CLOUD)*, Miami, Forida, USA, 2010, pp. 370–377.

[39] J. Bowen, "Legal issues in cloud computing," in *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. Goscinski, Eds. Wiley Press, 2011, ch. 24, pp. 593–613.

[40] E. Brewer, "Towards robust distributed systems," in *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, vol. 19, Portland, Oregon, USA, 2000, pp. 7–10.

[41] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[42] R. Buyya and M. Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[43] 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 (ICA3PP)*, Busan, Korea, 2010, pp. 13–31.

[44] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[45] R. Calheiros, A. Toosi, C. Vecchiola, and R. Buyya, "A coordinator for scaling elastic applications across multiple clouds," *Future Generation Computer Systems*, vol. 28, no. 8, pp. 1350–1362, 2012.

[46] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[47] R. Campbell, I. Gupta, M. Heath, S. Y. Ko, M. Kozuch, M. Kunze, T. Kwan, K. Lai, H. Y. Lee, M. Lyons, D. Milojicic, D. O'Hallaron, and Y. C. Soh, "Open Cirrus™

cloud computing testbed: Federated data centers for open source systems and services research," in *Proceedings of the 1st Conference on Hot Topics in Cloud Computing (HotCloud)*, San Diego, California, 2009.

[48] J. Cao, M. Andersson, C. Nyberg, and M. Kihl, "Web server performance modeling using an M/G/1/K*PS queue," in *Proceedings of the 10th International Conference on Telecommunications (ICT)*, vol. 2, Tahiti, Papeete, French Polynesia, 2003, pp. 1501–1506.

[49] E. Carlini, M. Coppola, P. Dazzi, L. Ricci, and G. Righetti, "Cloud federations in contrail," in *Proceedings of Euro-Par 2011: Parallel Processing Workshops*, M. e. a. Alexander, Ed. Bordeaux, France: Springer, 2012, vol. 7155, pp. 159–168.

[50] R. Cattell, "Scalable SQL and NoSQL data stores," *SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2010.

[51] T. CChieu, A. Mohindra, and A. Karve, "Scalability and performance of web applications in a compute cloud," in *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE)*, Beijing, China, 2011, pp. 317–323.

[52] Cecchet, E. and Chanda, A. and Elnikety, S. and Marguerite, J. and Zwaenepoel, W., "Performance comparison of middleware architectures for generating dynamic web content," in *Proceedings of the ACM/IFIP/USENIX International Conference on Middleware (Middleware)*, Rio de Janeiro, Brazil, 2003, pp. 242–261.

[53] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "How to enhance cloud architectures to enable cross-federation," in *Proceedings of the 3rd International Conference on Cloud Computing (CLOUD)*, Miami, Florida, USA, 2010, pp. 337–345.

[54] CESWP. (2015, Jul. 18) Cloud-enabled space weather modelling and data assimilation platform. [Online]. Available: http://www.cybera.ca/projects/completed-projects/ceswp/

[55] Y. Chen, S. Iyer, X. Liu, D. Milojicic, and A. Sahai, "SLA decomposition: Translating service level objectives to system level thresholds," in *Proceedings of the 4th International Conference on Autonomic Computing (ICAC)*, Jacksonville, Florida, USA, 2007, pp. 3–13.

[56] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "Automated anomaly detection and performance modeling of enterprise applications," *ACM Transactions on Computer Systems*, vol. 27, no. 3, pp. 1–32, 2009.

[57] T. Chieu, A. Mohindra, A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE)*, Macau, China, 2009, pp. 281–286.

[58] Cloud Service Measurement Index Consortium (CSMIC). (2015, Jul. 18) Service measurement index (SMI). [Online]. Available: http://csmic.org/

[59] CloudSuite. (2015, Jul. 18) CloudSuite's cloudstone. [Online]. Available: http://parsa.epfl.ch/cloudsuite/web.html

[60] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systtems*, vol. 31, no. 3, pp. 8:1–8:22, 2013.

[61] COUCHBASE, "NoSQL database technology," COUCHBASE, Tech. Rep., 2012.

