Resource Provisioning in Spot Market-based Cloud Computing Environments

William Voorsluys

Submitted in total fulfilment

of the requirements of the degree of

Doctor of Philosophy

October 2014

Department of Computing and Information Systems

The University of Melbourne, Australia

Abstract

Recently, cloud computing providers have started offering unused computational resources in the form of dynamically priced virtual machines (VMs), also known as "spot instances". In spite of the apparent economical advantage, an intermittent nature is inherent to these biddable resources, which may cause VM unavailability. When an out-of-bid situation occurs, i.e. the current spot price goes above the user's maximum bid, spot instances are terminated by the provider without prior notice.

This thesis presents a study on employing cloud computing spot instances as a means of executing computational jobs on cloud computing resources. We start by proposing a resource management and job scheduling policy, named SpotRMS, which addresses the problem of running deadline-constrained compute-intensive jobs on a pool of low-cost spot instances, while also exploiting variations in price and performance to run applications in a fast and economical way. This policy relies on job runtime estimations to decide what are the best types of spot instances to run each job and when jobs should run. It is able to minimise monetary spending and make sure jobs finish within their deadlines.

We also propose an improvement for SpotRMS, that addresses the problem of running compute-intensive jobs on a pool of intermittent virtual machines, while

also aiming to run applications in a fast and economical way. To mitigate potential unavailability periods, a multifaceted fault-aware resource provisioning policy is proposed. Our solution employs price and runtime estimation mechanisms, as well as three fault tolerance techniques, namely checkpointing, task duplication and migration.

As a further improvement, we equip SpotRMS with prediction-assisted resource provisioning and bidding strategies. Our results demonstrate that both costs savings and strict adherence to deadlines can be achieved when properly combining and tuning the policy mechanisms. Especially, the fault tolerance mechanism that employs migration of VM state provides superior results in virtually all metrics.

Finally, we employ a statistical model of spot price dynamics to artificially generate price patterns of varying volatility. We then analyse how SpotRMS performs in environments with highly variable price levels and more frequent changes. Fault tolerance is shown to be even more crucial in such scenarios.

Declaration

This is to certify that:

(i) the thesis comprises only my original work,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of table, maps, bibliographies, appendices and footnotes.

William Voorsluys

Acknowledgements

This thesis could not have been completed without the patience, determination, encouragement, and support offered by friends, colleagues, and organisations.

First and foremost, I would like to thank my supervisor, Prof. Rajkumar Buyya, for the opportunity of pursuing doctoral studies in his group. Raj has always shown patience, encouragement and pragmatism, and has given me the freedom to pursue independent studies while guiding me through good and hard moments of my PhD endeavour.

Members of my PhD committee Prof. Rao Kotagiri and Dr Rodrigo Calheiros deserve my sincere thanks for their support and helpful suggestions.

Special thanks to Prof. Richard Sinnott for the opportunity to perform exciting work at the Melbourne e-Research Group, and for the help and support in getting this thesis done.

I thank the CLOUDS Laboratory, the Department of Computing and Information Systems, and University of Melbourne for providing excellent facilities and incomparable academic support at all stages of my candidature. I'm also grateful to the University, as well the Australian Government and its funding agencies for providing a generous scholarship and travel support.

I'm grateful to Microsoft Research Asia for awarding me a fellowship, which provided invaluable financial and academic support.

I'm eternally indebted to my colleagues of the CLOUDS Laboratory where I got immense support and made the best friendships of my life. In particular I thank Suraj Pandey, Mohsen Amini, Adel Toosi, and Deepak Poola, whose sincere friendship made lab life a lot more pleasurable. To Amir Vahid for the useful comments on this thesis. To Mukkadim Pathan, Mustafizur Rahman, Marco Netto, Marcos Assuncao, Saurabh Garg, Bahman Javadi, and Rajiv Ranjan, whose experience provided invaluable advice to a newcomer. To James Broberg and Srikumar Venugopal for providing important initial guidance into practical and ground-breaking research. Many thanks for all other members of the lab, past and present, whose enthusiasm has continuously helped me keep motivated.

Many thanks to colleagues Davis Marques and Philip Greenwood of the Melbourne e-Research group for helping with proof-reading parts of this thesis.

My heartfelt thanks to my family, especially my parents, who always supported my endeavours despite long periods of my absence.

My most special thanks to my wife Barbara and our little Olivia, my most precious gifts.

Finally, I thank the anonymous reviewers of this thesis for the useful feedback.

William Voorsluys
Melbourne, Australia
October 2014

List of Publications

During the course of this project, a number of public presentations have been made which are based on the work presented in this thesis. Other publications were also produced, as collaborations with colleagues, but their content is not necessarily part of this thesis. All publications are listed here for reference. Wherever chapter contents are based on one or more of these publications, this fact is mentioned in the beginning of each chapter, as well as collectively in section 1.8.

- Pandey, S., Jin, C., Voorsluys, W., Rahman, M., and Buyya, R. Gridbus Worklow Management System on Clouds and Global Grids. In *Proceedings of the 4th IEEE International Conference on eScience* (2008), IEEE, pp. 323–324

- Voorsluys, W., Broberg, J., Venugopal, S., and Buyya, R. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *Proceedings of the 1st IEEE International Conference on Cloud Computing (CloudCom 2009)* (Berlin, Heidelberg, Sept. 2009), vol. 5931 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 1–12

- Pandey, S., Voorsluys, W., Rahman, M., Buyya, R., Dobson, J., and Chiu, K. A Grid Workflow Environment for Brain Imaging Analysis on

Distributed Systems. *Concurrency and Computation: Practice and Experience 21*, 16 (2009)

- Voorsluys, W., Broberg, J., and Buyya, R. Introduction to Cloud Computing. In *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. Goscinski, Eds. Wiley, 2011, ch. 1, pp. 3–42

- Pandey, S., Voorsluys, W., Niu, S., Khandoker, A., and Buyya, R. An autonomic cloud environment for hosting ECG data analysis services. *Future Generation Computer Systems 28*, 1 (May 2011), 147–154

- Voorsluys, W., Garg, S. K., and Buyya, R. Provisioning Spot Market Cloud Resources to Create Cost-effective Virtual Clusters. In *Proceedings of the 11th IEEE International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-11)* (Melbourne, Australia, 2011), Y. Xiang, A. Cuzzocrea, M. Hobbs, and W. Zhou, Eds., vol. 7016 of *Lecture Notes in Computer Science*, Springer, pp. 395–408

- Voorsluys, W., and Buyya, R. Reliable Provisioning of Spot Instances for Compute-intensive Applications. In *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications* (Fukuoka, Japan, Mar. 2012), IEEE, pp. 542–549

Contents

List of Tables

List of Figures

Chapter 1

Introduction

WHEN plugging an electric appliance into an outlet, we care neither how electric power is generated nor how it gets to that outlet. This is possible because electricity is virtualised, i.e. it is readily available from a wall socket that hides power generation stations and a huge distribution grid. When extended to information technologies, this concept is used for delivering useful functions while hiding how their internals work. Computing itself, to be considered fully virtualised, must allow computers to be built from distributed resources such as processing, storage, data, and software resources and offer these in a transparent manner [37].

Technologies such as *Cluster, Grid,* and more recently, *Cloud* computing, have all aimed at allowing access to large amounts of computing power in a fully virtualised manner, by seamlessly aggregating resources and offering a single system view. In addition, an important aim of these technologies has been delivering computing as a utility. Utility computing describes a business model for on-demand delivery of computing power; consumers pay providers based on usage ("pay-as-you-go"), similar to the way in which we currently obtain services from traditional public utility services such as water, electricity, gas, and telephony.

1.1 Inspiration for Cloud Computing

Cloud computing is at the centre of a switch from in-house generated computing power into utility-supplied computing resources delivered over the Internet, typically as Web services. Its inspiration comes directly from what occurred about a century

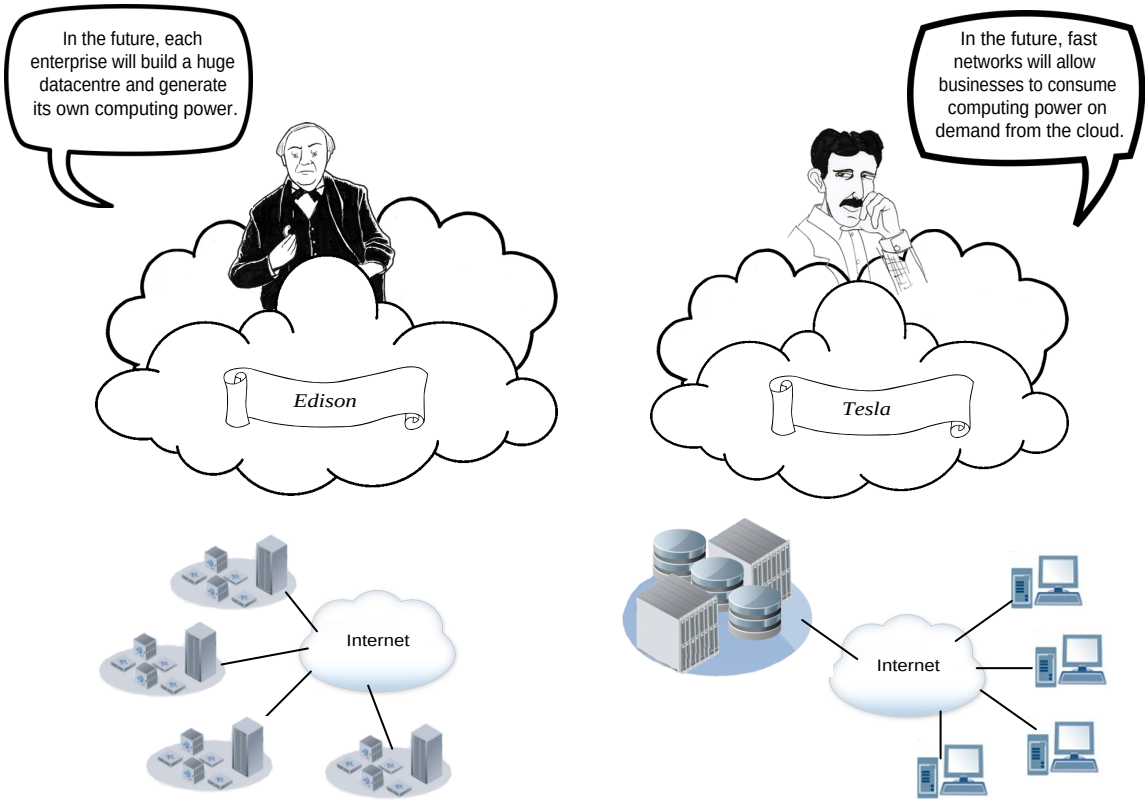


Figure 1.1: The advent of cloud computing, as a parallel to the War of Currents.

ago when factories, which used to generate their own electric power, realised that it was cheaper just plugging their machines into the newly formed electric power grid [19].

Building large power generating plants and a power grid to transmit electricity over long distances was possible not only because of technological advances (e.g. alternating current), but also because of commercial sense and the ideas of a few visionaries who aimed at changing the mindset of the proponents of the then dominant direct current system.

An often dramatic turn of events surrounded that switch, known as “The War of Currents”, antagonised by Thomas Edison, who favoured locally generated power transmitted to short distances over direct current, as opposed to Nikola Tesla and George Westinghouse, who preferred power to be generated by fewer, large, often distant, power plants and transmitted over long distances over high-voltage alternating current power lines (Figure 1.1).

Those events eventually led to the adoption of a technically and commercially sound model of centralised, large-scale, and efficient power generation method and its transmission over high-capacity lines.

Similarly, the switch to cloud computing is enabled by faster networks and new technologies such as virtualisation. However, the biggest change is a commercial one. The economies of scale brought by fewer, large-scale data centres allow providers to offer computing via a novel model. The main principle behind the cloud computing model is offering computing, storage and software "as a service".

In addition to "raw" computing and storage, cloud computing providers usually offer a broad range of value-added services. They also include application programming interfaces (APIs) and development tools that allow developers to build seamlessly scalable applications upon their services. The ultimate goal is allowing customers to run their everyday IT infrastructure "in the cloud".

Cloud computing has been coined as a generic umbrella term to describe a category of sophisticated on-demand computing services first offered by commercial providers, such as Amazon, Google, and Microsoft. In a more objective definition [17], cloud computing denotes a model through which a computing infrastructure is viewed as a "cloud", from which businesses and individuals access applications from anywhere in the world on demand. Figure 1.2 depicts a high-level view of how organisations and individuals would access their resources in the cloud computing model.

Similar to the "War of Currents", a lot of hype has surrounded the cloud computing area, often considered the most significant switch in the IT world since the advent of the Internet [19]. Therefore, researchers and practitioners are faced with new challenges, but also opportunities, when working their way around several aspects of this new business model. Often, the business and technical aspects come together, prompting the development of new technical approaches to overcome economic challenges. This is the case of the work presented in this thesis, which aims at tackling both economic and technical aspects of utilising cloud computing resources, by means of a practical approach involving new algorithms, heuristics, policies, and performance evaluations.

This thesis touches a few research areas, most notably: resource management and scheduling, computational markets, and dependable computing. The rest of this introduction briefly discusses the context of this research work among these areas;

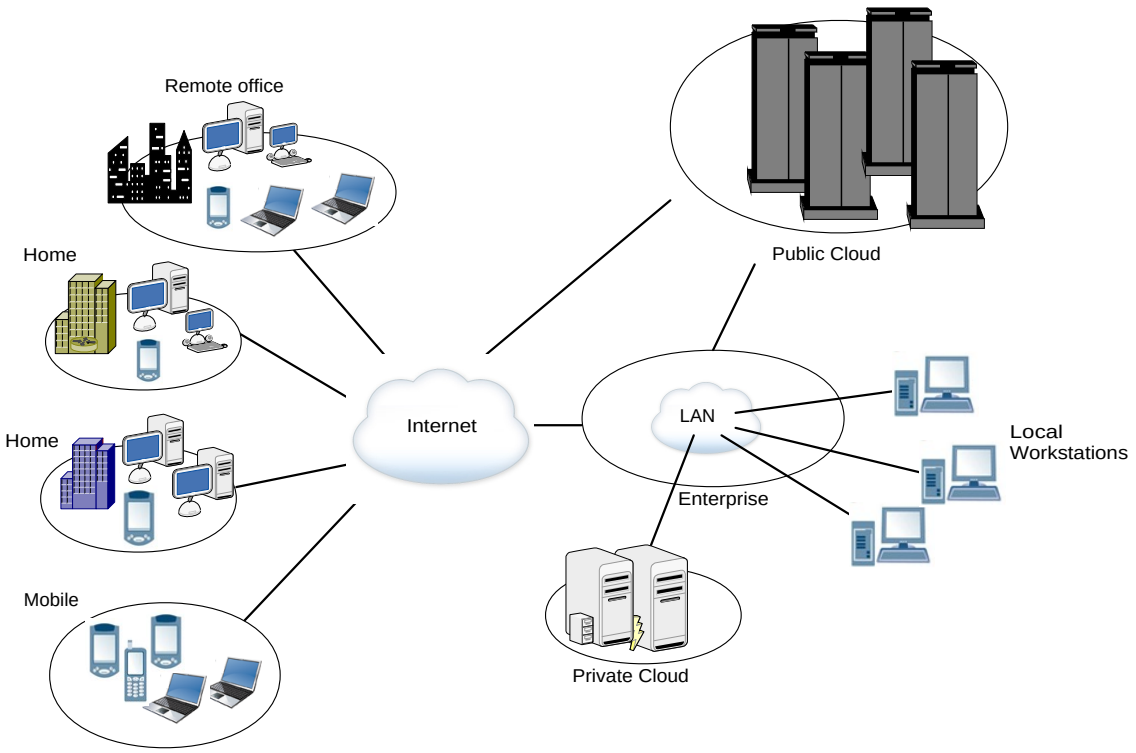


Figure 1.2: The cloud computing model: businesses and individuals access resources and applications from anywhere in the world on demand, potentially using a variety of devices. Enterprises may often maintain a "private cloud" and supplement their needs with additional computing power from public cloud computing providers.

identifies key challenges in the management of cloud resources; presents this thesis' research issues as well as the approach and methodology to solve them, and outlines the list of contributions and main findings.

1.2 Resource Management and Computational Economy

The problem of efficiently harnessing the computing power of distributed resources has been extensively studied [79, 58]. This problem, known as Resource Management and Scheduling (RMS), comprises the activities required to allocate suitable computing resources to jobs with the goal of optimising a certain objective. Traditional objective functions include minimising the time of job completion, maximising job throughput, and maximising the utilisation of resources.

Grid Computing research [13, 14] introduced the computational economy style of resource management, where resources are made available at a price, usually set based on supply and demand. In this "unconventional" style of RMS, novel objective functions were considered, such as executing jobs at the lowest possible cost, and meeting budget constraints [32].

In a computational market, major players can comprise: end-users, who consume particular resources (compute power, storage) to perform computational jobs; providers, who offer these resources backed by data centres built to serve many users, and brokers, who mediate and negotiate access to provider resources, thus reducing complexity for end-users.

Players will often behave differently, adopting strategies that suit their own interests. End-users will adopt a strategy of executing their jobs at the minimum possible cost while also obtaining a certain QoS level, often having to deal with a trade-off between these objectives [42]. Providers will manage their data centres as efficiently as possible in order to maximise profit, but without neglecting QoS level guarantees. Brokers will exploit changes in the market to obtain profit, for example, by performing jobs on behalf of users at times when resources are offered at a discount.

With the emergence of cloud computing, fresh challenges in the problem of resource management and scheduling have arisen, which limit the applicability of existing resource management and scheduling solutions originally developed for Grid computing environments and locally managed clusters. These new challenges are mainly due to novel underlying technological features of cloud platforms, as well the need for new optimisation objectives.

Novel features include: virtualisation, which allows multiple virtual machines to be hosted on a single server, thus improving server utilisation for providers; multi-tenancy, the capability of leasing data centre space to multiple customers, increasing efficiency at the data centre level, and dynamic resource pricing, where resource costs varying over time based on supply and demand. New optimisation objects include: elasticity, the capability of seamlessly increasing and decreasing the amount of resources one can use at a given time depending on demand; and energy-efficiency, the necessity of decreasing a data centre's demand for electricity and cooling.

These new challenges potentially affect the role of all players of a computational economy. Particularly, from a user perspective, dynamic resource pricing adds a new

level of complexity to the process of deciding when to perform a computation. RMS architectures and systems aimed at cloud computing must be designed with this complexity in mind, especially incorporating temporal and monetary constraints.

1.3 Dynamic Resource Pricing in Cloud Markets and Availability Concerns

Recently, cloud computing providers have started offering unused computational resources in the form of dynamically priced virtual machines (VMs), which in Amazon EC2 terminology are also known as "spot instances" [95]. These VMs are generally available at prices significantly lower than their standard statically priced counterparts. In order to lease a spot instance, a user specifies a maximum price (the bid) they are willing to pay per hour for one or more instance of a particular type. Instances will run only when the current price computed by the provider (the "spot price") is lower than the bid.

This computational market, since introduced by Amazon Web Services [95], has been considered as the first step towards a full-fledged market economy for computational resources [109]. Prices vary independently for each available data centre ("availability zone" in Amazon terminology), spot instance type, and operating system choice. Not all type/OS combinations are available in all data centres. In other words, there are multiple spot markets from where to choose suitable computational resources, making the provisioning problem significantly challenging.

In spite of the economic advantage of the spot market model, an intermittent nature is inherent to biddable resources, which may cause VM unavailability. When an out-of-bid situation occurs, i.e. the current spot price goes above the user's maximum bid, spot instances are terminated by the provider without prior notice. Such situations may lead to prolonged application execution times or to the inability of applications to finish within a specified deadline. Solutions that can mitigate this uncertainty include: sophisticated bidding strategies that choose bid values likely to reduce the chance of an out-of-bid situation, and fault-tolerance techniques, such as checkpointing [107], that minimise the impact of VM unavailability.

Despite the novelty of this area, some existing solutions tackle the problem of running computational jobs in spot market resources. Andrzejak et al. [3] have pro-

posed a probabilistic decision model to help users decide how much to bid for a certain spot instance type in order to meet a certain monetary budget or a deadline. Yi et al. [107] proposed a method to reduce costs of computations and providing fault-tolerance when using EC2 spot instances. Based on the price history, they simulated how several checkpointing policies would perform when faced with out-of-bid situations. This thesis builds up on previous research by proposing new bidding strategies aimed at avoiding out-of-bid situations, as well as new fault-tolerance techniques such as migration and job duplication.

1.4 Research Issues

This thesis explores employing spot market resources as a means of executing computationally intensive applications on cloud computing resources. Specifically, it aims at answering the following **research question**:

What are the advantages, disadvantages, and limitations of solely employing spot market resources as a means of executing deadline-constrained applications?

A number of objectives must be accomplished in order to answer this question, namely:

- Identifying the unique challenges and potential advantages that the use of dynamically priced resources bring to the issue of resource management and scheduling in cloud computing;
- Understanding how failures due to out-of-bid situations affect application execution, especially regarding time and cost constraints;
- Designing, implementing, and evaluating bidding strategies that aim to prevent failures and incorporating fault-tolerance mechanisms to mitigate their effect;
- Devising resource management and scheduling policies employing bidding strategies and fault-tolerance mechanisms;
- Analysing the effectiveness of the proposed mechanisms in a variety of scenarios, especially when volatility of prices is present;
- Evaluating the potential overheads of fault-tolerance mechanisms on the performance of applications;

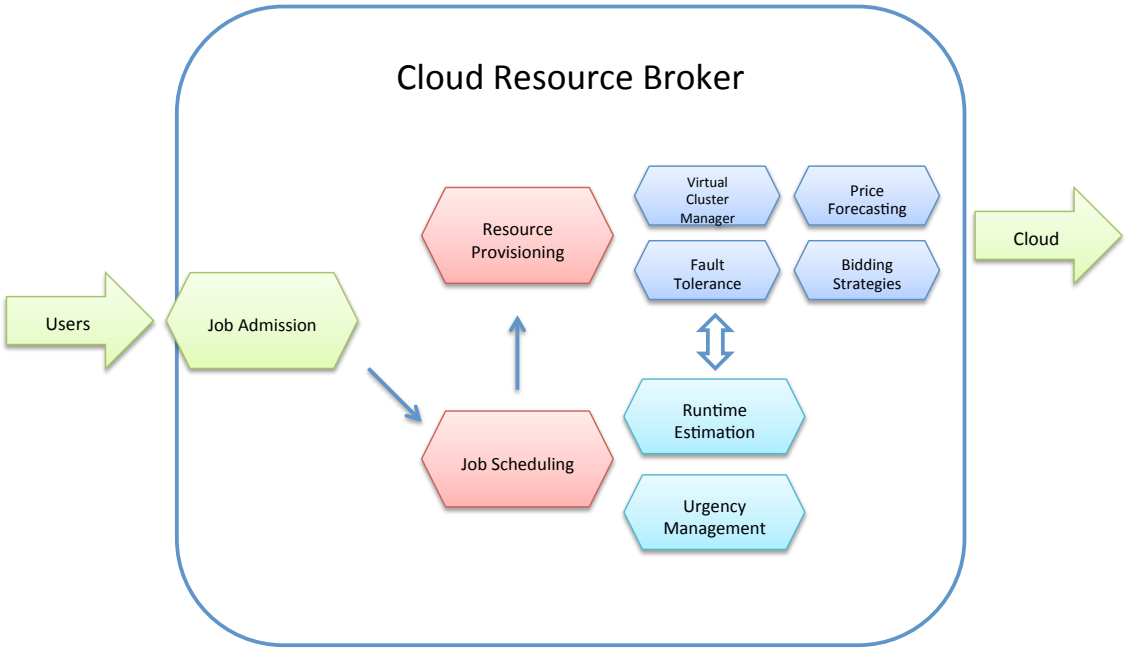


Figure 1.3: Components of a cloud resource broker containing mechanisms for managing the full life-cycle of applications.

1.5 The Solution

In order to allow cost-effective and reliable execution of applications on spot market resources, a multi-faceted resource management system, named SpotRMS, is introduced, which includes multiple policies and mechanisms. The main components of this system are a resource provisioning and job scheduling algorithm, a runtime estimation mechanism, a set of bidding strategies, and a set of fault-tolerance mechanisms. SpotRMS is contained within a resource brokering system, whose task is to manage the entire life-cycle of both applications and resources. The ultimate objective of the brokering system is to run users' applications within their specified deadline and at low cost.

Figure 1.3 depicts an architecture of a broker system that contains the aforementioned policies and mechanisms. The main features of this architecture, namely bidding strategies and fault tolerance mechanisms are the ones studied in detail in this thesis.

1.6 Methodology

To validate the results, an assessment of the effectiveness of SpotRMS is given in different scenarios. To evaluate the effect of the SpotRMS algorithm and its use of runtime estimations, a comparison with a best-case baseline method that uses perfect information about job length, parallelism, and spot price is given. A worst-case scenario where only fixed-price spot market resources are used is also presented.

Regarding the effect of fault-tolerance methods, an existing hourly checkpointing mechanism is implemented and used as a baseline. Furthermore, when applying spot price prediction to resource provisioning and bidding strategies, a comparison of results with history-based methods is given. A best-case scenario where actual future prices are used by the algorithm is also presented.

This work has built and utilised multiple tools to achieve the results presented in this thesis. Primarily, the results are based on trace-based discrete event simulations, carried out using a dedicated spot market simulator, which was augmented by job traces from a real cloud production system and spot price logs from the Amazon EC2 spot market. A discussion of the usage of such tools in as an integral part of the methodology and work as a whole.

1.6.1 Spotsim

Spotsim is a discrete event simulator built on top of the CloudSim [16] framework. It models all aspects of a cloud computing spot market, dividing it into two main subsystems: cloud provider and broker.

The cloud provider contains models of multiple instance types and their characteristics, trace-based instance pricing, usage accounting, bidding management and out-of-bid events.

The broker logic models job submission, provisioning of new spot instances, scheduling of jobs on instances, detection and recovery from out-of-bid situations, as well as collection of a range of performance metrics.

Spotsim contains a modular implementation of SpotRMS' architecture, which is discussed in details in section 3.2.

1.6.2 Job Traces

Simulated jobs are modelled according to traces collected from a real production parallel system. Such traces are made available in the Standard Workflow Format (SWF) by the Parallel Workloads Archive [34]. A trace is a collection of records, each corresponding to a job execution. Each record contains 18 fields with information about each job, of which 5 are relevant to this thesis: arrival time, estimated run time, actual run time, user ID, and number of nodes/processors used.

The trace used in the works of chapters 3, 4, and 5, was obtained from LCG (Large Hadron Collider Computing Grid) at CERN [34]. It is composed of grid-like embarrassingly parallel tasks. The first 100,000 jobs were selected, spanning a period of seven days of simulation time. About 60% of the jobs are short jobs (less than 10 minutes), with the majority having a runtime between 1 and 10 minutes; the remaining 40% are long jobs with durations from about 1 hour to 4 days. The arrival pattern of the workload has been analysed by Li et al. [61].

The original trace did not contain information about user supplied job runtime estimates and deadlines. User runtime estimates were generated according to the model of Tsafrir et al. [92]. Deadlines were then generated based on these runtime estimates. The job deadline corresponds to its submission time plus the user supplied runtime estimate multiplied by a random multiplier, uniformly generated between 1.5 and 4.

1.6.3 Price Traces

This work has collected and used real price variation traces obtained from the Amazon Web Services' EC2 platform. It includes most spot instance types and data centres available at the time of collection, particularly standard and high-cpu types, but excluding GPU-based and HPC instance types, which are tailored for specialised applications.

The work also uses price logs from multiples time periods according to the availability of such logs at the time the simulations were performed and considering the objective of each set of simulations.

For the simulations of chapter 3 logs comprising dates between March 1st, 2010 and February 1st, 2011, of Amazon's US-EAST region were used. At that time,

Amazon had a "per region" pricing policy whereby spot prices of each instance type were uniform across all data centres of a region, although each data centre is itself treated as a separate "silo". The simulations of that chapter reflects this policy. The model used in chapter 5 was based on prices of the US-WEST region from February 2010 to mid-February 2011, where prices were also given per region. This model was developed in the work of Javadi et al. [51]. For comparison purposes, wherever a model is used to predict spot prices, simulation periods are set so that they coincide with the model's.

Conversely, in the simulations of chapter 4, newer traces from a period where Amazon EC2 spot prices were given per availability zone (data centre) rather than per region (July 5^{th}, 2011 and October, 15^{th}, 2011) have been used. In view of the fact that different data centres had varying spot price patterns, the technique of migrating workload between instances, possibly of the same type but in another data centre, became especially relevant. Chapter 4 includes a detailed explanation of why migration was needed under this pricing policy and highlights the advantages of this technique over less sophisticated ones, such as checkpointing and job duplication.

1.7 Contributions and Main Findings

This thesis proposes policies, mechanisms, and performance evaluations that collectively contribute in the direction of answering the research question. More specifically the following are the contributions and main findings of this research:

1. **SpotRMS: A resource provisioning and job scheduling policy**. A heuristics-based policy has been developed that addresses the problem of dynamically building a virtual cluster out of spot instances and utilising them to run compute-intensive applications in a time and cost-effective manner. Results show that runtime estimation methods have a significant impact on monetary cost, although precise estimation does not always lead to better results. Also the use of SpotRMS leads to savings of up to 60% when compared to the worst-case scenario, i.e. using only fixed-price instances, and is only 23% worst than the best-case, i.e. when the most cost effective spot instances are always selected and run at 100% utilisation.

2. **Bidding strategies**. The policies developed in this work employ multiple bidding strategies that take advantage of information about spot price history and estimates of application runtime to allow informed decision-making in the resource provisioning process. Results demonstrate that both costs savings and stricter adherence to deadlines can be achieved when properly combining bidding strategies and tuning policy mechanisms.

3. **Fault-tolerance mechanisms** (checkpointing, migration, and job duplication) that allow the policy to recover from out-of-bid situations by minimising the amount of lost work. It is observed that the fault tolerance mechanism that employs migration of VM state provides superior results in virtually all metrics when compared to an existing hourly-checkpointing solution.

4. **A prediction-assisted resource provisioning and bidding strategy**, which improves SpotRMS with the addition of a statistical model of spot price dynamics. This approach shows superior results, especially by reducing monetary cost, when compared with the original strategies that use history-based or current price information on bidding and provisioning decisions.

5. **A study on the effect of spot price volatility** on the effectiveness of SpotRMS. This study takes into account price variations patterns generated by a modified statistical model. Results demonstrate that under more volatile scenarios fault-tolerance mechanisms become considerably more useful and accurate runtime estimation provides significant benefit.

6. A **performance evaluation study on the cost of live migration of virtual machines** as a limitation for its employment in real-world scenarios. This study is aimed at validating the use of this technique as a fault-tolerance mechanism. Results show that the impact of live migration on the performance of applications is low but cannot be disregarded, especially when multiple migrations are needed in a short period.

1.8 Thesis Organisation

This thesis is organised into 7 chapters.

The core chapters of this thesis are derived from various research publications produced during the course of the PhD candidature, as detailed below.

Chapter 2 presents a comprehensive survey of the cloud computing field. It focuses specifically on the roots of cloud computing by surveying the main technological advancements that significantly contributed to the advent of this emerging field. It also presents and discusses the most relevant works related to the topics studied in this thesis. This chapter is derived from [103]:

Voorsluys, W., Broberg, J., and Buyya, R. Introduction to Cloud Computing. In *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. Goscinski, Eds. Wiley, 2011, ch. 1, pp. 3–42

Chapter 3 presents a system architecture of a broker system for managing resources and scheduling applications in spot market scenarios, named SpotRMS. Specifically, it presents a resource provisioning policy that addresses the problem of running deadline-constrained computational jobs on a pool of resources composed solely of spot instances, while exploiting variations in price and performance to run applications in a fast and economical way. This chapter is derived from [106]:

Voorsluys, W., Garg, S. K., and Buyya, R. Provisioning Spot Market Cloud Resources to Create Cost-effective Virtual Clusters. In *Proceedings of the 11th IEEE International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-11)* (Melbourne, Australia, 2011), Y. Xiang, A. Cuzzocrea, M. Hobbs, and W. Zhou, Eds., vol. 7016 of *Lecture Notes in Computer Science*, Springer, pp. 395–408

Chapter 4 introduces strategies for fault-tolerant resource management for reliably running applications on spot market resources. A multi-faceted policy is proposed, which employs price and runtime estimation mechanisms, as well as three fault tolerance techniques, namely checkpointing, task duplication and migration. This chapter is derived from [105]:

Voorsluys, W., and Buyya, R. Reliable Provisioning of Spot Instances for Compute-intensive Applications. In *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications* (Fukuoka, Japan, Mar. 2012), IEEE, pp. 542–549

Chapter 5 proposes the use of a statistical model for predictive bidding strategies, which use prediction of price variation patterns aiding SpotRMS in choosing the most appropriate time slots to run jobs. This chapter also presents a study on the

effect of price volatility on the resource management strategies presented in the previous chapters.

Chapter 6 evaluates the viability of virtual machine migration as a method to achieve fault tolerance for a variety of applications, including compute-intensive jobs and multi-tier modern Internet applications. This chapter is derived from [104]:

Voorsluys, W., Broberg, J., Venugopal, S., and Buyya, R. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *Proceedings of the 1st IEEE International Conference on Cloud Computing (CloudCom 2009)* (Berlin, Heidelberg, Sept. 2009), vol. 5931 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 1–12

Chapter 7 concludes this thesis, synthesising the thesis objectives and the approach to achieve them, summarising the role of the SpotRMS system and its components, and discussing the impact of the main findings on cloud computing research. It also addresses how these findings expose the need for more research on computational markets by expounding on topics of potential future work.

Chapter 2

A Survey of Cloud Computing and Related Work

We are currently experiencing a switch in the IT world, from in-house generated computing power into utility-supplied computing resources delivered over the Internet as Web Services. Delivering computing as a utility is defined as "on demand delivery of infrastructure, applications, and business processes in a security-rich, shared, scalable, and based computer environment over the Internet for a fee" [80].

This model brings benefits to both consumers and providers of IT services. Consumers can attain reduction on IT-related costs by choosing to obtain cheaper services from external providers as opposed to heavily investing on IT infrastructure and personnel hiring. The "on-demand" component of this model allows consumers to adapt their IT usage to rapidly increasing or unpredictable computing needs.

A variety of applications can take advantage of on-demand services. Rather than prompting the emergence of new kinds of applications, cloud computing is capable of making existing classes of applications more compelling, thus revealing new opportunities, which in turn, drive the need for more sophisticated services [8].

Providers of IT services can achieve better operational costs; hardware and software infrastructures are built to provide multiple solutions and serve many users thus increasing efficiency and ultimately leading to faster Return On Investment (ROI) as well as lower Total Cost of Ownership (TCO) [40].

Indeed, the long-held dream of delivering computing as a utility has been realised with the advent of cloud computing [8]. However, over the years, several technologies have matured and significantly contributed to make cloud computing viable.

This chapter aims at presenting a comprehensive introduction to cloud computing, and the challenges and opportunities it brings to researchers. It also tracks the roots of cloud computing by surveying the main technological advancements that significantly contributed to the advent of this emerging field, with the objective of clarifying potential confusion over the "revolutionary or evolutionary" nature of cloud computing. In addition, this chapter presents various research works that are related to the contents of this thesis. Especially, it positions the thesis among those works highlighting differences and similarities.

The contents of this chapter are derived from a published book chapter:

Voorsluys, W., Broberg, J., and Buyya, R. Introduction to Cloud Computing. In *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. Goscinski, Eds. Wiley, 2011, ch. 1, pp. 3–42

2.1 Roots of Cloud Computing

We can track the roots of cloud computing by observing the advancement of several technologies, especially in hardware (virtualisation, multi-core chips), Internet technologies (Web Services, Service Oriented Architectures, Web 2.0), distributed computing (Clusters, Grids), and systems management (autonomic computing, data centre automation). Figure 2.1 shows the convergence of technology fields that significantly advanced and contributed to the advent of cloud computing.

Akin to cloud computing, some of these technologies have been tagged as hype in their early stages of development; however, they later received significant attention from academia and were sanctioned by major industry players. Consequently, a specification and standardisation process followed, leading to maturity and wide adoption. The emergence of cloud computing itself is closely linked to the maturity of such technologies. We present now a closer look at the technologies that form the base of cloud computing, with the aim of providing a clearer picture of the cloud ecosystem as a whole.

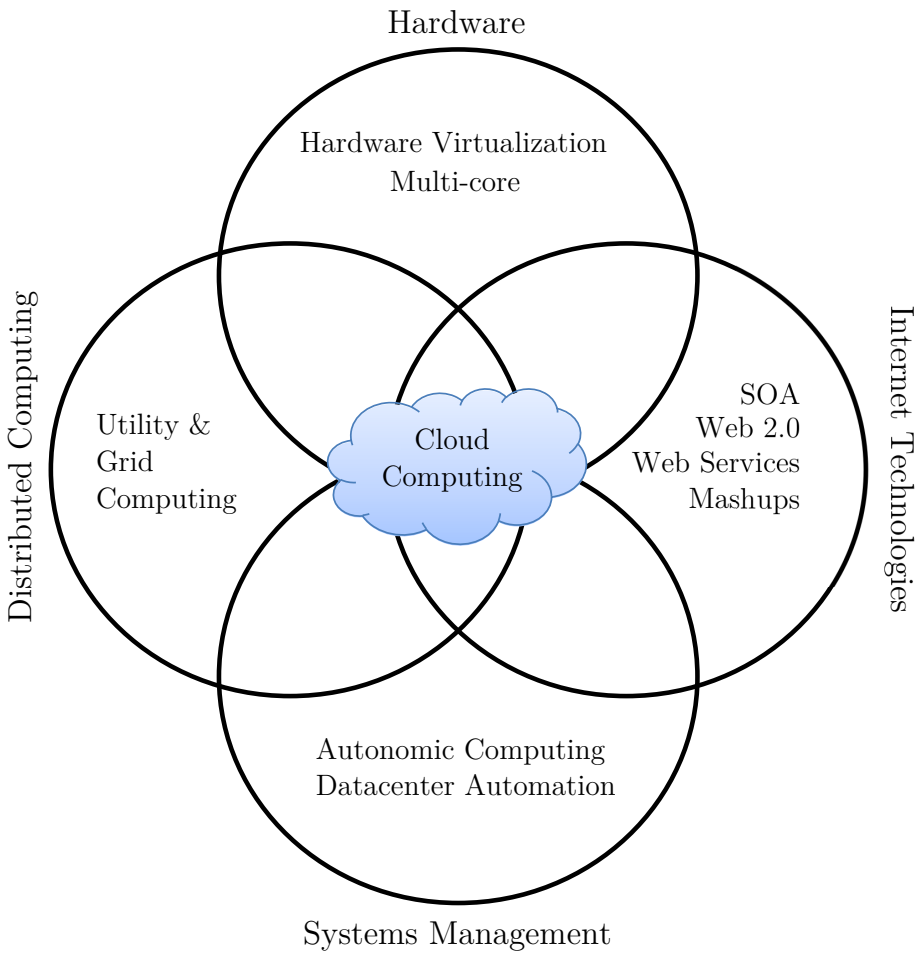


Figure 2.1: Convergence of various advances leading to the advent of cloud computing

2.1.1 From Mainframes to Clouds

Several technologies have in some way aimed at turning the utility computing concept into reality. In the 1970s, companies who offered common data processing tasks, such as payroll automation, operated time-shared mainframes as utilities, which could serve dozens of applications and often operated close to 100% of their capacity. In fact, mainframes had to operate at very high utilisation rates simply because they were very expensive and costs should be justified by efficient usage [19].