[62] CSCC Workgroup, "Practical guide to cloud service level agreements version 1.0," Cloud Standards Customer Council (CSCC), Tech. Rep., 2012.

[63] J. Dejun, G. Pierre, and C.-H. Chi, "EC2 performance analysis for resource provisioning of service-oriented applications," in *Proceedings of the 7th International Conference on Service-Oriented Computing (ICSOC)*, Stockholm, Sweden, 2009, pp. 197–207.

[64] A. Di Costanzo, M. De Assuncao, and R. Buyya, "Harnessing cloud technologies for a virtualized distributed computing infrastructure," *Internet Computing, IEEE*, vol. 13, no. 5, pp. 24–33, 2009.

[65] F. Dong and S. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," School of Computing, Queen's University, Kingston, Ontario, Tech. Rep., 2006.

[66] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow," in *Proceedings of the 7th International Conference on Autonomic and Autonomous Systems (ICAS)*, Venice, Italy, 2011, pp. 67–74.

[67] Ebay. (2015, Jul. 18) Ebay. [Online]. Available: http://www.ebay.com/

[68] Ebay Tech Blog. (2015, Jul. 18) Cloud bursting for fun and profit. [Online]. Available: http://ebaytechblog.com/2011/03/28/cloud-bursting-for-fun-and-profit/

[69] Enstratius. (2015, Jul. 18) Enstratius. [Online]. Available: http://www.enstratius.com

[70] European Parliament. (2015, Jul. 18) Data protection directive (95/46/EC). [Online]. Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:HTML

[71] FANN. (2015, Jul. 18) FANN. [Online]. Available: http://leenissen.dk/fann/wp/

[72] H. Fernandez, G. Pierre, and T. Kielmann, "Autoscaling web applications in heterogeneous cloud infrastructures," in *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*, Boston, Massachusetts, USA, 2014, pp. 195–204.

[73] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan, "OPTIMIS: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, 2012.

[74] Fog. (2015, Jul. 18) Fog - they ruby cloud service library. [Online]. Available: http://fog.io/

[75] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proceedings of the Grid Computing Environments Workshop (GCE)*, Austin, Texas, USA, 2008, pp. 1–10.

[76] M. Fowler, *Patterns of enterprise application architecture*. New York, NY, USA: Addison-Wesley, 2003.

[77] M. Fowler. (2015, Jul. 18) Polyglot Persistence. [Online]. Available: http://martinfowler.com/bliki/PolyglotPersistence.html

[78] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Adaptive, model-driven autoscaling for cloud applications," in *Proceedings of the 11th International Conference on Autonomic Computing ICAC*, Philadelphia, Pennsylvania, USA, 2014, pp. 57–64.

[79] S. Garg, S. Versteeg, and R. Buyya, "SMICloud: A framework for comparing and ranking cloud services," in *Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing (UCC 2011)*, Melbourne, Australia, 2011, pp. 210–218.

[80] GeoLite. (2015, Jul. 18) GeoLite2 free downloadable databases. [Online]. Available: http://dev.maxmind.com/geoip/legacy/geolite/

[81] Global Inter-Cloud Technology Forum, "Use cases and functional requirements for inter-cloud computing," Global Inter-Cloud Technology Forum, Tech. Rep., 2010.

[82] Google. (2015, Jul. 18) Google guava. [Online]. Available: https://github.com/google/guava

[83] Grok. (2015, Jul. 18) Grok. [Online]. Available: http://numenta.com/grok/

[84] M. Hassan, B. Song, C. Yoon, H. W. Lee, and E.-N. Huh, "A novel market oriented dynamic collaborative cloud service infrastructure," in *Proceedings of the World Conference on Services (SERVICES)*, Los Angeles, California, USA, 2009, pp. 9–16.

[85] M. Hassan, B. Song, and E.-N. Huh, "A market-oriented dynamic collaborative cloud services platform," *Annals of Telecommunications*, vol. 65, no. 11, pp. 669–688, 2010.

[86] J. Hawkins, S. Ahmad, and D. Dubinsky, "Hierarchical temporal memory including HTM cortical learning algorithm," Numenta Inc, Tech. Rep., September 2011.

[87] J. Hawkins and S. Blakeslee, *On Intelligence*. New York, NY, USA: Times Books, 2004.

[88] P. Helland, "Condos and clouds," *ACM Queue*, vol. 10, no. 11, pp. 20–35, 2012.

[89] N. Ilyadis, "The evolution of next-generation data center networks for high capacity computing," in *Proceedings of the Symposium on VLSI Circuits (VLSIC)*, Honolulu, Hawaii, USA, 2012, pp. 1–5.

[90] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman, and N. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Indianapolis, Indiana, USA, 2010, pp. 159–168.

[91] G. Jung, G. Swint, J. Parekh, C. Pu, and A. Sahai, "Detecting bottleneck in n-tier it applications through analysis," in *Large Scale Management of Distributed Systems*, ser. Lecture Notes in Computer Science, R. State, S. Meer, D. O'Sullivan, and T. Pfeifer, Eds. Springer, 2006, vol. 4269, pp. 149–160.

[92] Kaavo. (2015, Jul. 18) Kaavo. [Online]. Available: http://www.kaavo.com/

[93] A. Kamra, V. Misra, and E. Nahum, "Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites," in *Proceedings of the 12th IEEE International Workshop on Quality of Service (IWQOS)*, Montreal, Canada, 2004, pp. 47–56.

[94] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, "Sky computing," *Internet Computing*, vol. 13, no. 5, pp. 43–51, 2009.

[95] G. Kecskemeti, M. Maurer, I. Brandic, A. Kertesz, Z. Nemeth, and S. Dustdar, "Facilitating self-adaptable inter-cloud management," in *Proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, Munich, Germany, 2012, pp. 575–582.

[96] K. Kelly. (2015, Jul. 18) The Technium: A cloudbook for the cloud. [Online]. Available: http://www.kk.org/thetechnium/archives/2007/11/a_cloudbook_for.php

[97] Kliazovich, D. and Bouvry, P. and Khan, S. U. , "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers," *The Journal of Supercomputing*, vol. 62, pp. 1263–1283, 2012.

[98] M. A. Kozuch, M. P. Ryan, R. Gass, S. W. Schlosser, D. O'Hallaron, J. Cipar, E. Krevat, J. López, M. Stroucken, and G. R. Ganger, "Tashi: Location-aware cluster management," in *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds (ACDC)*, Barcelona, Spain, 2009, pp. 43–48.

[99] KPMG, "2014 cloud survey report," KPMG, Report, 2014.

[100] S. Leberknight. (2015, Jul. 18) Polyglot persistence. [Online]. Available: http://www.sleberknight.com/blog/sleberkn/entry/polyglot_persistence

[101] G. Lewis, "Basics about cloud computing," Software Engineering Institute - Carnegie Mellon, Tech. Rep. September, 2010.

[102] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. Das, "MDCSim: A multi-tier data center simulation, platform," in *Proceedings of the IEEE International Conference on Cluster Computing and Workshops (CLUSTER)*, New Orleans, Louisiana, USA, 2009, pp. 1–9.

[103] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas, "Performance implications of multi-tier application deployments on infrastructure-as-a-service clouds: Towards performance modeling," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1254–1264, 2013.

[104] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Auto-scaling techniques for elastic applications in cloud environments," Department of Computer Architecture and Technology, University of the Basque Country, Tech. Rep. EHU-KAT-IK-09-12, 2012.

[105] J. Lucas Simarro, R. Moreno-Vozmediano, R. Montero, and I. Llorente, "Dynamic placement of virtual machines for cost optimization in multi-cloud environments," in *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS)*, Istanbul, Turkey, 2011, pp. 1–7.

[106] S. Malkowski, D. Jayasinghe, M. Hedwig, J. Park, Y. Kanemasa, and C. Pu, "Empirical analysis of database server scalability using an n-tier benchmark with read-intensive workload," in *Proceedings of the 25th ACM Symposium on Applied Computing (SAC)*, Sierre, Switzerland, 2010, pp. 1680–1687.