The mainframe era collapsed with the advent of fast and inexpensive microprocessors and IT data centres moved to collections of commodity servers. Apart from its clear advantages, this new model inevitably led to isolation of workload into dedicated servers, mainly due to incompatibilities between software stacks and op-

erating systems [40]. In addition, the unavailability of efficient computer networks meant IT infrastructure should be hosted in proximity to where it would be consumed. Altogether, these facts have prevented the utility computing reality of taking place on modern computer systems.

Similar to old electricity generation stations, which used to power individual factories, computing servers and desktop computers in a modern organisation are often underutilised, since IT infrastructure is configured to handle theoretical demand peaks.

In addition, in the early stages of electricity generation, electric current could not travel long distances without significant voltage losses. However, new paradigms emerged culminating on transmission systems able to make electricity available hundreds of kilometres far off from where it is generated. Likewise, the advent of increasingly fast fiber-optics networks has relit the fire and new technologies for enabling sharing of computing power over great distances have appeared.

These facts reveal the potential of delivering computing services with the speed and reliability that businesses enjoy with their local machines. The benefits of economies of scale and high utilisation allow providers to offer computing services for a fraction of what it costs for a typical company that generates its own computing power [19].

2.1.2 SOA, Web Services, Web 2.0, and Mashups

The emergence of Web Services (WS) open standards has significantly contributed to advances in the domain of software integration [77]. Web services can glue together applications running on different messaging product platforms, enabling information from one application to be made available to others, and enabling internal applications to be made available over the Internet.

Over the years a rich WS software stack has been specified and standardised resulting in a multitude of technologies to describe, compose and orchestrate services, package and transport messages between services, publish and discover services, represent QoS parameters, and ensure security in service access [59].

WS standards have been created on top of existing ubiquitous technologies such as HTTP and XML, thus providing a common mechanism for delivering services, making them ideal for implementing a Service Oriented Architecture (SOA). The

purpose of a SOA is to address requirements of loosely coupled, standards-based, and protocol-independent distributed computing. In a SOA, software resources are packaged as "services", which are well-defined, self-contained modules that provide standard business functionality and are independent of the state or context of other services. Services are described in a standard definition language and have a published interface [77].

The maturity of WS has enabled the creation of powerful services that can be accessed on-demand, in a uniform way. While some WS are published with the intent of serving end-user applications, their true power resides in its interface being accessible by other services. An enterprise application that follows the SOA paradigm is a collection of services that together perform complex business logic [77].

This concept of gluing services initially focused on the enterprise Web, but gained space in the consumer realm as well, especially with the advent of Web 2.0. In the consumer Web, information and services may be programmatically aggregated, acting as building blocks of complex compositions, called *service mashups*. Many service providers, such as Amazon, del.icio.us, Facebook, and Google make their service APIs publicly accessible using standard protocols such as SOAP and REST [59]. Consequently, one can put an idea of a fully functional Web application into practice just by gluing pieces with few lines of code.

In the Software as a Service domain, cloud applications can be built as compositions of other services from the same or different providers. Services such user authentication, e-mail, payroll management and calendars are examples of building-blocks that can be reused and combined in a business solution in case a single, ready-made system does not provide all those features. Many building blocks and solutions are now available in public marketplaces. For example, Programmable Web[1] is a public repository of service APIs and mashups currently listing thousands of APIs and mashups. Popular APIs such as Google Maps, Flickr, Youtube, Amazon eCommerce, and Twitter, when combined, produce a variety of interesting solutions, from finding video game retailers to weather maps. Similarly, Salesforce.com's offers AppExchange[2], which enables the sharing of solutions developed by third-party developers on top of salesforce.com components.

[1]http://www.programmableweb.com
[2]http://sites.force.com/appexchange

2.1.3 Grid Computing

Grid computing enables aggregation of distributed resources and transparently access to them. Most production Grids such as TeraGrid [21] and EGEE [41] seek to share compute and storage resources distributed across different administrative domains, with their main focus being speeding up a broad range of scientific applications, such as climate modeling, drug design, and protein analysis [13].

A key aspect of the Grid vision realisation has been building standard Web services-based protocols that allow distributed resources to be "discovered, accessed, allocated, monitored, accounted for, and billed for, etc. and in general managed as a single virtual system". The Open Grid Services Architecture (OGSA) addresses this need for standardisation by defining a set of core capabilities and behaviours that address key concerns in Grid systems [39].

Globus Toolkit [38] is a middleware that implements several standard Grid services and over the years has aided the deployment of several service-oriented Grid infrastructures and applications. An ecosystem of tools is available to interact with service Grids, including Grid brokers, which facilitate user interaction with multiple middleware and implement policies to meet Quality of Service (QoS) needs.

The development of standard protocols for several Grid computing activities has contributed – theoretically – to allow delivery of on-demand computing services over the Internet. However, ensuring QoS in Grids has been perceived as a difficult endeavour.

Lack of performance isolation has prevented Grids adoption in a variety of scenarios, especially on environments where resources are oversubscribed or users are uncooperative. Activities associated with one user or virtual organisation (VO) can influence, in an uncontrollable way, the performance perceived by other users using the same platform. Therefore, the impossibility of enforcing QoS and guaranteeing execution time became a problem, especially for time-critical applications [53].

Another issue that has lead to frustration when using Grids is the availability of resources with diverse software configurations, including disparate operating systems, libraries, compilers, runtime environments and so forth. At the same time, user applications would often run only on specially customised environments. Con-

sequently, a portability barrier has been often present on most Grid infrastructures, inhibiting users of adopting Grids as utility computing environments [53].

Virtualisation technology has been identified as the perfect fit to issues that have caused frustration when using Grids, such as hosting many dissimilar software applications on a single physical platform. In this direction, some research projects (e.g. Globus Virtual Workspaces [53]) aimed at evolving Grids to support an additional layer to virtualise computation, storage, and network resources.

2.1.4 Utility Computing

With increasing popularity and usage, large Grid installations have faced new problems, such as excessive spikes in demand for resources coupled with strategic and adversarial behaviour by users. Initially, Grid resource management techniques did not ensure fair and equitable access to resources in many systems. Traditional metrics (throughput, waiting time, and slowdown) failed to capture the more subtle requirements of users. There were no real incentives for users to be flexible about resource requirements or job deadlines, nor provisions to accommodate users with urgent work.

In utility computing environments, users assign a "utility" value to their jobs, where utility is a fixed or time-varying valuation that captures various QoS constraints (deadline, importance, satisfaction). The valuation is the amount they are willing to pay a service provider to satisfy their demands. The service providers then attempt to maximise their own utility, where said utility may directly correlate with their profit.

Providers can choose to prioritise high yield (i.e. profit per unit of resource) user jobs, leading to a scenario where shared systems are viewed as a marketplace, where users compete for resources based on the perceived utility or value of their jobs. Further information and comparison of these utility computing environments are available in an extensive survey of these platforms by Broberg et al. [13].

2.1.5 Hardware Virtualisation

Cloud computing services are usually backed by large-scale data centres composed of thousands of computers. Such data centres are built to serve many users and host many disparate applications. For this purpose, hardware virtualisation (also referred to as system virtualisation) can be considered as a perfect fit to overcome most

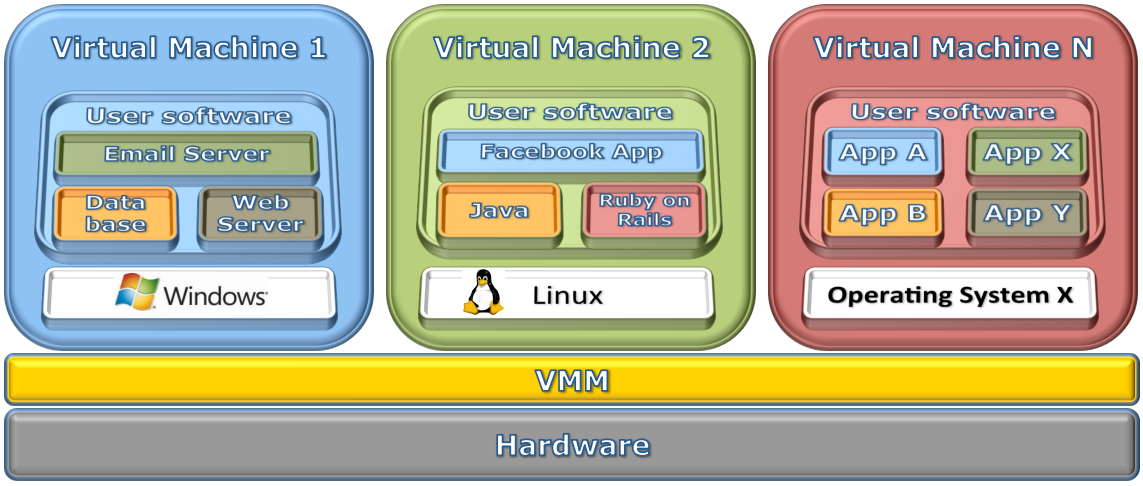


Figure 2.2: A hardware virtualised server hosting 3 virtual machines, each one running distinct operating system and user level software stack.

operational issues of data centre building and maintenance. The idea of virtualising a computer system's resources, including processors, memory, and I/O devices, has been well established for decades, aiming at improving sharing and utilisation of computer systems [44].

Hardware virtualisation allows running multiple operating systems and software stacks on a single physical platform. As depicted in Figure 2, a software layer, the Virtual Machine Monitor (VMM), also called a hypervisor, mediates access to the physical hardware presenting to each guest operating system a Virtual Machine (VM), which is a set of virtual platform interfaces [94].

The advent of several innovative technologies – multi-core chips, paravirtualisation, hardware-assisted virtualisation, and live migration of VMs – has contributed to an increasing adoption of virtualisation on server systems. Traditionally, perceived benefits were improvements on sharing and utilisation, better manageability and higher reliability.

More recently, with the adoption of virtualisation on a broad range of server and client systems, researchers and practitioners have been emphasising 3 basic capabilities regarding management of workload in a virtualised system, namely isolation, consolidation, and migration [11].

Workload isolation is achieved since all program instructions are fully confined inside a VM, which leads to improvements in security. Better reliability is also

achieved because software failures inside one VM do not affect others [94]. Moreover, better performance control is attained since execution of one VM should not affect the performance of another VM [11].

The consolidation of several individual and heterogeneous workloads onto a single physical platform leads to better system utilisation. This practice is also employed for overcoming potential software and hardware incompatibilities in case of upgrades, given that it is possible to run legacy and new operation systems concurrently [94].

Workload migration, also referred as application mobility [11], targets at facilitating hardware maintenance, load balancing, and disaster recovery. It is done by encapsulating a guest OS state within a VM and allowing it to be suspended, fully serialised, migrated to a different platform and resumed immediately or preserved to be restored at a later date [94]. A VM's state includes a full disk or partition image, configuration files and an image of its RAM [53].

A number of VMM platforms exist that are the basis of many utility or cloud computing environments. The most notable ones, VMWare, Xen and KVM, are out-lined in the following sections.

VMWare ESXi

VMware is a pioneer in the virtualisation market. Its ecosystem of tools ranges from server and desktop virtualisation to high-level management tools [98].

ESXi is a VMM from VMWare. It is a bare metal hypervisor, meaning that it installs directly on the physical server, whereas others may require a host operating system. It pro-vides advanced virtualisation techniques of processor, memory, and I/O. Especially, through memory ballooning and page sharing, it can over commit memory, thus increasing the density of VMs inside a single physical server.

Xen

The Xen hypervisor started as an open-source project and has served as a base to other virtualisation products, both commercial and open-source. It has pioneered the para-virtualisation concept, on which the guest operating system, by means of a specialised kernel, can interact with the hypervisor, thus significantly improving performance. In addition to an open-source distribution [11], Xen currently forms

the base of commercial hypervisors of a number of vendors, most notably Citrix XenServer and Oracle VM.

KVM

The Kernel-based Virtual Machine (KVM) is a Linux virtualisation subsystem. Is has been part of the mainline Linux kernel since version 2.6.20, thus being natively supported by several distributions. In addition, activities such as memory management and scheduling are carried out by existing Kernel features, thus making KVM simpler and smaller than hypervisors that take control of the entire machine [56].

KVM leverages hardware assisted virtualisation, which improves performance and allows it to support unmodified guest operating systems; currently, it supports several versions of Windows, Linux and UNIX.

2.1.6 Virtual Appliances and the Open Virtualisation Format

An application combined with the environment needed to run it (operating system, libraries, compilers, databases, application containers and so forth) is referred to as a "virtual appliance" [30]. Packaging application environments in the shape of virtual appliances eases software customisation, configuration and patching, and improves portability. Most commonly, an appliance is shaped as a VM disk image associated with hardware requirements, and can be readily deployed in a hypervisor.

On-line marketplaces have been set up to allow the exchange of ready-made appliances containing popular operating systems and useful software combinations, both commercial and open-source. Most notably, the VMWare Virtual Appliance Marketplace [99] allows users to deploy appliances on VMWare hypervisors or on partners public clouds, and Amazon allows developers to share specialised Amazon Machine Images (AMI) and monetize their usage on Amazon EC2 [95].

In a multitude of hypervisors, where each one supports a different VM image format and the formats are incompatible with one another, a great deal of interoperability issues arises. For instance, Amazon has its Amazon Machine Image (AMI) format, made popular on the Amazon EC2 public cloud.

In order to facilitate packing and distribution of software to be run on VMs several vendors, including VMware, IBM, Citrix, Cisco, Microsoft, Dell, and HP

have devised the Open Virtual Format (OVF). It aims at being "open, secure, portal, efficient and extensible" [31]. An OVF package consists of a file, or set of files, describing the VM hardware characteristics (e.g. memory, network cards, and disks), operating system details, startup, and shutdown actions, the virtual disks themselves and other metadata containing product and licensing information. OVF also supports complex packages composed of multiple VMs (e.g. multi-tier applications) [31].

OVF's extensibility has encouraged additions relevant to management of data centres and clouds. Mathews et al. [63] have devised Virtual Machine Contracts (VMC) as an extension to OVF. A VMC aids in communicating and managing the complex expectations that VMs have of their runtime environment and vice versa. A simple example of a VMC is when a cloud consumer wants to specify minimum and maximum amounts of a resource that a VM needs to function; similarly the cloud provider could express resource limits as a way to bound resource consumption and costs.

2.1.7 Autonomic Computing

The increasing complexity of computing systems has motivated research on autonomic computing, which seeks to improve systems by decreasing human involvement in their operation. In other words, systems should manage themselves, with high-level guidance from humans [48].

Autonomic, or self-managing, systems rely on monitoring probes and gauge Definitions s (sensors), on an adaptation engine (autonomic manager) for computing optimisations based on monitoring data, and on effectors, to carry out changes on the system. IBM's Autonomic Computing Initiative has contributed to define the four properties of autonomic systems: self-configuration, self-optimisation, self-healing, and self-protection. IBM has also suggested a reference model for autonomic control loops of autonomic managers, called MAPE-K (Monitor Analyze Plan Execute – Knowledge) [48, 47].

The large data centres of cloud computing providers must be managed in an efficient way. In this sense, the concepts of autonomic computing inspire software technologies for data centre automation, which may perform tasks such as: management of service levels of running applications, management of data centre capacity, proactive disaster recovery, and automation of VM provisioning.

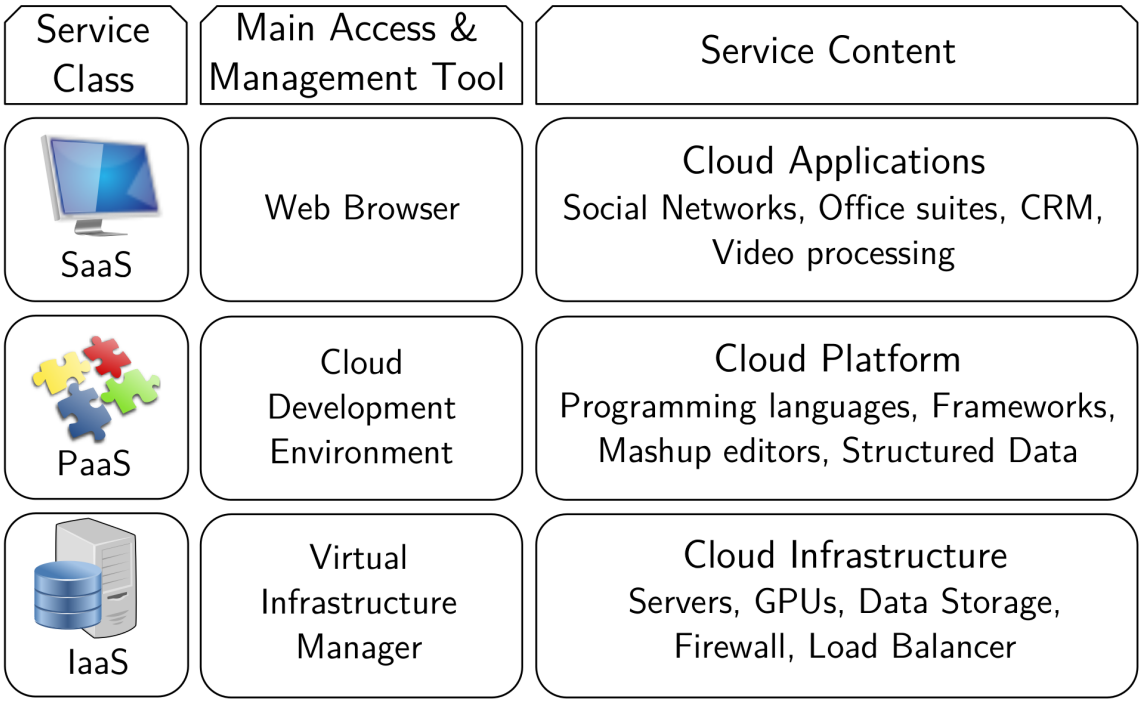


Figure 2.3: The cloud computing stack

2.2 Layers and Types of Clouds

Cloud computing services are divided into 3 classes, according to the abstraction level of the capability provided and the service model of providers, namely: (1) Infrastructure as a Service, (2) Platform as a Service and (3) Software as a Service [65]. Figure 2.3 depicts the layered organisation of the cloud stack from physical infrastructure to applications.

These abstraction levels can also be viewed as a layered architecture where services of a higher layer can be composed from services of the underlying layer [108]. The reference model of Buyya et al. [15] explains the role of each layer in an integrated architecture. A core middleware manages physical resources and the VMs deployed on top of them; in addition, it provides the required features (e.g. accounting and billing) to offer multi-tenant pay-as-you-go services.

Cloud development environments are built on top of infrastructure services to offer application development and deployment capabilities; in this level various programming models, libraries, APIs, and mashup editors enable the creation of a

range of business, Web, and scientific applications. Once deployed in the cloud, these applications can be consumed by end users.

Offering virtualised resources (computation, storage, and communication) on demand is known as *Infrastructure as a Service (IaaS)* [87]. A cloud infrastructure enables on-demand provisioning of servers running several choices of operating systems and a customised software stack. Infrastructure services are considered to be the bottom layer of cloud computing systems [71].

In addition to infrastructure-oriented clouds that provide raw computing and storage services, another approach is to offer a higher level of abstraction to make a cloud easily programmable, known as *Platform as a Service (PaaS)*. A cloud platform offers an environment on which developers create and deploy applications and do not necessarily need to know how many processors or how much memory that applications will be using. In addition, multiple programming models and specialised services (e.g. data access, authentication, and payments) are offered as building-blocks to new applications [7].

Applications reside on the top of the cloud stack. Services provided by this layer can be accessed by end-users through Web portals. Therefore, consumers are increasingly shifting from locally installed computer programs to on-line software services that offer the same functionally. Traditional desktop applications such as word processing and spreadsheet can now be accessed as a service in the Web. This model of delivering applications, known as *Software as a Service (SaaS)*, alleviates the burden of software maintenance for customers and simplifies development and testing for providers [108, 45].

2.2.1 Deployment Models

Although cloud computing has emerged mainly from the appearance of public computing utilities, other deployment models, with variations in physical location and distribution, have been adopted. In this sense, regardless of its service class, a cloud can be classified as public, private, community or hybrid [65].

Armbrust et al. propose definitions for *public cloud* as a "cloud made available in a pay-as-you-go manner to the general public" and *private cloud* as "internal data centre of a business or other organisation, not made available to the general public" [8].

In most cases, establishing a private cloud means restructuring an existing infrastructure by adding virtualisation and cloud-like interfaces. This allows users to interact with the local data centre while experiencing the same advantages of public clouds, most notably self-service interface, privileged access to virtual servers, and per-usage metering and billing.

A *community cloud* is "shared by several organisations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations) [65]."

A *hybrid cloud* takes shape when a private cloud is supplemented with computing capacity from public clouds [87]. The approach of temporarily renting capacity to handle spikes in load is known as "cloud-bursting" [50].

2.3 Desired Features of a Cloud

Certain features of a cloud are essential to enable services that truly represent the cloud computing model and satisfy expectations of consumers, cloud offerings must generally be: (i) self-service; (ii) per-usage metered and billed; (iii) elastic; and (iv) customisable.

2.3.1 Self-service

Consumers of cloud computing services expect on-demand, nearly instant access to resources. To support this expectation, clouds must allow self-service access so that customers can request, customise, pay and use services without intervention of human operators [65].

2.3.2 Per-usage Metering and Billing

Cloud computing eliminates up-front commitment by users, allowing them to request and use only the necessary amount. Services must be priced on a short-term basis (e.g. by the hour) allowing users to release (and not pay for) resources as soon as they are not needed [8]. For this reasons, clouds must implement features to allow efficient trading of service such as pricing, accounting and billing [17]. Metering should be done accordingly for different types of service (e.g., storage, processing and bandwidth) and usage promptly reported, thus providing greater transparency [65].

2.3.3 Elasticity

Cloud computing gives the illusion of infinite computing resources available on demand [8]. Therefore users expect clouds to rapidly provide resources in any quantity at any time. Especially, it is expected that the additional resources can be provisioned, possibly automatically, when an application load increases, and released when load decreases (scale up and down) [65].

2.3.4 Customisation

In a multi-tenant cloud a great disparity between user needs is often the case. Thus, resources rented from the cloud must be highly customisable. In the case of infrastructure services, customisation means allowing users to deploy specialised virtual appliances and to be given privileged (root) access to the virtual servers. Other service classes (PaaS and SaaS) offer less flexibility and not suitable for general purpose computing [8], but still are expected to provide a certain level of customisation.

2.4 Cloud Infrastructure Management

A key challenge IaaS providers face when building a cloud infrastructure is managing physical and virtual resources, namely servers, storage and networks, in a holistic fashion. The orchestration of resources must be performed in a way to rapidly and dynamically provision resources to applications [87].

The software toolkit responsible for this orchestration is called a virtual infrastructure manager (VIM) [87]. This type of software resembles a traditional operating system – but instead of dealing with a single computer, it aggregates resources from multiple computers, presenting a uniform view to user and applications. Other terms often used to refer to VIMs include "cloud operating system [101]," "infrastructure sharing software [73]," and "virtual infrastructure engine [28]."

Sotomayor et al. [87], in their description of the cloud ecosystem of software tools, propose a differentiation between two categories of tools used to manage clouds. The first category – cloud toolkits – includes those that "expose a remote and secure interface for creating, controlling and monitoring virtualised resources", but do not specialise in VI management. Tools in the second category – the virtual infrastructure managers – provide advanced features such as automatic load balancing and server

consolidation, but do not expose remote cloud-like interfaces. However, the authors point out that there is a superposition between the categories; cloud toolkits can also manage virtual infrastructures, although they usually provide less sophisticated features than specialised VIMs do.

The availability of a remote cloud-like interface and the ability of managing many users and their permissions are the primary features that would distinguish "cloud toolkits" from "VIMs". However, in this chapter, we place both categories of tools under the same group (of the VIMs) and when applicable, we highlight the availability of a remote interface as a feature.

Virtually all VIMs we investigated present a set of basic features related to managing the lifecycle of VMs, including networking groups of VMs together and setting up virtual disks for VMs. These basic features pretty much define whether a tool can be used in practical cloud deployments or not. On the other hand, only a handful of software present advanced features (e.g. high availability) which allow them to be used in large-scale production clouds.

2.4.1 Features

We now present a list of both basic and advanced features that are usually available in VIMs.

Virtualisation Support

The multi-tenancy aspect of clouds requires multiple customers with disparate requirements to be served by a single hardware infrastructure. virtualised resources (CPUs, memory, etc) can be size and resize with certain flexibility. These features make hardware virtualisation, the ideal technology to create a virtual infrastructure that partitions a data centre among multiple tenants.

Self-service, On-demand Resource Provisioning

Self-service access to resources has been perceived as one the most attractive features of clouds. This feature enables users to directly obtain services from clouds, such as spawning the creation of a server and tailoring its software, configurations and security policies, without interacting with a human system administrator.

That capability eliminates the need for more time-consuming, human-driven resource provisioning, once typical in IT. Therefore, exposing a self-service interface, through which users can easily interact with the system, is a highly desirable feature of a VI manager.

Multiple Backend Hypervisors

Different virtualisation models and tools offer different benefits, drawbacks and limitations. Thus, some VIMs provide a uniform management layer regardless of the virtualisation technology used. This characteristic is more visible in open-source VIMs, which usually provide pluggable drivers to interact with multiple hypervisors [87].

Also in this direction, the aim of libvirt [47] is to provide a uniform API that VIMs can use to manage domains (a VM or container running an instance of an operating system) in virtualised nodes using standard operations that abstract hypervisor specific calls.

Storage Virtualisation

Virtualising storage means abstracting logical storage from physical storage. By consolidating all available storage devices in a data centre, it allows creating virtual disks independent from device and location.

Storage devices are commonly organised in a storage area network (SAN) and attached to servers via protocols such as Fibre Channel, iSCSI and NFS; a storage controller provides the layer of abstraction between virtual and physical storage [82].

In the VI management sphere, storage virtualisation support is often restricted to commercial products of companies such as VMWare and Citrix. Other products feature ways of pooling and managing storage devices, but administrators are still aware of each individual device.

Interface to Public Clouds

Researchers have perceived that extending the capacity of a local in-house computing infrastructure by borrowing resources from public clouds is advantageous. In this fashion, institutions can make good use of their available resources and, in case of spikes in demand, extra load can be offloaded to rented resources [28].

A VI manager can be used in a hybrid cloud set up if it offers a driver to manage the lifecycle of virtualised resources obtained from external cloud providers. To the applications, the use of leases resources must ideally be transparent.

Virtual Networking

Virtual networks allow creating an isolated network on top of a physical infrastructure independently from physical topology and locations [78]. A virtual LAN (VLAN) allows isolating traffic that shares a switched network, allowing VMs to be grouped into the same broadcast domain.

Additionally, a VLAN can be configured to block traffic originated from VMs from other networks. Similarly, the VPN (virtual private network) concept is used to describe a secure and private overlay network on top of a public network (most commonly the public Internet) [90].

Support for creating and configuring virtual networks to group VMs placed throughout a data centre is provided by most VIMs. Additionally, VIMs that interface with public clouds often support secure VPNs connecting local and remote VMs.

Dynamic Resource Allocation

Increased awareness of energy consumption in data centres has encouraged the practice of dynamic consolidating VMs in a fewer number of servers. In cloud infrastructures, where applications have variable and dynamic needs, capacity management and demand prediction are especially complicated. This fact triggers the need for dynamic resource allocation aiming at obtaining a timely match of supply and demand [43].

Energy consumption reduction and better management of SLAs can be achieved by dynamically remapping VMs to physical machines at regular intervals. Machines that are not assigned any VM can be turned off or put on a low power state. In the same fashion, overheating can be avoided by moving load away from hotspots [97].

A number of VIMs include a dynamic resource allocation feature that continuously monitors utilisation across resource pools and reallocates available resources among VMs according to application needs.

Virtual Clusters

Several VIMs can holistically manage groups of VMs. This feature is useful for provisioning computing *virtual clusters on demand*, and interconnected VMs for multi-tier Internet applications [54].

Reservation and Negotiation Mechanism

When users request computational resources to available at a specific time, requests are termed advance reservations (AR), in contrast to best-effort requests, when users request resources whenever available [85]. To support complex requests, such as AR, a VI manager must allow users to "lease" resources expressing more complex terms (e.g. the period of time of a reservation). This is especially useful in clouds on which resources are scarce; since not all requests may be satisfied immediately they can benefit of VM placement strategies that support queues, priorities, and advance reservations [87].

Additionally, leases may be negotiated and renegotiated allowing provider and consumer to modify a lease or present counter proposals until an agreement is reached. This feature is illustrated by the case on which an AR request for a given slot cannot be satisfied but the provider can offer a distinct slot that is still satisfactory to the user. This problem has been addressed in OpenPEX, which incorporates a bilateral negotiation protocol that allows users and providers to come to an alternative agreement by exchanging offers and counter-offers [96].

High Availability and Data Recovery

The high availability (HA) feature of VIMs aims at minimising application downtime and preventing business disruption. A few VIMs accomplish this by providing a failover mechanism, which detects failure of both physical and virtual servers and restarts VMs on healthy physical servers. This style of HA protects from host, but not VM, failures [100, 25].

For mission critical applications, when a failover solution involving restarting VMs does not suffice, additional levels of fault tolerance that rely on redundancy of VMs are implemented. In this style, redundant and synchronised VMs (running or in stand by) are kept in a secondary physical server. The HA solution monitors failures of system components such as servers, VMs, disks and network, and ensures that a duplicate VM serves the application in case of failures [25].

Data backup in clouds should take into account the high data volume involved in VM management. Frequent backup of a large number of VMs, each one with multiple virtual disks attached, should be done with minimal interference in the systems performance. In this sense, some VIMs offer data protection mechanisms that perform incremental backups of VM images. The backup workload is often assigned to proxies, thus offloading production server and reducing network overhead [102].

2.4.2 Case Studies

In this section, we briefly describe the main features of the some of the most popular VI managers. Only the most prominent and distinguishing features of each tool are discussed in detail.

AppLogic

AppLogic [18] is a commercial VI manager, the flagship product of CA Technologies. It provides a fabric to manage clusters of virtualised servers, focusing on managing multi-tier Web applications. It views an entire application as collection of components (virtual appliances) that must be managed as a single entity. Several components such as firewalls, load balancers, Web servers, application servers and database servers can be set up and linked together.

Whenever the application is started, the system manufactures and assembles the virtual infrastructure required to run it. Once the application is stopped, AppLogic tears down the infrastructure built for it.

AppLogic offers dynamic appliances to add functionality such as Disaster Recovery and Power optimisation to applications. The key differential of this approach is that additional functionalities are implemented as another pluggable appliance instead of being added as a core functionality of the VI manager.

Apache CloudStack

The CloudStack [4] suite is one the most feature complete VI management software available, focusing on management and automation of data centres, enabling the orchestration of most resources required to build a IaaS cloud. It is essentially a hypervisor-agnostic solution, supporting VMware, Oracle VM, KVM, XenServer and Xen Cloud Platform.

By providing several access interfaces, it facilitates both human and programmatic interaction with the controller. Automation of tasks is also aided by a workflow orchestration mechanism.

Eucalyptus

The Eucalyptus[71] framework was one of the first open-source projects to focus on building IaaS clouds. It has been developed with the intent of providing an open-source implementation nearly identical in functionality to Amazon Web Services APIs.

Users can interact with a Eucalyptus cloud using the same tools they use to access Amazon EC2. It also provides a storage cloud API – emulating the Amazon S3 API – for storing general user data and VM images.

Nimbus Infrastructure

The Nimbus [52] suite provides most features in common with other open-source VI managers, such as an EC2-compatible front-end API, support for Xen and KVM, and a backend interface to Amazon EC2. Nimbus' core was engineered around the Spring framework to be easily extensible, thus allowing several internal components to be replaced and also eases the integration with other systems.

OpenNebula

OpenNebula [69] is an open-source VI managers initially conceived to manage local virtual infrastructure, but has also included remote interfaces that make it viable to build public clouds. Several programming APIs are available: XML-RPC and libvirt for local interaction; a subset of EC2 (Query) APIs and the OpenNebula Cloud API (OCA) for public access.

Its architecture is modular, encompassing several specialised pluggable components. The Core module orchestrates physical servers and their hypervisors, storage nodes and network fabric. Management operations are performed through pluggable Drivers, which interact with APIs of hypervisors, storage and network technologies and public clouds. The Scheduler module, which is in charge of assigning pending VM requests to physical hosts, offers dynamic resource allocation features.

Administrators can choose between different scheduling objectives such as packing VMs in fewer hosts or keeping the load balanced. Via integration with

the Haizea lease scheduler [87], OpenNebula also supports advance reservation of capacity and queuing of best-effort leases.

oVirt

oVirt [81] is another open-source VI manager, started under sponsorship of Red Hat's Emergent Technology group. It provides most of the basic features of other VI managers, including support for managing physical server pools, storage pools, user accounts and VMs. All features are accessible through a Web interface. The oVirt admin node, which is also a VM, provides a Web Server, secure authentication services based on freeIPA, and provisioning services to manage VM image and their transfer to the managed nodes. Each managed node libvirt, which inter-faces with the hypervisor.

OpenStack

OpenStack is a project initiated by Rackspace Hosting and NASA with the aim of defining standards and building open-source cloud management software. It comprises of a series of sub-projects, most notably three core projects: OPENSTACK COMPUTE, for large-scale deployments of automatically provisioned virtual compute instances; OPENSTACK OBJECT STORAGE: for large-scale, redundant storage of static objects; and OPENSTACK IMAGE SERVICE: provides discovery, registration, and delivery services for virtual disk images [72].

VMWare vSphere and vCloud

vSphere is VMware's suite of tools aimed at transforming IT infrastructures into private clouds. It distinguishes from other VI managers as one the most feature rich, due to the company's several offerings in all levels the architecture.

In the vSphere architecture, servers run on the ESXi platform. A separate server runs vCenter Server, which centralises control over the entire virtual infrastructure. Through the vSphere Client software, administrators connect to vCenter Server to perform various tasks.

The Distributed Resource Scheduler (DRS) makes allocation decisions based on pre-defined rules and policies. It continuously monitors the amount of resources available to VMs and, if necessary, makes allocation changes to meet VM requirements. In the storage virtualisation realm, vStorage VMFS is a cluster filesystem

to provide aggregate several disks in a single volume. VMFS is especially optimised to store VM images and virtual disks. It supports storage equipment that use Fibre Channel or iSCSI SAN.

In its basic set-up, vSphere is essentially a private administration suite. Self-service VM provisioning to end-users is provided via the vCloud API, which interfaces with vCenter Server. In this configuration, vSphere can be used by service providers to build public clouds. In terms of interfacing with public clouds, vSphere inter-faces with the vCloud API, thus enabling cloud-bursting into external clouds.

2.5 Infrastructure as a Service

Public Infrastructure as a Service providers commonly offer virtual servers containing one or more CPUs, running several choices of operating systems and a customised software stack. In addition, storage space and communication facilities are often provided.

2.5.1 Features

In spite of being based on a common set of features, IaaS offerings can be distinguished by the availability of special features that influence the cost-benefit ratio to be experienced by user applications when moved to the cloud. The most relevant features are:

1. Geographic distribution of data centres;

2. Variety of user interfaces and APIs to access the system;

3. Specialised components and services that aid particular applications, e.g. load-balancers, firewalls;

4. Choice of virtualisation platform and operating systems;

5. Different billing methods and period (e.g. pre-paid vs. post-paid, hourly vs. monthly).

Geographic Presence

To improve availability and responsiveness, a provider of worldwide services would typically build several data centres distributed around the world. For example, Amazon Web Services presents the concept of "Availability Zones" and "Regions" for its EC2 service. Availability Zones are "distinct locations that are engineered to be insulated from failures in other Availability Zones and provide inexpensive, low latency network connectivity to other Availability Zones in the same Region". Regions, in turn, "are geographically dispersed, and will be in separate geographic areas or countries [95]."

User Interfaces and Access to Servers

Ideally, a public IaaS provider must provide multiple access means to its cloud, thus catering for various users and their preferences. Different types of user interfaces (UI) provide different levels of abstraction, the most common being Graphical User Interfaces (GUI), command-line tools (CLI) and Web Service (WS) APIs.

GUIs are preferred by end-users who need to launch, customise and monitor a few virtual servers and do not necessary need to repeat the process several times. On the other hand, CLIs offer more flexibility and the possibility of automating repetitive tasks via scripts (e.g. start and shutdown a number of virtual servers at regular intervals). WS APIs offer programmatic access to a cloud using standard HTTP requests, thus allowing complex services to be built on top of IaaS clouds.

Advance Reservation of Capacity

Advance reservations allow users to request for an IaaS provider to reserve resources for a specific time frame in the future, thus ensuring that cloud resources will be available at that time. However, most clouds only support best-effort requests, i.e. users requests are server whenever resources are available [85].

Amazon Reserved Instances is a form of advance reservation of capacity, allowing users to pay a fixed amount of money in advance to guarantee resource availability at anytime during an agreed period and then paying a discounted hourly rate when resources are in use. However, only long periods of 1 to 3 years are offered; therefore, users cannot express their reservations in finer granularities, e.g. hours or days.

Automatic Scaling and Load Balancing

As mentioned earlier in this chapter, elasticity is a key characteristic of the cloud computing model. Applications often need to scale up and down to meet varying load conditions. Automatic scaling is a highly desirable feature of IaaS clouds. They allow users to set conditions for when they want their applications to scale up and down, based on application specific metrics such as transactions per second, number of simultaneous users, request latency and so forth.

When the number of virtual servers is increased by automatic scaling, incoming traffic must be automatically distributed among the available servers. This activity enables applications to promptly respond to traffic increase, while also achieving greater fault tolerance.

Service Level Agreement

Service Level Agreements (SLAs) are offered by IaaS providers to express their commitment to delivery of a certain QoS. To customers it serves as a warranty. An SLA usually includes availability and performance guarantees. Additionally, metrics must be agreed upon by all parties as well as penalties for violating these expectations.