[107] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Proceedings of the 5th IEEE Conference on Cloud Computing (CLOUD)*, Honolulu, Hawaii, USA, 2012, pp. 423–430.

[108] A. Marosi, G. Kecskemeti, A. Kertesz, and P. Kacsuk, "FCM: an architecture for integrating iaas cloud systems," in *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization*, Rome, Italy, 2011, pp. 7–12.

[109] A. S. McGough, C. Gerrard, P. Haldane, D. Sharples, D. Swan, P. Robinson, S. Hamlander, and S. Wheater, "Intelligent power management over large clusters," in *Proceedings of the IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing (GREENCOM-CPSCOM)*, Hangzhou, China, 2010, pp. 88–95.

[110] A. McGough, C. Gerrard, J. Noble, P. Robinson, and S. Wheater, "Analysis of power-saving techniques over a large multi-use cluster," in *Proceedings of the 9th International Conference on Dependable, Autonomic and Secure Computing (DASC)*, Sydney, Australia, 2011, pp. 364–371.

[111] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology (NIST), Special Publication 800-145, 2011.

[112] Microsoft. (2015, Jul. 18) Windows azure service disruption update. [Online]. Available: http://azure.microsoft.com/blog/2012/02/29/windows-azure-service-disruption-update/

[113] M. Mont, S. Pearson, and P. Bramhall, "Towards accountable management of identity and privacy: sticky policies and enforceable tracing services," in *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA)*, Prague, Czech Republic, 2003, pp. 377–382.

[114] M. Moreira and E. Fiesler, "Neural networks with adaptive learning rate and momentum terms," IDIAP, Martigny, Switzerland, Tech. Rep. 95-04, October 1995.

[115] V. Mountcastle, "An organizing principle for cerebral function: the unit model and the distributed system," in *The Mindful Brain*, G. Edelman and V. Mountcastle, Eds. Cambridge, Massachusetts, USA: MIT Press, 1978.

[116] M. Mowbray and S. Pearson, "A client-based privacy manager for cloud computing," in *Proceedings of the 4th International ICST Conference on COMmunication System softWAre and middlewaRE (COMSWARE)*, Dublin, Ireland, 2009, pp. 5:1–5:8.

[117] V. Méndez Muñoz, A. Casajús Ramo, V. Fernández Albor, R. Graciani Diaz, and G. Merino Arévalo, "Rafhyc: an architecture for constructing resilient services on federated hybrid clouds," *Journal of Grid Computing*, vol. 11, no. 4, pp. 753–770, 2013.

[118] NeCTAR. (2015, Jul. 18) Home NeCTAR. [Online]. Available: http://nectar.org.au/

[119] Numenta. (2015, Jul. 18) Numenta platform for intelligent computing (NuPIC). [Online]. Available: http://numenta.org/

[120] Numenta. (2015, Jul. 18) Numenta Platform for Intelligent Computing (NuPIC). [Online]. Available: http://numenta.org/nupic.html

[121] A. Núñez, J. Vázquez-Poletti, A. Caminero, G. Castañé, J. Carretero, and I. Llorente, "iCanCloud: A flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, pp. 185–209, 2012.

[122] OASIS. (2015, Jul. 18) eXtensible Access Control Markup Language (XACML) version 3.0. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[123] OAuth. (2015, Jul. 18) OAuth. [Online]. Available: http://oauth.net/

[124] OMG. (2015, Jul. 18) Model driven architecture (mda). [Online]. Available: http://www.omg.org/mda/

[125] OpenID Foundation. (2015, Jul. 18) OpenID. [Online]. Available: http://openid.net/

[126] Oracle, "Oracle database and the oracle database cloud," Oracle, Report, 2012.

[127] Oracle. (2012) Oracle database cloud service.

[128] Parliament of the United Kingdom. (2015, Jul. 18) The Regulation of Investigatory Powers Act 2000 (c.23) (RIP or RIPA). [Online]. Available: http://www.legislation.gov.uk/ukpga/2000/23/contents

[129] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, "Introducing STRATOS: A cloud broker service," in *Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD)*, Honolulu, Hawaii, USA, 2012.

[130] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, 1995.

[131] PCI Security Standards Council. (2015, Jul. 18) PCI-DSS. [Online]. Available: https://www.pcisecuritystandards.org/

[132] S. Pearson and A. Benameur, "Privacy, security and trust issues arising from cloud computing," in *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Indianapolis, Indiana, USA, Nov 2010, pp. 693–702.

[133] S. Pearson, "Taking account of privacy when designing cloud computing services," in *Proceedings of the Workshop on Software Engineering Challenges of Cloud Computing (ICSE)*, Vancouver, Canada, 2009, pp. 44–52.

[134] D. Petcu, E. Di Nitto, D. Ardagna, A. Solberg, and G. Casale, "Towards multi-clouds engineering," in *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Toronto, Canada, 2014, pp. 1–6.

[135] D. Petcu, "Multi-Cloud: expectations and current approaches," in *Proceedings of the 1st International Workshop on Multi-cloud Applications and Federated Clouds (MultiCloud)*, Prague, Czech Republic, 2013, pp. 1–6.

[136] D. Petcu, "Consuming resources and services from multiple clouds," *Journal of Grid Computing*, vol. 12, no. 2, pp. 321–345, 2014.

[137] D. Petcu, C. Crăciun, M. Neagul, S. Panica, B. Di Martino, S. Venticinque, M. Rak, and R. Aversa, "Architecturing a sky computing platform," in *Proceedings of the 4th International Conference Towards a Service-Based Internet (ServiceWave)*, M. Cezon and Y. Wolfsthal, Eds. Ghent, Belgium: Springer-Verlag, 2011, vol. 6569, pp. 1–13.

[138] PingER. (2015, Jul. 18) Ping end-to-end reporting. [Online]. Available: http://www-iepm.slac.stanford.edu/pinger/

[139] C. Quinton, N. Haderer, R. Rouvoy, and L. Duchien, "Towards multi-cloud configurations using feature models and ontologies," in *Proceedings of the 1st International Workshop on Multi-cloud Applications and Federated Clouds (MultiCloud)*, Prague, Czech Republic, 2013, pp. 21–26.

[140] C. Quinton, D. Romero, and L. Duchien, "SALOON: a platform for selecting and configuring cloud environments," *Software: Practice and Experience* ***(to appear)***, 2015.

[141] I. Raicu, I. Foster, and Y. Zhao, "Many-task computing for grids and supercomputers," in *Proceedings of the 2nd Workshop on ManyTask Computing on Grids and Supercomputers (MTAGS)*, Austin, Texas, USA, 2008, pp. 1–11.

[142] A. O. Ramirez, "Three-Tier architecture," *Linux Journal*, vol. 2000, no. 75, 2000.

[143] Red Hat. (2015, Jul. 18) Drools. [Online]. Available: http://www.drools.org/

[144] RightScale. (2015, Jul. 18) RightScale. [Online]. Available: http://www.rightscale.com/

[145] A. Robertson, B. Wittenmark, and M. Kihl, "Analysis and design of admission control in web-server systems," in *Proceedings of the American Control Conference (ACC)*, vol. 1, Denver, Colorado, USA, 2003, pp. 254–259.

[146] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Muñ andoz, and G. Tofetti, "Reservoir - when one cloud is not enough," *Computer*, vol. 44, no. 3, pp. 44–51, 2011.

[147] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, L. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres *et al.*, "The RESERVOIR model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 1–11, 2009.

[148] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente, "From infrastructure delivery to service management in clouds," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1226–1240, 2010.

[149] RUBiS. (2015, Jul. 18) RUBiS: Rice university bidding system. [Online]. Available: http://rubis.ow2.org/

[150] Y. Sakashita, K. Takayama, A. Matsuo, and H. Kurihara, "Cloud fusion concept," *FUJITSU Scientific & Technical Journal (FSTJ)*, vol. 48, no. 2, pp. 143–150, 2012.