Most IaaS providers focus their SLA terms on availability guarantees, specifying the minimum percentage of time the system will be available during a certain period. For instance, Amazon EC2 states that "if the Annual Uptime Percentage for a customer drops below 99.95% for the Service Year, that customer is eligible to receive a Service Credit equal to 10% of their bill.[2]"

Hypervisor and operating system choice

Traditionally, IaaS offerings have been based on heavily customised open-source Xen deployments. IaaS providers needed expertise in Linux, networking, virtualisation, metering, resource management and many other low level aspects to successfully deploy and maintain their cloud offerings. More recently, there has been an emergence of turn-key IaaS platforms such as VMWare vCloud and Citrix Cloud centre (C3) which have lowered the barrier of entry for IaaS competitors, leading to a rapid expansion in the IaaS marketplace.

2.5.2 Case Studies

In this section, we describe the main features of popular and pioneering public IaaS clouds. Only the most prominent and distinguishing features of each one are discussed in detail.

Amazon Web Services

Amazon Web Services (AWS) is one of the major players in the cloud computing market. It has pioneered the introduction of IaaS clouds in 2006. It offers a variety cloud services, most notably: S3 (storage), EC2 (virtual servers), Cloudfront (content delivery), Cloudfront Streaming (video streaming), Dynamo (structured NoSQL datastore), RDS (Hosted Relational Database), persistent disks for EC2 (Elastic Block Storage) SQS (reliable messaging) and Elastic MapReduce (data processing).

The Elastic Compute Cloud (EC2) offers Xen-based virtual servers (instances) that can be instantiated from Amazon Machine Images (AMI) hosted on S3 or from EBS volumes. Instances are available in a variety of sizes, operating systems, architectures and price.

CPU capacity of instances is measured in Amazon Compute Units and, although fixed for each instance, vary among instance types, e.g 1 ECU (small instance) and 20 ECUs (high CPU instance). Each instance provides a certain amount of non-persistent disk space; the persistence disk service (EBS) allows attaching virtual disks to instances with considerably higher amount of disk space.

Elasticity can be achieved by combining the CloudWatch, Auto Scaling and Elastic Load Balancing features, which allow the number of instances to scale up and down automatically based on a set of customisable rules, and traffic to be distributed across available instances. Fixed IP address (Elastic IPs) are not available by default, but can be obtained at an additional cost.

Flexiscale

Flexiscale is a UK-based provider offering services similar in nature to Amazon Web Services. However, its virtual servers offer some distinct features, most notably: persistent storage by default, fixed IP addresses, dedicated VLAN, a wider range of server sizes and runtime adjustment of CPU capacity (termed CPU bursting/vertical scaling). Similar to other providers, this service is also priced by the hour.

Joyent

Joyent's Public Cloud offers servers based on Solaris containers virtualisation technology. These servers, dubbed Accelerators, allow deploying various specialised software-stack based on a customised version of OpenSolaris operating system, which include by default a web-based configuration tool and several pre-installed software, such as Apache, MySQL, PHP, Ruby on Rails and Java. Software load balancing is available as an Accelerator in addition to hardware load balancers.

A notable feature of Joyent's virtual servers is automatic vertical scaling of CPU cores, which means a virtual server can make use of additional CPUs automatically up to the maximum number of cores available in the physical host.

GoGrid

GoGrid, like many other IaaS providers, allows its customers to utilise a range of pre-made Windows and Linux images, in a range of fixed instance sizes. GoGrid also offers "value-added" stacks on top for applications such as high-volume web serving, e-Commerce and database stores.

It offers some notable features, such as a "hybrid hosting" facility, which combines traditional dedicated hosts with auto-scaling cloud server infrastructure. In this approach, users can take advantage of dedicated hosting (which may be required due to specific performance, security or legal compliance reasons) and combine it with on-demand cloud infrastructure as appropriate, taking the benefits of each style of computing. As part of its core IaaS offerings, GoGrid also providers free hardware load balancing, auto-scaling capabilities and persistent storage, features which typically an additional cost on most other IaaS providers.

Rackspace Cloud Servers

Rackspace Cloud Servers is an IaaS solution that provides fixed size instances in the cloud. Cloud Servers offers a range of Linux-based pre-made images. A user can request different sized images, where the size is measured by requested RAM, not CPU.

Like GoGrid, Cloud Servers also offers hybrid approach where dedicated and cloud server infrastructures can be combined to take the best aspects of both styles of hosting as required. Cloud Servers, as part of its default offering, enables fixed (static) IP addresses, persistent storage and load balancing (via A-DNS) at no additional cost.

2.6 Challenges and Risks

Despite the initial success and popularity of the cloud computing paradigm and the extensive availability of providers and tools, a significant number of challenges and risks are inherent to this new model of computing. Providers, developers and end-users must consider these challenges and risks to take good advantage of cloud computing. Issues to be faced include user privacy, data security, data lock-in, availability of service, disaster recovery, performance, scalability, energy-efficiency, and programmability.

2.6.1 Security, Privacy and Trust

Ambrust et al. [8] cite information security as a main issue as "current cloud offerings are essentially public ... exposing the system to more attacks." For this reason there are potentially additional challenges to make cloud computing environments as secure as in-house IT systems. At the same time, existing, well-understood technologies can be leveraged, such as data encryption, VLANs and firewalls. Security and privacy affect the entire cloud computing stack, since there is a massive use of third party services and infrastructures that are used to host important data or to perform critical operations. In this scenario, the trust towards providers is fundamental to ensure the desired level of privacy for applications hosted in the Cloud [15].

Legal and regulatory issues also need attention. When data is moved into the Cloud, providers may choose to locate them anywhere on the planet. The physical location of data centres determines the set of laws that can be applied to the management of data. For example, specific cryptography techniques could not be used because they are not allowed in some countries. Similarly, country laws can impose that sensitive data, such as patient health records, is to be stored within national borders.

2.6.2 Data lock-in and standardisation

A major concern of cloud computing users is about having their data locked-in by a certain provider. Users may want to move data and applications out from a provider that does not meet their requirements. However, in their current form, cloud computing infrastructures and platforms do not employ standard methods of storing user

data and applications. Consequently, they do not interoperate and user data is not portable. The answer to this concern is standardisation. In this direction, there are efforts to create open standards for cloud computing.

In the hardware virtualisation sphere, the Open Virtual Format (OVF) aims at facilitating packing and distribution of software to be run on VMs so that virtual appliances can be made portable, i.e. seamlessly run on hypervisor of different vendors.

2.6.3 Availability, fault-tolerance and disaster recovery

It is expected that users will have certain expectations about the service level to be provided once their applications are moved to the Cloud. These expectations include availability of the service, its overall performance, as well as what measures are to be taken when something goes wrong in the system or its components. In summary, users seek for a warranty before they can comfortably move their business to the Cloud.

SLAs, which include QoS requirements, must be ideally set up between customers and cloud computing providers to act as warranty. An SLA specifies the details of the service to be provided, including availability and performance guarantees. Additionally, metrics must be agreed upon by all parties as well as penalties for violating the expectations.

2.6.4 Resource management and energy-efficiency

One important challenge faced by providers of cloud computing services is the efficient management of virtualised resource pools. Physical resources such as CPU cores, disk space and network bandwidth must be sliced and shared among virtual machines running potentially heterogeneous workloads.

The multi-dimensional nature of virtual machines complicates the activity of finding a good mapping of VMs onto available physical hosts while maximising user utility. Dimensions to be considered include: number of CPUs, amount of memory, size of virtual disks and network bandwidth. Dynamic VM mapping policies may leverage the ability to suspend, migrate and resume VMs as an easy way of preempting low priority allocations in favour of higher priority ones.

Migration of VMs brings additional challenges that need to be solved by mapping policies, such as detecting when to initiate a migration, which VM to migrate, and where to migrate. In addition, policies may take advantage of live migration of virtual machines to relocate data centre load without significantly disrupting running services. In this case, an additional concern is the trade-off between the negative impact of a live migration on the performance and stability of a service and the benefits to be achieved with that migration [104].

Another challenge concerns the outstanding amount of data to be managed in various VM management activities. Such data amount is a result of particular abilities of virtual machines, including the ability of travelling through space (i.e. migration) and time (i.e. checkpointing and rewinding) [66], operations that may be required in load balancing, backup and recovery scenarios. In addition, dynamic provisioning of new VMs and replicating existing VMs require efficient mechanisms to make VM block storage devices (e.g. image files) quickly available at selected hosts.

Data centres consumer large amounts of electricity. According to a data published by HP [36], 100 server racks can consume 1.3MW of power and another 1.3MW are required by the cooling system, thus costing USD 2.6 million per year. Besides the monetary cost, data centres significantly impact the environment in terms of CO2 emissions from the cooling systems [97].

In addition to optimise application performance, dynamic resource management can also improve utilisation and consequently minimise energy consumption in data centres. This can be done by judiciously consolidating workload onto smaller number of servers and turning off idle resources.

2.7 Related Work

2.7.1 Virtual Clusters: Building high-performance clusters using spot instances

Some researchers have proposed techniques to extend the capacity of in-house clusters at times of the peak demand using them to run compute-intensive applications. Assuncao et al. [28] have evaluated a set of well known scheduling policies, including backfilling techniques, in a system that extends the capacity of a local cluster

using fixed-priced (on-demand) cloud resources. Similarly, Mattess et al. [62] have evaluated policies that offload extra demand from a local cluster to a resource pool composed by Amazon EC2 spot instances. In contrast to these works, our provisioning policies do not consider the existence of a local cluster, instead, we assume all provisioned resources are cloud-based spot instances.

2.7.2 Variable pricing cloud resources: Usage and analyses

A few recently published works have touched the subject of using variable pricing cloud resources in high performance computing.

Danak and Mannor [27] present a uniform-price auction for resource allocation that is suitable for the dynamic nature of grid computing systems. They study the problem from the provider's point of view. Providers are able to adjust market capacity based on demand level, which helps them maximise profit. In their model, service providers offer a supply schedule to the Service Level Agreement (SLA) manager in which the provider describes its preferred amount of CPU time to sell at each unit price. After receiving user bids, the SLA manager declares the unit price at which the transactions must take place. Then, the provider adjusts supply, indicating the actual amount of CPU time it is willing to share.

Andrzejak et al. [3] have proposed a probabilistic decision model to help users decide how much to bid for a certain spot instance type in order to meet a certain monetary budget or a deadline. The model suggests bid values based on the probability of failures calculated using a mean of past prices from Amazon EC2. It can then estimate, with a given confidence, values for a budget and a deadline that can be achieved if the given bid is used.

Yi et al. [107] proposed a method to reduce costs of computations and providing fault-tolerance when using EC2 spot instances. Based on the price history, they simulated how several checkpointing policies would perform when faced with out-of-bid situations. The proposed policies used two distinct techniques for deciding when to checkpoint a running program: at hour boundaries and at price rising edges. In the hour boundary scheme, checkpoints are taken periodically every hour, while in the rising edge scheme, checkpoints are taken when the spot price for a given instance type is increasing. The authors proposed combinations of the above mentioned schemes, including adaptive decisions, such as taking or skipping check-

pointing at certain times. Their evaluation has shown that checkpointing schemes, in spite of the inherent overhead, can tolerate instance failures while reducing the price paid, as compared to normal on-demand instances. Similarly, we evaluate a checkpointing mechanism implemented according to this work, with the objective of comparing with other fault tolerance approaches.

Ben-Yehuda et al. [12] examined price history of the Amazon EC2 spot market through a reverse engineering process to identify how Amazon set prices in the market. Their research postulates that the auction mechanism used by Amazon is not completely driven by supply and demand, implying that Amazon sometimes uses a hidden reserve price. The evidence provided suggests that spot prices are usually drawn from a tight, fixed price interval, reflecting a random reserve price. In conclusion, they affirm that current published spot prices reveal little information about actual bids by Amazon clients and should not be used to model client behaviours.

Spot instances have been used by Chohan et al. [23] to accelerate Map-Reduce processes. The typical lower price of spot instances was shown to reduce monetary cost of running jobs. In spite of the benefits of using spot instances, the study shows that the high failure rate of instances can significantly increase overall completion time. Accordingly, they propose bidding strategies tailored for Map-Reduce jobs running on spot instances.

Mihailescu and Teo [67] considered the spot market pricing model of Amazon EC2 as a use case in a federated cloud environment. They argue that spot pricing used by Amazon is a truthful mechanism only in a market with a single provider and show that a rational user can increase their benefit by being untruthful in a federated cloud environment.

Song et al. [84] propose a bidding strategy for a cloud service broker utilising spot instances. Their brokering strategy reduces computational cost while not requiring a priori statistical information on the job sizes submitted by cloud users. Their method aims at maximising the broker's profit while charging end-users a reasonable rate.

2.7.3 Workload mobility and live migration of virtual machines

The impact of virtualisation in a variety of scenarios has been the focus of considerable attention. A number of studies have presented individual and side by side measurements of VM runtime overhead imposed by hypervisors on a variety of workloads [11, 22].

Apparao et al. [6] present a study on the impact of consolidating several applications on a single server running Xen. As workload the authors employed the vConsolidate benchmark [20] defined by Intel, which consists of a Web server VM, a database server VM, a Java server VM and mail server VM. An idle VM is also added to comply with real world scenarios, on which servers are hardly fully utilised.

The studies presented by Zhao & Figueiredo [110] and Clark et al. [26] specifically deal with VM migration. The former analyses performance degradation when migrating CPU and memory intensive workloads as well as migrating multiple VMs at the same time; however such study employs a pure stop-and-copy migration approach rather than live migration. The later introduces Xen live migration and quantifies its effects on a set of four applications common to hosting environments, primarily focusing on quantifying downtime and total migration time and demonstrating the viability of live migration. However, these works have not evaluated the effect of migration in the performance of modern Internet workloads, such as multi-tier and social network oriented applications.

A few studies propose and evaluate the efficacy of migrating VMs across long distances, such as over the Internet. For instance, Travostino et al. [91] have demonstrated the effectiveness of VM live migration over an WAN connected by dedicated 1Gbps links; application downtime has been quantified at 5-10 times greater than that experienced on an intra-LAN set-up, despite a 1000 times higher RTT. Besides its feasibility, the concept of WAN live migration is still to be implemented in commercial hypervisors, which demands all involved machines to be in the same subnet and share storage. Our work focuses only on migrating VMs within a data center.

The Cloudstone benchmark [83] aims at computing the monetary cost, in dollars/user/month, for hosting Web 2.0 applications in cloud computing platforms such as Amazon EC2. From this work we borrow the idea of using Olio [5] and Faban [89]

to compose our target workload for Web 2.0 applications. However, Cloudstone does not define a procedure to evaluate the cost of virtual machine migration and, to the best of our knowledge, no previous work has considered using this type of workload in migration experiments.

Chapter 3

Cost-effective Provisioning of Spot Market Resources

This chapter describes a resource management and scheduling policy — SpotRMS — that addresses the problem of running deadline-constrained computational jobs on a pool of resources composed solely of spot instances. SpotRMS relies on variations in price and performance of spot instances to run applications in a fast and economic way.

The purpose of SpotRMS, and the system architecture of which it is a part of, is to allow organisations that do not have a local cluster of resources to run batches of computational jobs on dynamically provisioned resources.

The policy relies on job runtime estimations to decide both the best types of instances to run each job, and when jobs should be run so that their deadlines can be met. Several estimation methods are evaluated and compared, using trace-based simulations, which take real price variation traces obtained from Amazon Web Services as input, as well as an application trace from the Parallel Workload Archive.

Results demonstrate the possibility and effectiveness of running computational jobs solely on spot instances, at a fraction (up to 60% lower) of the price that they would normally cost on fixed priced resources.

Specifically, this chapter introduces:

- A novel architecture that defines a resource management system that enables the creation of cloud-based virtual clusters solely utilising low-cost spot instances;
- A formal statement of the problem of running deadline-constrained jobs on spot instances;
- A resource provisioning policy that decides when to request spot instances to accommodate incoming jobs, which instance type to request, and when to start jobs on available instances;
- An information mechanism that aids decision-making by providing runtime estimates;

The contents of this chapter are derived from a published research paper:

Voorsluys, W., Garg, S. K., and Buyya, R. Provisioning Spot Market Cloud Resources to Create Cost-effective Virtual Clusters. In *Proceedings of the 11th IEEE International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-11)* (Melbourne, Australia, 2011), Y. Xiang, A. Cuzzocrea, M. Hobbs, and W. Zhou, Eds., vol. 7016 of *Lecture Notes in Computer Science*, Springer, pp. 395–408

3.1 Background and Motivation

The introduction of affordable cloud computing infrastructures has had a major impact in the business IT community. Cloud infrastructures are also being explored as a means of accomplishing high performance processing tasks, often present in areas such as science and finance.

However, the use of virtualisation and network shaping have been cited as factors that hinder the viability of cloud computing resources for running compute intensive applications, as opposed to using a dedicated HPC cluster [46]. Nonetheless, the potential cost savings offered by clouds has led to an increased adoption of cloud-based virtual clusters, as well as to the practice of extending the capacity of local clusters using cloud resources in situations of peak demand [28].

The cost to build these virtual clusters depends on the type of leased VMs. These are often offered via various pricing schemes through separate “markets”, most

notably: the on-demand market, which offers standard VMs at a fixed cost; and the spot market, which offers unused capacity in the form of variable price VMs. For example, Amazon EC2 offers VMs for lease (spot instances), for as low as $\frac{1}{3}$ of the standard fixed price as a similar instance. In this fashion, users submit a request that specifies a maximum price (bid) they are willing to pay per hour and instances associated to that request will run for as long as the current spot price is lower than the specified bid.

In spite of the apparent economic advantage, an intermittent nature is inherent to biddable resources, which may cause instance unavailability. When an out-of-bid situation occurs, i.e. the current spot price goes above the user's maximum bid, spot instances are terminated by the provider without prior notice to the end user. Such situations may lead to prolonged application execution times or to the inability of applications to finish within a specified deadline. However, this situation can be avoided with slightly higher bids, thus mitigating this uncertainty, or by using fault-tolerance techniques such as checkpointing [107].

Virtual clusters can be heterogeneous, e.g. when different types of instances with distinct number of CPU cores are leased and added to the resource pool. In this case, the ratio between price and performance of different types of spot instances may not be constant over time, creating opportunities for optimisations. For example, one could decide to run a certain compute intensive job by observing the performance per dollar ratio of two high-end CPU EC2 spot instances. The "c1.medium" instance type has a CPU power of 5 ECUs (EC2 Compute Units) and the "c1.xlarge" type has a power of 20 ECUs (one ECU is defined as equivalent to the power of a 1.0-1.2 GHz 2007 AMD Opteron or 2007 Intel Xeon processor.)

As the spot price of each instance type varies, the performance per dollar ratio offered by each instance type varies accordingly so that, at different periods of time, one instance type may offer a better ratio than the other. In these situations, if an application that would normally run using 5 ECUs could provide enough parallelism, it could run significantly faster on the relatively cheaper 20 ECU instance.

That approach offers extra flexibility to users since they may choose to assemble an instance pool by bidding on resource types that are currently at a discounted hourly price and then adapt their jobs to run more efficiently on the new resources.

Figure 3.1 depicts an example scenario of such a virtual cluster composed of instances of different sizes.

Indeed, new programming paradigms for concurrent applications, such as Map-Reduce and Dryad, were designed from the ground up based on the assumption that "future advances in local computing power will come from increasing the number of cores on a chip rather than improving the speed of a single core" [49]. Therefore, it is reasonable to assume that many applications will be able to efficiently utilise resource pools with a varying number of cores per processor. For example, if a user job can scale up and down to run with more or fewer CPU cores, the user would most likely choose the resources that offer the highest core per dollar ratio. In this sense, this work considers the scenario where computational jobs are executed on a virtual cluster platform style, i.e., a high-performance computing platform dynamically assembled out of inexpensive cloud-based resources. Nonetheless, legacy applications that are only capable of taking advantage of a single processing unit can still efficiently be run in cloud computing instances equipped with a single core, which are often available at the lowest prices by cloud computing providers.

The techniques presented here focus on reducing the costs of building virtual clusters by leasing spot market resources. Specifically, this chapter proposes a **resource provisioning and job scheduling policy that addresses the problem of dynamically building a virtual cluster out of low-cost VMs and utilising them to run compute-intensive applications**. The main objective of devising such a policy is exploiting variations in price and performance of resources to run applications in a more economical manner when compared to using fixed price resources.

Target user applications are often deadline-constrained. For this reason, Spot-RMS is augmented by a job runtime estimation mechanism, that aids in deciding how to allocate jobs in a way that they finish within their identified deadlines.

The results of this work demonstrate that it is possible to run batches of computational jobs by solely utilising spot instances, an approach that provides significant cost savings when compared to using fixed priced instances. Chiefly, these results quantify the effect of the accuracy of runtime estimation on the monetary cost. It is shown that less accurate estimations usually lead to higher costs (in the case of over estimations) or more missed deadlines (in the case of under estimations).

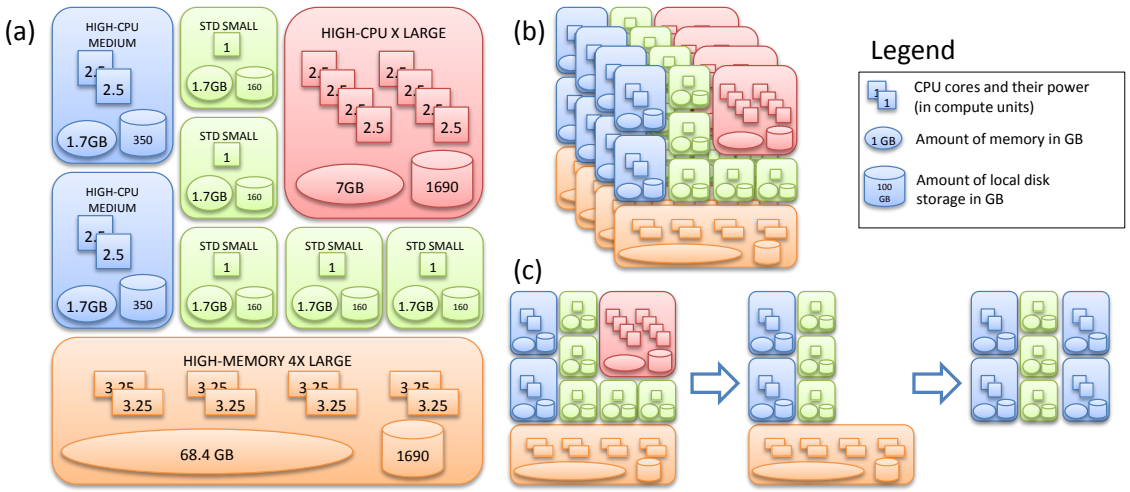


Figure 3.1: A virtual cluster dynamically assembled out of inexpensive cloud-based resources, e.g. Amazon EC2 spot instances. (a) VMs of appropriate sizes have to be chosen, depending on the immediate requirements of running applications; (b) the cluster is able to scale to much larger capacities; (c) VMs are replaced by different types of VMs as application requirements change

In some exceptional cases, less precise runtime estimated performed as well as more precise ones.

3.2 System Model of a Cloud-based Virtual Cluster

Considering a scenario where an organisation is interested in building a dynamic cluster using cloud resources, it can be assumed that resources are to be leased exclusively from one cloud provider, such as Amazon EC2.

Resources are spot instances, instantiated according to a previously configured template, which defines the capacity required (or "instance type") as well as the operating system and software stack details (i.e., Amazon Machine Image). A job scheduling and middleware system, called Broker, is responsible for receiving job requests from users, creating a suitable instance pool, and managing the QoS of jobs, i.e. ensuring that jobs finish within given deadlines.

Jobs are embarrassingly parallel groups of tasks submitted by local users of the organisation. A job request must contain information such as: the task(s) to be

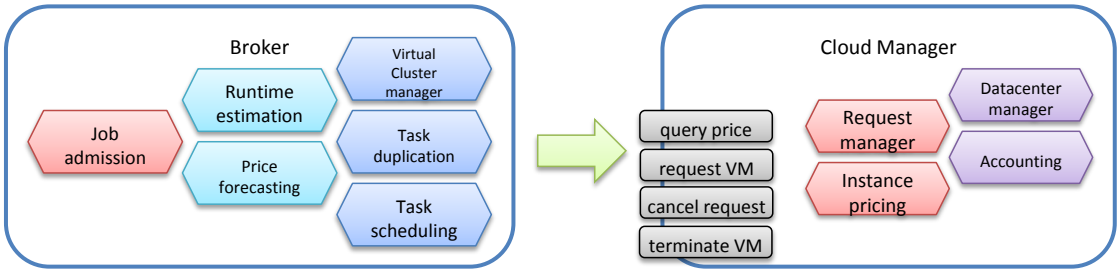


Figure 3.2: Modelled Architecture: Client (Broker) and Server (Cloud)-side Components

executed (e.g., binary files or scripts), the number of required processors, the amount of memory needed, total runtime estimation, and deadlines to be met.

The system model defined in this work has two main components: a Broker (which contains the SpotRMS policy and its various mechanisms), and a cloud provider-side instance management infrastructure, a so called "cloud manager". A graphical representation of the modelled system components is presented in Figure 3.2.

3.2.1 Broker Component

Computational job execution is managed by the Broker, which takes the responsibility of receiving job submissions and assembling a pool of cloud spot instances on behalf of the organisation. The Broker obtains all available information about a job and uses that information to perform scheduling decisions.

More specifically, the Broker manages the following activities: i) Job management: job admission, job execution, job failure, and monitoring of job QoS constraints; ii) Virtual cluster management: bidding, instance selection, instance requesting, and termination.

3.2.2 Cloud Provider Model

Spot instance management activities are performed by the cloud provider. For this component, a similar cloud model as the one described by Yi et al. [107] is adopted, which reflects how the spot market works in the Amazon Cloud. More specifically, the model considers the following characteristics:

- Clients submit a spot instance request, specifying how many instances are needed, an instance type, and an upper limit on how much they are willing to pay per instance/hour (bid);

- The system provides spot instances whenever the client's bid is greater than the currently advertised price; on the other hand, it terminates instances that do not meet the current spot price, immediately without any notice, when a client's bid is less than or equal to the current price.

- The system does not charge the last partial hour when it stops an instance, but it charges the last partial hour when the termination is initiated by the client (the price of a partial hour is considered the same as a full hour). The price of each instance/hour is the spot price at the beginning of the hour.

The cloud manager is responsible for the following tasks: i) Request management: admission control based on bids, and serving valid requests; ii) Spot instance management: new instance requests, terminations due to out-of-bid situations, and billing of hourly charges.

3.3 Problem Context

The problem considered in this chapter is that of provisioning computing instances to execute batches of deadline-constrained jobs. This activity is performed according to the rules of the above-mentioned cloud provider model. The notions of jobs, instances, moldability, speedup, and runtime estimations are now formalised.

A job is an independent unit of work that arrives in the system at a certain time and must be executed on the compute units of a single instance. In this direction, a job j is defined by the following parameters:

- T_j^a is the arrival time of a job given in seconds since $T_1 = 0$ (the start of the job stream, which corresponds to the arrival of the first job);

- R_j is the actual amount of work, or computing time, required by the job. It is the runtime of the job, in seconds, if it runs in a standard instance (an instance of computing power equal to 1 ECU);

- R_j^e is the estimated runtime, as supplied by the job's owner, of how long the job will take to run in a standard instance;
- T_j^d is the deadline of a job, given in seconds since T_1;
- A_j is the average parallelism;
- σ_j is a coefficient of variance of parallelism;
- U is the ID of the user who submitted the job.

Jobs are assumed to be moldable, i.e. they can take advantage of any number of compute units, but restricted to a single instance. Instance types are modelled as containing one or more compute units, each assumed to have a power equal to the amount of ECUs of an equivalent real EC2 instance type. Each instance can only run one job at a time.

Similar to Cirne & Berman [24], to determine the run time of a job in a particular number of compute units, this work employs Downey's analytical model for job speedup [33]. Downey's model requires two parameters: the average parallelism A and an approximation to the coefficient of variance of parallelism σ. The value A defines the maximum speedup a job can achieve; the greater its value the more compute units a job can take advantage of. σ reflects how fast a job can achieve its maximum parallelism; the smaller the σ, the faster the job reaches its maximum speedup, and hence the closer to linear the job's speedup curve is [24]. As this model is derived from characteristics of a range of application classes and has been validated on several hardware architectures, it is general enough to represent the class of application and hardware profile studied in this work, i.e. that of multi-core cloud computing instances.

The values for A and σ are generated using the model of Cirne & Berman [24], which has been shown to capture the behaviour of a range of user jobs. It is assumed that the values of A and σ are known by the users who submit the jobs or, as assumed by Cirne & Berman [24], can be modelled based on other job characteristics provided by users. Therefore they can be used by the resource provisioning strategies.

I is the set of available instance types provided by the cloud, each one containing n compute units. For the instance types considered in this work, the set of possible values of instance ECU is $N : \{1, 4, 5, 8, 20\}$

$$S_j(n_i) = \begin{cases} \frac{A_j n_i}{A_j + \sigma_j(n_i - 1)/2} & (\sigma_j \leq 1) \wedge (1 \leq n_i \leq A_j) \\ \frac{A_j n_i}{\sigma_j(A_j - 1/2) + n_i(1 - \sigma_j/2)} & (\sigma_j \leq 1) \wedge (n_i \geq 2A_j - 1) \\ A_j & (\sigma_j \leq 1) \wedge (1 \leq n_i \leq A_j) \\ \frac{n_i A_j(\sigma_j + 1)}{\sigma_j(n_i + A_j - 1) + A} & (\sigma_j \geq 1) \wedge (1 \leq n_i \leq A_j + A_j\sigma_j - \sigma_j) \\ A_j & (\sigma_j \geq 1) \wedge (n_i \geq A_j + A_j\sigma_j - \sigma_j) \end{cases} \tag{3.1}$$

The speedup $S_j(n_i)$ of a job j on instance i, containing n_i ECUs is given by Equation 3.1. The cases where $\sigma \leq 1$ represent low variance application profiles, where the parallelism is equal to A, except for some fraction σ of the duration which corresponds to the sequential component of the job. A high variable model is also considered ($\sigma \geq 1$) where variance is unbounded and the sequential component can take a greater portion of runtime. Each speedup model is obtained by working out $\frac{R_j(1)}{R_j(n_i)}$. The complete analytical model, including the necessary steps to obtain speedup functions for all application profiles, is found in the work of Downey [33].

3.4 SpotRMS: Resource Provisioning and Scheduling Policy

The Broker is equipped with an instance provisioning and a job scheduling policy (SpotRMS), which forms the core contribution of this chapter.

The following steps are involved in allocating each incoming job to an instance. These activities are described in Algorithm 1.

- When a job is submitted, it is inserted into a list of unscheduled jobs.
- At each scheduling interval, the SpotRMS uses a runtime estimation method to predict the runtime of the job on each available instance type;
- The policy then attempts to allocate the job to an idle, already initiated instance with enough time before a whole hour finishes;
- If unsuccessful, it decides whether the allocation decision can be postponed, based on the maximum time the job can wait so that the chance of meeting the deadline is increased. The maximum wait time is conservatively computed by

subtracting twice the estimated runtime of the job a single compute unit from the time until the deadline ($T_j^d - T_j^a - E_j * 2$). This decision factor is further studied in Chapter 4, referred to as the job's "urgency factor" (U_j).

- If the job cannot be postponed, it attempts to allocate it to an instance that it is expected to become idle soon. Runtime estimates of all jobs running on the VM are required at this step;
- If the job still cannot be allocated, the policy will decide whether to extend a current lease or to start a new one. This decision is made based on information about the estimated runtime of the incoming job on each instance type;
- The policy then checks if there are still idle instances, which are then matched to non-urgent jobs that have been postponed in previous iterations. Idle VMs are the ones that have been flagged to be terminated when the next whole hours finishes.
- Finally, for each job that was allocated to an instance in the current iteration, a correction event is scheduled to trigger at the time the job is expected to finish. This event, if necessary, will then activate the estimation correction and rescheduling mechanism, which corrects the estimation and reinserts all jobs allocated to the affected instance into the list of unscheduled jobs.

These steps are conducted in regular intervals (time t, which can be defined based on the arrival rate of jobs). In this work, t is set to 10 seconds, so that the algorithm operates in on-line fashion, especially to guarantee that jobs with a tight deadline are given the opportunity to start as soon as possible. Use of techniques such as job postponing, runtime estimation, and delayed termination of idle instances are required to ensure that the policy keeps a holistic view of the system workload.

Details of the various steps performed by SpotRMS, such as runtime estimation and correction, rescheduling are given in the following sections.

3.4.1 Job Runtime Estimation

In order to circumvent inaccurate user supplied estimates, especially harmful in backfilling schedulers, several works have proposed methods to predict job runtime, where the system computes the estimated runtime of a job and uses it in place of a user

```
   input  : available instance types I
 1 while true do
 2     while T_current < T_sched do
 3         Receives incoming jobs and inserts in the list LJ;
 4     e ← compute set of estimated runtime of j on all instance types;
 5     P ← all instances currently in the pool;
 6     foreach Job j ∈ LJ do
 7         decision ← FindFreeSpace(j, P);
 8         if decision.allocated = true then
 9             p ← decision.instance;
10             AllocateJobToVM(j, p);
11             continue;
12         U_j ← T_j^d − T_j^a − E_j ∗ 2
13         if U_j ≥ 0 then
14             Delay allocation of j until U_j;
15             Add j to list LNU of non urgent jobs;
16             Remove j from LJ;
17             continue;
18         extendDecision = ComputeLeaseExtensions(j, P);
19         i_pref ← select most economic instance type based on current price;
20         newDecision = ComputeCostForANew(j, i_pref);
21         if extendDecision.cost <= newDecision.cost then
22             trigger lease extension;
23             p = newDecision.instance;
24             AllocateJobToVM(j, i);
25         else
26             p ← LeaseNewVM(i_pref);
27             AllocateJobToVM(j, i);
28     update state of instances;
29     if there are idle instances and LNU is not empty then
30         foreach Job j ∈ LNU do
31             p ← FindFreeSpace(j, P_idle);
32             if p ≠ ∅ then
33                 AllocateJobToVM(j, p);
34                 remove j from LNU;
35                 add j to LJ;
36     foreach Job j ∈ LJ do
37         if j.isAllocated = true then
38             dispatch correction and rescheduling event at time T_sched + E_j(i_p);
39      clear LJ;
40     T_sched = T_sched + t;
```

Algorithm 1: Resource Provisioning and Job Scheduling Algorithm.

supplied estimate. Although complex methods have been proposed, Tsafrir [93] has observed that simple algorithms (e.g. using the average runtime of two preceding jobs by the same user) can result in significant improvement.

Although this work does not make use of backfilling techniques in this work, its motivation to employ runtime estimates is similar to those schedulers. Namely to improve system utilisation, especially important since there is a minimum cost involved for each new computing instance added to the pool. Due to the hourly billing model (assumed in this work to reflect practices of cloud computing providers) every instance runs for a minimum of one hour. As such, relying solely on user-supplied runtime (mostly over-estimated) can lead to unnecessary resource requests.

In the scenarios given here, runtime predictions aid the decision-making process in the following ways: i) SpotRMS can decide whether to add new resources to the pool based on the expected time that currently busy VMs would be free; ii) along with information about current instance prices, it is possible to estimate the cost of running a job on an instance of a given type, thus increasing the chances of meeting monetary constraints.

Previous studies have shown that the "one size fits all" notion does not apply to estimation of job runtime, since no method has been shown to work well in all scenarios [93]. For this reason, several methods have been implemented: "Actual runtime", "Actual runtime with error", "User-Supplied", "Fraction of User-Supplied", and "Recent Average". A runtime estimation is computed as the predicted job runtime in one compute unit (E_j); the estimated runtime on an instance type where $n_i > 1$ is easily obtained from Equation 3.2

$$E_j(i) = \frac{E_j}{S_j(n_i)} \tag{3.2}$$

In the *"Actual runtime"* method, the actual job length, as per the workload trace, is supplied to the allocation algorithm (Equation 3.3); while the *"Actual runtime with error"* approach consists of using the actual length slightly modified by a uniformly distributed random percentage between 0 and 10% (Equation 3.4). Naturally, these two approaches are not realistic as they are based on information that would normally not be available to the estimation method in practice. They are included here for comparison purposes.

$$E_j = R_j \tag{3.3}$$

$$E_j = R_j(1 + \mathcal{U}[-0.1, 0.1]) \tag{3.4}$$

The *"User-Supplied"* approach assumes the job length to be the value informed by the user at job submission (Equation 3.5). Based on previous observations that user supplied runtime estimations are mostly over estimated, the *"Fraction of User-Supplied"* approach has been also devised, which uses a value equal to $\frac{1}{3}$ of the original value as the job length (Equation 3.6).

$$E_j = R_j^e \tag{3.5}$$

$$E_j = \frac{R_j^e}{3} \tag{3.6}$$

The *"Recent Average"* approach consists of using the average runtime of n jobs completed prior to the submission of an incoming job by the same user (in this work, n is fixed at 2). If not enough information is available to compute the estimated length of an incoming job, i.e. less than n jobs have completed at the time of decision-making, the estimation is assumed to be given by the *"User-Supplied"* method. Equation 3.7 describes this calculation; R^u is the set containing the actual runtimes of k recently completed user jobs.

$$E_j = \begin{cases} \sum_{i=k-1}^{k-n} \frac{R_i^u}{n} & k \geq n \\ R_j^e & k < n \end{cases} \tag{3.7}$$

3.4.2 Estimation Correction and Rescheduling

SpotRMS is equipped with a correction and rescheduling mechanism, which is activated whenever a job is detected to have been running longer than expected, regardless of which runtime estimation method provided the estimate. Whenever a job starts running, a correction event is scheduled to trigger immediately after the moment the job should have finished. If, at that moment, the job is still running,

a correction operation is performed. The policy simply assumes a new estimate, and the job remains allocated to the current VM. All other affected jobs, i.e. the ones scheduled to the same instance, which might be delayed, are resubmitted to the scheduler for rescheduling.

The new estimate is assumed to be double the old estimate. At the point where an estimation correction is needed, it is, naturally, unknown for how long the job will run until completion, and the original estimate has no value. Additionally, the job in question is now more likely not to meet its deadline. However, it is important to minimise impact on other jobs scheduled to the same resource. Therefore, doubling the old estimate is assumed to provide a reasonable window for the current job to complete and as result, trigger rescheduling of other jobs.

3.5 Performance Evaluation

This section presents an evaluation of the proposed resource allocation and scheduling policy using trace-driven discrete event simulation which is implemented using Spotsim. The overall objective of these experiments is to quantify the performance of the proposed policy based on three metrics: monetary cost, system utilisation, and deadline misses. Since the proposed policy aims at minimising the cost of building virtual clusters, the monetary cost of such activity is considered as a primary metric. System utilisation indicates how long instances remain idle before they are terminated. Deadline misses refers to the number of submitted jobs which did not finish within the specified deadline; this is a metric directly related to user satisfaction.