[151] Scalr. (2015, Jul. 18) Scalr. [Online]. Available: http://scalr.net/

[152] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," *The Proceedings of the VLDB Endowment (PVLDB)*, vol. 3, no. 1-2, pp. 460–471, 2010.

[153] B. Simmons, H. Ghanbari, M. Litoiu, and G. Iszlai, "Managing a saas application in the cloud using paas policy sets and a strategy-tree," in *Proceedings of the 7th International Conference on Network and Services Management (CNSM)*, Paris, France, 2011, pp. 343–347.

[154] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy, "Autonomic mix-aware provisioning for non-stationary data center workloads," in *Proceedings of the 7th International Conference on Autonomic Computing (ICAC)*, Washington, DC, USA, 2010, pp. 21–30.

[155] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson, "CloudStone: Multiplatform, multi-language

benchmark and measurement tools for web 2.0," in *Proceedings of the Cloud Computing and Its Applications Conference (CCA)*, Chicago, Illinois, USA, 2008.

[156] S. Sotiriadis, N. Bessis, and N. Antonopoulos, "Towards inter-cloud schedulers: A survey of meta-scheduling approaches," in *Proceedings of the International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, Barcelona, Catalonia, Spain, 2011, pp. 59–66.

[157] I. Sriram, "SPECI, a simulation tool exploring cloud-scale data centres," in *Cloud Computing*, ser. Lecture Notes in Computer Science, M. Jaatun, G. Zhao, and C. Rong, Eds. Springer, 2009, vol. 5931, pp. 381–392.

[158] SYSSTAT. (2015, Jul. 18) SYSSTAT. [Online]. Available: http://sebastien.godard.pagesperso-orange.fr/

[159] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "Prepare: Predictive performance anomaly prevention for virtualized cloud systems," in *Proceedings of the 32nd International Conference on Distributed Computing Systems (ICDCS)*, Philadelphia, Pennsylvania, USA, 2012, pp. 285–294.

[160] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling virtual machine performance: Challenges and approaches," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 3, pp. 55–60, 2010.

[161] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Computing Surveys*, vol. 47, no. 1, pp. 7:1–7:47, 2014.

[162] TPC BenchmarkTM W 1.8, "TPC BENCHMARK W (Web Commerce)," Transaction Processing Performance Council (TPC), San Francisco, California, USA, Specification, Version 1.8, February 2002.

[163] TR02-388, "Bottleneck characterization of dynamic web site benchmarks," Department of Computer Science Rice University, Houston, Texas, USA, Tech. Rep., 2002.

[164] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," in *Proceedings of*

the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Banff, Alberta, Canada, 2005, pp. 291–302.

[165] J. Varia, "Best practices in architecting cloud applications in the AWS Cloud," in *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. Goscinski, Eds. Wiley Press, 2011, ch. 18, pp. 459–490.

[166] D. Vij, D. Bernstein, M. Morsey, P. Grosso, T. Magedanz, and A. Willner, "Software defined bearer intercloud networks semantic-based network exchange for the ieee p2302 intercloud approach," in *Proceedings of the 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, Hua Hin, Thailand, 2015, pp. 1–6.

[167] T. Vogl, J. Mangis, A. Rigler, W. Zink, and D. Alkon, "Accelerating the convergence of the back-propagation method," *Biological Cybernetics*, vol. 59, no. 4-5, pp. 257–263, 1988.

[168] K. Zachos, J. Lockerbie, B. Hughes, and P. Matthews, "Towards a framework for describing cloud service characteristics for use by chief information officers," in *Proceedings of the Workshop on Requirements Engineering for Systems, Services and Systems-of-Systems (RESS)*, Trento, Italy, 2011, pp. 16–23.

[169] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *Proceedings of the 4th International Conference on Autonomic Computing (ICAC)*, Jacksonville, Florida, USA, 2007, pp. 27–37.

[170] Z. Zhang and W. Fan, "Web server load balancing: A queueing analysis," *European Journal of Operational Research*, vol. 186, no. 2, pp. 681–693, 2008.