In a first scenario, the policy is compared with two base provisioning policies: worst-case and best-case resource provisioning. The objective of such comparison is demonstrating where the proposed policy stands when compared to the baselines, for the purpose of understanding what level of performance can be expected when varying the runtime estimation method. The policy itself is fixed, in order to allow an accurate evaluation of runtime estimation techniques independent of other factors.

In a second experimental scenario, the effects of various runtime estimations on the proposed policy are evaluated and compared to one another to allow an understanding of which runtime estimation method should be used for a given workload.

Virtual Machine Types

As stated earlier, SpotRMS aims at choosing the most efficient instance type to run a job. Five instance types were used in the experiments given here. They were modelled directly after the characteristics of available standard and high-CPU Amazon EC2 types. The types available to be used are m1.small (1 ECU), m1.large (4 ECUs), m1.xlarge (8 ECUs), c1.medium (5 ECUs), and c1.xlarge (20 ECUs).

Workload

The chosen job stream was obtained from the LHC Grid at CERN, available at the Parallel Workloads Archive [34], and is composed of grid-like embarrassingly parallel tasks. A total of 100,000 jobs were submitted over a period of seven days of simulation time. This workload is suitable to these experiments because of its bursty nature and for being composed of highly variable job lengths. These features require a highly dynamic computation platform that must serve variable loads while maintaining cost efficiency.

Originally, this workload trace did not contain information about user supplied job runtime estimates and deadlines. User runtime estimates were generated according to the model of Tsafrir et al. [92]. Deadlines were then generated based on these runtime estimates. The job deadline corresponds to its submission time plus the user supplied runtime estimate multiplied by a random multiplier, uniformly generated between 1.5 and 4.

3.5.1 Comparison with Best-case and Fixed-price Scenarios

In this section, a comparison is made of the proposed scheduling policy with two baseline policies. Based on perfect information from the workload trace (actual job length and parallelism parameters A and σ), it has been possible to compute how much would be the best possible cost that could be achieved to run all 100,000 jobs using the most efficient instance type for each job, considering multiple pricing schemes. The most efficient match for a job depends on its maximum speedup, the job's length and the instance's cost per hour. As a general rule, short jobs or jobs that provide little or no parallelism run more efficiently on less powerful, cheaper instances; whereas longer jobs (execution time in the order of hours) that provide good parallelism are more suitable for high-CPU instances, which provide a lower cost per ECU.

Table 3.1: Total costs that would be obtained to execute all jobs using the most efficient on-demand and spot instances in a hypothetical scenario where all information about each job is known

Instance type	Percentage of jobs	Fixed-price	Best-case
M1.SMALL (1 ECU)	6.65%	$1114.62	$371.54
C1.MEDIUM (5 ECUs)	84.56%	$6942.38	$2314.13
C1.XLARGE (20 ECUs)	8.79%	$313.84	$104.61
	Total:	$8370.84	$2790.28

To compute monetary cost in the fixed-price scenario, an off-line simulation has been performed where only on-demand fixed-price instances are provisioned, but jobs are scheduled on the most efficient machines types for each job. The off-line simulation assumes all jobs are known at time 0, and that jobs can run back-to-back, leaving no idle machine time. Therefore, this scenario provides an optimistic figure. The best-case resource provisioning scenario is a hypothetical lower bound devised to evaluate the cost-effectiveness of the proposed policy. Monetary costs in this scenario are assumed to be $\frac{1}{3}$ of their fixed-priced counterparts; in line with observed price patterns of Amazon EC2, where spot instances cost on average a third of an on-demand instance offering identical features.

Table 3.1 lists the costs that would be obtained in both fixed-price and best-case resource provisioning scenarios. It also reports on the individual percentage of jobs that ran on each instance type as an indication of the bias towards high-end CPU instances.

As shown, $2790.28 corresponds to the best result achievable. Therefore, the aim of any resource provisioning policy is to obtain a cost as close as possible to this value.

SpotRMS, when running with the *"Recent Average"* estimation method was able to obtain an improvement of about 60% when compared to the fixed-price scenario, and just 23% worse than the best case.

3.5.2 Effect of Different Runtime Estimation Methods

This section presents the results of applying the 5 proposed runtime estimation schemes. Specifically, it presents a comparison of their effects on the following metrics: monetary cost, number of deadline misses, and the overall system utilisation.

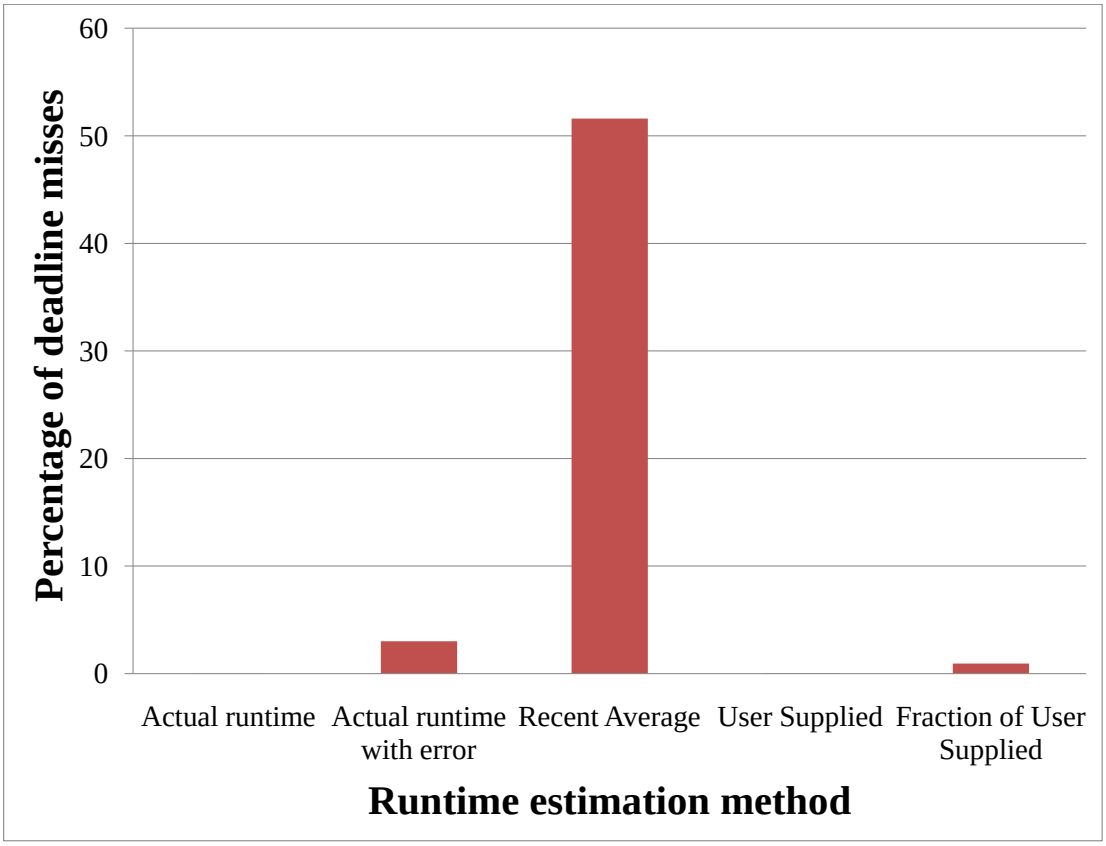


Figure 3.3: Deadline misses before the correction and rescheduling mechanism was introduced

All values presented correspond to an average of 30 simulation runs. Each run is set to start at a random point in time, uniformly chosen between March 1st, 2010 and February 1st, 2011. These dates correspond to the available price traces obtained from Amazon EC2's US-EAST region.

Two sets of experiments were conducted. In first set, SpotRMS was not equipped with the correction and rescheduling mechanism, described in section 3.4.2. In these experiments, once the policy made a decision based on a runtime estimate, jobs would remain allocated to the instance chosen in the first decision. This resulted in an excessive amount of deadline misses, especially when using the *"Recent Average"* estimation method, as can be seen on Figure 3.3. This fact is due to underestimations, that caused many jobs to be allocated to the same instances. Once a certain job that was expected to finish at a certain time did not finish, all other jobs

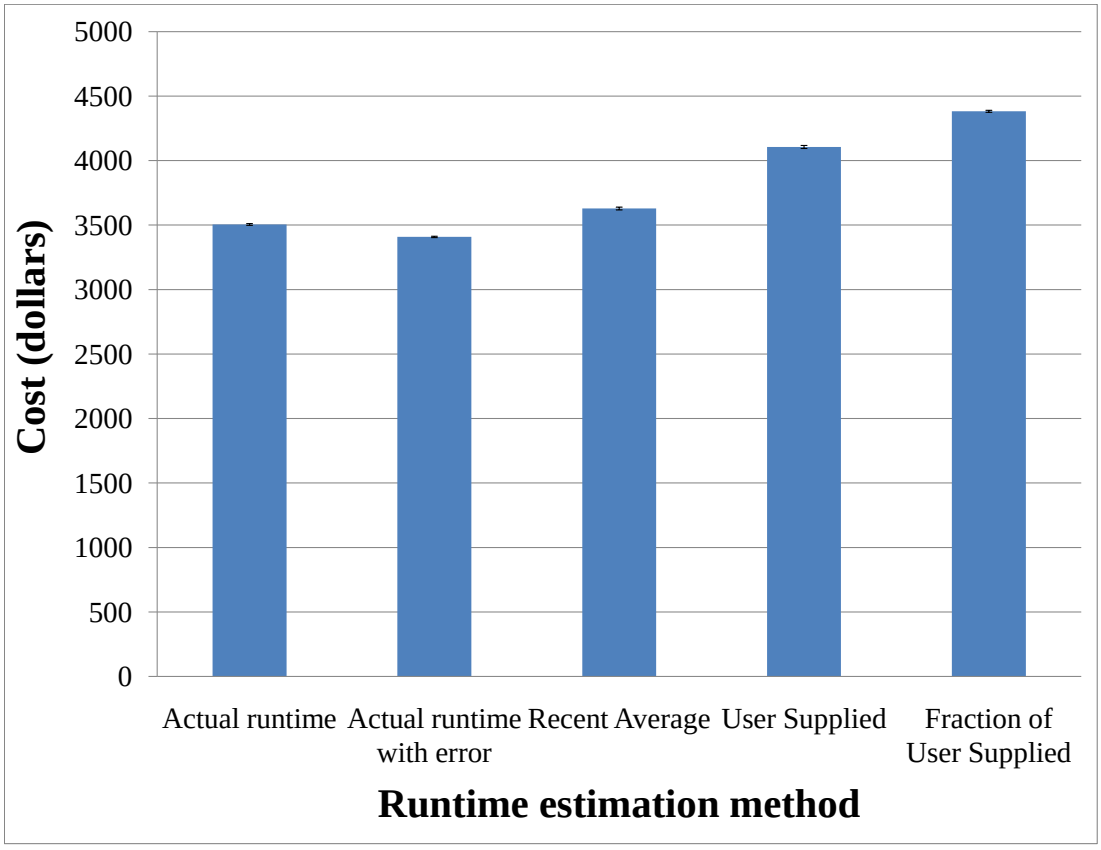


Figure 3.4: Effect of different runtime estimation methods on monetary cost

would be delayed. By adding correction and rescheduling, the policy was able to virtually eliminate the occurrence of deadline misses. Therefore, this metric is not reported in the following results.

Figures 3.4, and 3.5 show the effect of changing the runtime estimation component on the total monetary cost, and system utilisation respectively. These results correspond to the second set of experiments, which were collected after the correction and rescheduling mechanism was implemented.

Results demonstrate that, contrary to expectations, precise information does not necessarily translates into more efficient allocation, especially in terms of cost. This fact can be observed in the comparison between the *"Actual runtime"* and *"Actual runtime with error"* methods, where the latter performs better. Based on observations of the simulation logs, it is possible to attribute this difference to moderate over-estimations, that cause more instances to be requested, which are reused of-

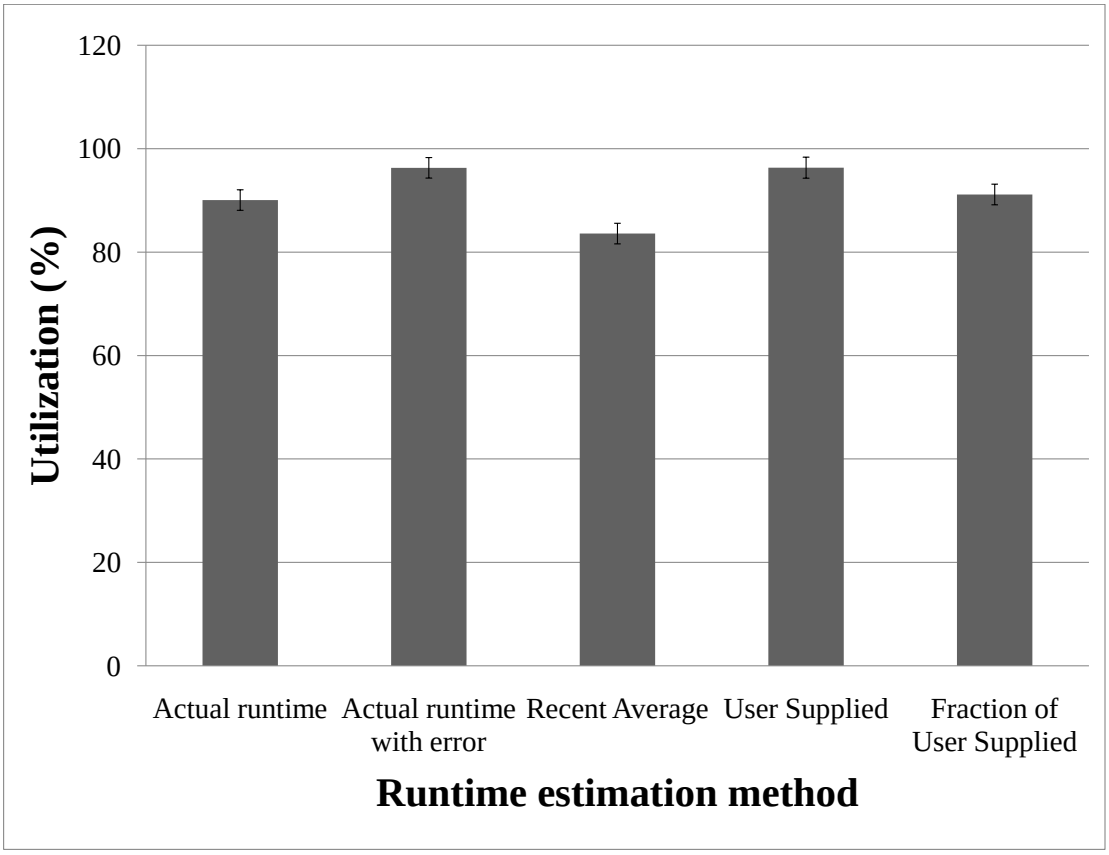


Figure 3.5: Effect of different runtime estimation methods on system utilisation

ten by short jobs. On the other hand, the exact estimates provided by the optimal method cause the allocation system to request fewer instances and also to terminate instances more often. Incoming jobs then cause more new instances to be requested, which are then more likely to remain idle if the job that triggered the request was a short one. This fact can be confirmed by observing the system utilisation under the effects of the *"Actual runtime"* method (90%) and *"Actual runtime with error"* (96%).

In term of deadline misses, runtime estimations methods that tend to over-estimate provide better results. However, excessively over-estimated runtimes were shown to increase costs significantly, as they cause a much higher number of instances to be requested, especially more expensive instances. Although these instances are sometimes reused by other jobs, in most cases the allocation algorithm will choose to create new instances, by mistakenly considering that incoming jobs are long, thus

requiring more powerful instances to complete their execution within their deadlines. This observation can be inferred from the results obtained by the *"User-Supplied"* and *"Fraction of User-Supplied"* estimation methods, also depicted on Figure 3.4; both methods tend to provide excessively over-estimated runtimes.

It was also observed that good system utilisation alone does not necessarily translates into lower costs. For example, by using the *"Recent Average"* estimation method, SpotRMS could achieve a better cost than *"User-Supplied"*, but the utilisation was significantly lower. The key aspect to observe in this scenario is the importance of choosing the correct type of instances for each job. Smaller instances, even when not utilised efficiently, have less impact on the final cost, than larger and more costly instances, which must be used efficiently to compensate for their cost.

In conclusion, in these scenarios, slightly over-estimated runtimes have shown to be beneficial. On the other hand excessively over-estimates are ineffective because of the bias towards larger and costly instances. Under-estimations have shown to increase the chance of deadline misses, but they are not as ineffective, as they can be corrected and affected jobs can be rescheduled. In any case, it is reasonable to conclude that accurate estimations help in achieving lower costs.

3.6 Summary

Building dynamic virtual clusters using cloud resources is an effective way of saving in monetary costs. This chapter has provided a cost-effective solution to provision a virtual cluster using spot market resources to run computational jobs having a deadline as a QoS constraint. To address this challenge, a resource allocation and job scheduling policy which takes into account variations in price and performance of cloud resources and aims at choosing the most efficient VMs for deadline-constrained jobs was presented. The performance of the proposed policy has been evaluated by comparing it to worst-case and best-case scenarios. An improvement of up to 60% in cost has been obtained in comparison with the worst-case, and the policy has performed close to the best-case.

This work also evaluated 5 runtime estimation methods and their effects on the policy performance. For the given workload, it can be concluded that more accurate estimation methods provide significantly superior results, when compared to methods that excessively over-estimate runtimes.

Chapter 4

Reliable Provisioning of Spot Market Resources

This chapter proposes a resource provisioning strategy that addresses the problem of running computational jobs on intermittent VMs. The main objective is to run applications in a fast and economic way, while tolerating sudden (unplanned) unavailability of spot instances. This work builds on SpotRMS, adapting the algorithm introduced in chapter 3 to provide more sophisticated bidding strategies to handle such failures.

To mitigate potential unavailability periods, a multifaceted fault-aware resource provisioning policy is proposed. This solution employs price and runtime estimation mechanisms, as well as one of three fault tolerance techniques: checkpointing, task duplication, and migration. This is evaluated with strategies using trace-driven simulations, which take as input real price variation traces, as well as application traces from the Parallel Workload Archive. The results demonstrate the effectiveness of executing applications on spot instances, respecting QoS constraints, despite occasional failures.

More specifically, this chapter introduces:

- A multifaceted resource provisioning approach, that includes novel mechanisms for maximising reliability, while minimising costs in a spot instance-based computational platform;

- A bidding mechanism that aids the decision-making process by estimating future spot prices and making informed bidding decisions;
- An evaluation of two novel fault tolerance techniques using migration and job duplication, and comparing them to existing checkpointing-based approaches.

The contents of this chapter are derived from a published research paper:

VOORSLUYS, W., AND BUYYA, R. Reliable Provisioning of Spot Instances for Compute-intensive Applications. In *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications* (Fukuoka, Japan, Mar. 2012), IEEE, pp. 542–549

4.1 Background and Motivation

Despite the possibility of failures due to out-of-bid situations, as discussed in chapter 3, it is advantageous to utilise spot instances to run compute-intensive applications at a fraction of the price that it would normally cost when using standard fixed-priced VMs. Specifically, this work has demonstrated the effect of different runtime estimation methods on the decision-making process of a dynamic job allocation policy. This policy requests and terminates spot instances on-the-fly as needed by the computational jobs, as well as choosing the best instance type for each job based on the estimated job execution time of each available type.

Chapter 3 identified that users could bid sufficiently high so that the chance of spot instance failures due to out-of-bid situations would be negligible. In reality however, even though users only pay the current spot price at the beginning of each hour, regardless of the specified bid, there are incentives for bidding lower. Andrzejak et al [3], evaluated checkpointing techniques for spot instance fault tolerance, and observed that by bidding low, significant cost savings can be achieved, however execution times increase significantly. By increasing the budget slightly, execution times can be reduced by a significant factor.

4.1.1 The Need for Fault Tolerance

There are numerous potential risks and rewards when provisioning a resource pool composed exclusively of spot instances in scenarios where QoS constraints play an important role.

Failures due to out-of-bid situations may lead to the inability to provide the desired quality of service, e.g. prolonged application execution times or an inability of applications to finish within a specified deadline. To overcome this uncertainty, several strategies can be used to decrease the chance of failure or mitigate their effects. To decrease the chance of out-of-bid situations occurring, it is possible to choose to bid as high as possible. However under the spot instance pricing model, users pay the current spot price (not the actual bid). In this situation there is no real advantage in bidding much higher than the spot price. However, there are incentives for adopting more aggressive bidding strategies, i.e. bidding close to or even lower than the current spot price.

Firstly, the cloud provider offers on-demand instances at a fixed price, which are functionally identical to spot instances and are not subject to terminations due to pricing issues. The value set by the provider to these on-demand instances can influence the maximum price a user is willing to bid. Thus, this value acts as an upper bound for bids of users that would rather lease a reliable on-demand instance in cases the spot price is equal or above the on-demand price. However, by analysing the history of spot prices of Amazon EC2, it can be observed that, over the period of about 100 days from July 5th 2011 to Oct 15th 2011, spot prices surpassed on-demand prices several times across most instances types and data centres. For example, the spot price of one of the most economical instances (m1.small) in the US-EAST region, has reached this situation 11 times, for periods of up to 2 hours and 20 minutes, at a price value of up to 17% above the on-demand price. Nonetheless, this occurrences are relatively rare.

Secondly, in scenarios where users predominantly submit high bids, providers would likely increase the spot price to maximise profits. As previously postulated [64], the Amazon EC2 spot market resembles a Vickrey auction style [9], where users submit sealed bids following which the provider gathers them and computes a clearing price.

The pricing scheme thought to be used by Amazon, where all buyers pay the clearing price, is a generalisation of the Vickrey model for multiple divisible goods – the standard uniform price auction upon which the provider assigns resources to users starting with the highest bidder until all bids are satisfied or there are no more resources. The price paid by all users is the value of the lowest winning bid (sometimes, the highest non-winning bid) [109]. It has been observed that this scheme

is a truthful auction, provided that the supply level is adjustable ex-post, i.e. after the bids have been decided [109].

It has also been observed that Amazon may be artificially intervening in the prices by setting a reserve price and generating prices at random [12]. In any case, there is an incentive for users to submit fair bids, based on the true value they are willing to pay for the resource, especially in a scenario where the pricing scheme is not completely transparent.

Thirdly, on a similar note, users may choose to postpone non-urgent tasks when prices are relatively high, hoping to obtain a lower price (the true value) later – a strategy that can be accomplished by placing a bid at the desired price and waiting for it to be fulfilled. In the case of an out-of-bid situation, owners of a non-urgent task may prefer to wait for the request to be in-bid again, rather than obtaining a new resource under new lease terms (e.g. another VM type, or the same type of VM at a higher bid).

Finally, as observed by Yi et al [107], one can bid low to take advantage of the fact that the provider does not charge the partial hour that precedes an out-of-bid situation. Thus, delaying the termination of an instance, even when it is not needed, to the next hour boundary, one can expect a probability of failure before termination, potentially avoiding to pay for the last hour.

The choice of an exact bid value can be empirically derived from a number of factors, including observations of price history; the willingness of the user to run instances at less than a given price or to not run at all, and the minimum reliability level required. These factors, when reflected on the bid value, define how likely the system is able to meet time and cost constraints.

The adoption of aggressive bidding strategies can result in more failures, potentially undermining the cost savings as a result of frequent loss of work. Therefore, resource provisioning policies aimed at running computational jobs on spot instances must be accompanied by fault mitigation techniques, especially tailored for the features of these resources.

Some features of spot instances may also influence the way fault tolerance works. Most notably, hourly-based billing granularity and non-payment of partial hours in the case of failures, guarantees payment of the actual progress of computation [107]. Additionally, given that providers freely provide a history of price

variations, significantly more informed decisions can be made by observing the past behaviour.

4.2 Mechanisms for Reliable Provisioning

To tackle the risks associated with using spot instances, a multifaceted approach which relies on inter-related mechanisms is required. This combination of mechanisms defines how reliably SpotRMS can ensure that computational jobs finish before their deadlines.

The first mechanism (bidding strategies) aims at choosing appropriate bid values based on estimation of price variations and on the job's urgency factor U_j, which influences the choice of when to provision a resource for a given job and how much to bid. The second mechanism (fault tolerance) adds extra levels of fault mitigation through checkpointing and migration of virtual machines, as well as job duplication.

These mechanisms aim at mitigating spot instance unavailability due to out-of-bid situations only, i.e. failures due to price variations. Other types of instance failures, for example, due to hardware faults or network interruptions are not considered. In other words, this work assumes that, if no out-of-bid situation takes place during an instance lifetime, its availability is 100%.

4.2.1 Estimating Cost and Job Urgency

The urgency factor U_j of a job j is the maximum estimated time the job can wait for a resource to be provisioned so that the deadline can still be met. It is computed as per Equation 4.1, where T_j^d is the job's deadline, T_{sched} is the current time, so that $T_j^d - T_{sched}$ corresponds to the time until the job's deadline; α is the urgency modifier; $E_j(i_{pref})$ is the estimated runtime of j on its preferred instance type; and B is the expected time the provider takes to provision a new instance, or boot time (this is assumed to be fixed at 5 minutes).

$$U_j = max(0, T_j^d - T_{sched} - (\alpha * E_j(i_{pref})) + B + t)) \tag{4.1}$$

The greater the value of the α modifier, the more conservative the algorithm becomes, i.e. with higher values of α, U_j approaches 0. A value equal to 0 indicates that a resource must be provisioned immediately to complete the job within the given deadline. Alternatively, lower values of α cause the algorithm to be aggressive, i.e. postpone more provisioning actions in order to maximise the chances of finding lower prices or reusing other job instances.

The key part of the resource provisioning approach of SpotRMS involves the cooperation of bidding strategies that base their decisions on price variation information and the calculation of the value of U_j, based on job runtime estimations. The amalgamation of these information-based mechanisms aids the provisioning process by allowing the broker to make informed decisions on how much to bid, a choice that directly influences the risk of failure and monetary spending.

Furthermore, by combining price information and a job's urgency factor, SpotRMS can decide the best point in time to start a new machine for a job, thus seeking to cover the period that will yield the minimum cost. The rationale behind combining these two pieces of information is to avoid hasty decisions that may increase costs, i.e. to avoid commissioning new resources too early, at times when non urgent jobs can be postponed, or too late, when jobs will most likely miss their deadlines.

This thesis has previously presented a comparison of several runtime estimation policies and their impact on cost, deadline violations, and system utilisation. The mechanism that computes the average runtime of two preceding jobs of the same user ("Recent Average") has performed consistently well. Therefore, in this chapter, this mechanism is used exclusively.

4.3 Bidding Strategies

Five bidding strategies are proposed and evaluated, as listed in table 4.1.

Two strategies use historical information to compute the bid; in these cases, a window of one week worth of price history, specific to each instance type, OS and data centre combination, is fed to the bidding strategy. The output of each strategy is the maximum price, in dollars per hour, to be paid for one particular instance. The minimum bid granularity G is 0.001.

Table 4.1: Evaluated bidding strategies

Bidding strategy	Bid value definition
Minimum	The minimum value observed in the price history window + G
On-demand	The listed on-demand price
Mean	The mean of all values in the price history window
High	A value much greater than any price observed (defined as 100)
Current	The current spot price + G

4.3.1 Minimum

The "Minimum" bidding strategy uses the available historical information to identify the cloud provider's minimum possible bid for a particular instance type. The minimum price observed in the price history is interpreted as being the reserve price and therefore the lowest value to be encountered in the near future.

A bid equal or lower than the reserve price is very unlikely to be met. This strategy defines the bid value as minimum value observed in the price history + G, resulting in a value just over the reserve price. When bidding at such level, SpotRMS will only provision resources when they are available at the minimum possible price, with an inherent risk of not receiving any resources until the job is considered urgent.

The "Minimum" strategy is considered to be aggressive, due to the added risk of eventually causing SpotRMS to provision resources immediately when jobs become urgent. On the other hand, throughout the entire price history interval available in this study, the reserve price for all instance types was observed to be about 1/3 of the on-demand price for an identical instance type. Most instance types have remained at this price point for extend period of times.

Therefore, in the best case, i.e. when an non-urgent job can be executed in its entirety for the lowest price, this strategy is expected to achieve significant savings over conservative strategies.

4.3.2 On-demand

All instance types are also provided as fixed-price on-demand instances. On-demand price information is made publicly available by the cloud provider. The "On-demand" bidding strategy uses the on-demand price as the bid value. As discussed previously,

the on-demand price acts as an upper-bound that users chose with the intent of paying less than the fixed-price for an identical instance, while ensuring longer availability periods than when using lower bids.

4.3.3 Mean

The "Mean" strategy aims at choosing values that are a middle ground between low and high bids. It computes the mean of all observed prices in the price history window and uses the resulting mean as the bid.

4.3.4 High

The "High" bidding strategy uses bid values that are much larger that the hourly price of any instance type, defined in this work as $100. Its aim is to guarantee that the instance will never be out-of-bid especially by ensuring availability in situations that the spot price might be higher that the fixed-price. As discussed previously, these situations are rare but cannot be disregarded.

4.3.5 Current

Finally, the "Current‘" strategy aims at choosing a bid so that the resource is provisioned immediately. It bids slightly higher than the current spot price, thus submitting a value that is immediately in-bid.

In all cases that can yield values lower than the current price, the broker uses the value of U_j to override the bid value, if necessary. Specifically, if the bid is less than the current price and $U_j = 0$, it updates the bid to $p_i + G$. If $U_j > 0$, the algorithm schedules a new bid check to occur at time $T_{sched} + U_j$.

4.4 Fault Tolerance Mechanisms

4.4.1 Hourly Checkpointing and Relocation

Checkpointing consists of saving the state of a VM, application, or process, during execution and restoring the saved state after a failure to reduce the amount of lost

work [55]. In the context of virtual machines, the action of encapsulating execution and user customisation state is a commonplace feature in most virtual machine monitors (VMM) [60]. Saving a VM state consists of serialising its entire memory contents to a persistent storage, thus including all applications and processes that might be running [57].

In this work, it is assumed that checkpointing a running application is the same as saving the state of an entire VM. The advantage of relying on VMM-supported checkpointing is that applications do not need to be modified to enable checkpointing-based fault tolerance. However, it is necessary that cloud computing providers explicitly support such capabilities.

The technique considered in this work is based on an hourly-based VM checkpointing, where states are saved at hour-boundaries. This technique has been previously identified by Yi et al. [107] as the simplest and most intuitive yet effective form for dealing with the cost/reliability trade-off when running applications on spot instances. More specifically, taking a checkpoint on an hourly basis guarantees that only useful computational time is paid, i.e. given that spot instances are billed at an hourly granularity and partial hours, in the case of failures, users are not charged.

The checkpointing technique of Yi et al. [107] assumed that a checkpointed VM would resume when the original spot request, which has a fixed bid and machine type, is in-bid again. No attempt was made to provision a new VM by submitting higher bids for the same machine type, or to bid for other types. This contrasts with the checkpointing solution of this work, which considers relocating the saved state to a new VM to hasten the job completion. This method consists of employing a mechanism by which the state of a VM is frequently saved on a global file system and upon an out-of-bid situation the state is relocated.

SpotRMS triggers a checkpoint only if the currently running job's runtime estimation is more than one hour. In this fashion it prioritises saving the state of long running jobs, potentially reducing the overhead imposed by frequent checkpoints.

The definition of new lease terms is subject to the following decision (which attempts to minimise the price for the remaining duration of the job): (1) leasing an instance of the same type for a higher price in the same data centre; (2) leasing an instance of a different type on the same data centre, or (3) relocating the workload to another data centre where a suitable VM may be leased for a cheaper price. The

overhead of restoring a failed VM in a distinct data centre is assumed to be higher than when the same data centre is chosen. This overhead is taken into account by the algorithm when making a relocation decision.

All computation in the VM is paused while the snapshot is being taken. The overhead of saving an instance state (effectively the same as taking a checkpoint) is defined as the time to serialise a VM's memory snapshot into a file in a global file system. This value is different for each instance type, according to their maximum memory size. The exact values are computed as in the work of Sotomayor et al. [86], which provides a comprehensive model to predict the time to suspend and resume VMs. The times to suspend (i.e. save the state) and to resume (i.e. restore from the latest saved state) a spot instance with m MB of memory, are defined as per equations 4.2 and 4.3 respectively. Values for the rates s and r (in MB/s) are based on the time to write/read m MB of memory to/from a global file system), obtained from numerous experiments on a realistic test-bed [86]. They calculated s to be 63.67 MB/s, and r is 81.27 MB/s (to restore a state in the same data centre). This work assumes half the rate (40.64 MB/s) when moving/restoring a VM state into/from a non-local data centre.

$$t_s = m/s \tag{4.2}$$

$$t_r = m/r \tag{4.3}$$

4.4.2 Live Migration of VM State

Live migrating a VM between two physical servers in a data centre consists of pre-copying the VM state to the destination without interrupting the operation system or any of its applications. The memory page copying process is repeated in multiple rounds in which dirty pages are continuously transferred. Normally, there is a set of pages that is modified so often that the VM must be stopped for a period of time, until this set is fully transferred to the destination. Subsequently, the VM can be resumed in the new server.

Migration allows moving an instance out of its physical server with near zero application downtime. Nevertheless, the performance of the running job is negatively affected. A more detailed study of this technique's impact on application performance

is discussed in chapter 6. In this section, we introduce the technique as a fault tolerance mechanism.

This technique is similar to the checkpointing and relocation approach, which used a form of "cold migration", where the VM is paused while a snapshot is being taken. When using live migration, computation being performed in the VM is not paused. Moreover, no state is saved to a global filesystem while the instance is running so that the instance cannot be restored after a failure has occurred. In this sense, SpotRMS anticipates the need of a live migration to avoid loss of work in the case of an out-of-bid situation.

Information about the bid and current spot price are used to trigger a live migration. The migration heuristics consist of tracking changes in spot prices and migrate VM state away from a spot instance when the current spot price approaches the bid. SpotRMS triggers migration on a rising-edge, similar to a novel checkpointing technique presented by Yi et al. [107]. A rising-edge is any occurrence of a price increase.

Additionally, similarly to checkpointing, SpotRMS only triggers a migration of a certain VM if the currently running job's runtime estimation is more than one hour. This technique prioritises migration of long running jobs, which would impose more loss of work if prematurely terminated, especially if it had been running for a long time.

When a migration is triggered, the affected spot instance is scheduled to be terminated in the next hour boundary. It remains usable by short running jobs until it is terminated. The destination spot instance is leased by SpotRMS using the High bidding strategy, a step taken to ensure successful execution of the remaining job length.

The time taken to migrate a VM is defined similarly to the checkpointing time, as it also involves transferring the entire VM memory over the network. However, as described in chapter 6, and measured by Akoush et al. [1], extra migration time needs to be added to account for the transfer of dirty pages, which is dependent on VM memory-size, as well pre- and post-migration steps, and downtime, which are constant. Extra migration time is assumed to 10% of t_s, plus a constant factor of 5 seconds. The performance of the running job is degraded by 20% for the duration of migration, in line with values presented in chapter 6.

Table 4.2: Factors and their levels

Factor	Possible values
Bidding Strategy	Minimum, Mean, On-demand, High, Current
α	1, 2, 4, 8, 10, 20
Fault tolerance mechanisms	None, Migration, Checkpointing, Job duplication

4.4.3 Duplication of Long Jobs

A fault tolerance mechanism that does not require any application-assisted or provider-assisted techniques has also been developed. Job duplication is proposed as a simpler fault-tolerance method to allow rapid deployment of applications on spot instances.

Similar to replication and migration, duplication of jobs aims at increasing the chance of success in meeting deadlines when running longer jobs (greater than one hour) over a period of frequent price changes. Therefore, a duplication-based technique was implemented and evaluated.

This technique also relies on estimates of job runtimes. It creates a replica of each job that is expected to run for more than 1 hour. The replica is submitted to the same scheduling policy as the original job. The algorithm applies the same rules as it does to a regular job, but avoids choosing the data centre/type combination where the original job will run. Choosing a different combination for a replica is an obvious choice, since two jobs running on the same data centre, using the same instance type, will fail at the same time if/when the price increases.

4.5 Performance Evaluation

In this section, the proposed fault-aware resource allocation policy, along with the effect of its mechanisms, is evaluated using trace-driven discrete event simulations. The policy's performance is assessed based on three quantitative metrics: two absolute (monetary cost and deadline violations) and one relative (dollar per useful computation).

Given that there is a trade-off between these metrics, their interaction is observed. For example, assuring fewer deadline violations usually means provisioning more resources, thus leading to higher costs.

4.5.1 Experimental Design

The experiments were designed to study the influence of the following factors: (1) the bidding strategy; (2) the value of the urgency factor modifier α, and (3) the choice of fault tolerance mechanism. The factors and their levels are listed in Table 4.2.

Not all combinations of factors have been simulated. For example, there is little sense in combining the High bidding strategy with a fault tolerance mechanism, given that the bidding fashion itself completely avoids failures. All values presented correspond to an average of 31 simulation runs. When available, error bars correspond to a 95% confidence interval.

Cloud Characteristics

The cloud provider characteristics were modelled after the features of Amazon EC2's US-EAST geographic region, which contains 4 isolated data centres. Instance types were modelled directly after the characteristics of available standard and high-CPU types with the Linux OS. An instance CPU power is measured in compute units (CUs). Standard instances contains a balanced amount of memory and CUs. High-CPU instance types contain proportionally more CUs than memory. The standard instance types available are: SMALL (1 ECU), LARGE (5 ECUs), and XLARGE (8 ECUs), while high-CPU types are: MEDIUM (5 ECUs), and XLARGE (20 ECUs). It is assumed that an unlimited amount

A period of 100 days worth of pricing history traces was collected from Amazon EC2 comprising dates between July 5th, 2011 and October, 15th, 2011. The beginning of this period corresponds to the date Amazon EC2 started offering distinct prices per individual data centre, rather than per geographic region. In total, 20 different price traces were collected, i.e. for all 5 instances types in 4 different data centres.

Workload

The chosen job stream was obtained from the LHC Grid at CERN [34], and is composed of embarrassingly parallel tasks. A total of 100,000 jobs were submitted over a period of seven days, starting from a randomly generated time within the available price history.

This workload was suitable for these experiments due to its bursty nature and for it being composed of highly variable job lengths. These features require a highly dynamic computation platform that must serve variable loads while maintaining cost efficiency. The moldability parameters A and σ of each job are assumed to be known by the broker.

Originally, this workload trace did not contain information about user-supplied job runtime estimates and deadlines. User runtime estimates were generated according to the model of Tzafrir et al. [92]. In this model, a job's maximum allowed runtime corresponds to the runtime estimate multiplied by a random multiplier, uniformly generated between 1.5 and 4. Consequently, the job's deadline corresponds to its submission time plus its maximum allowed runtime.

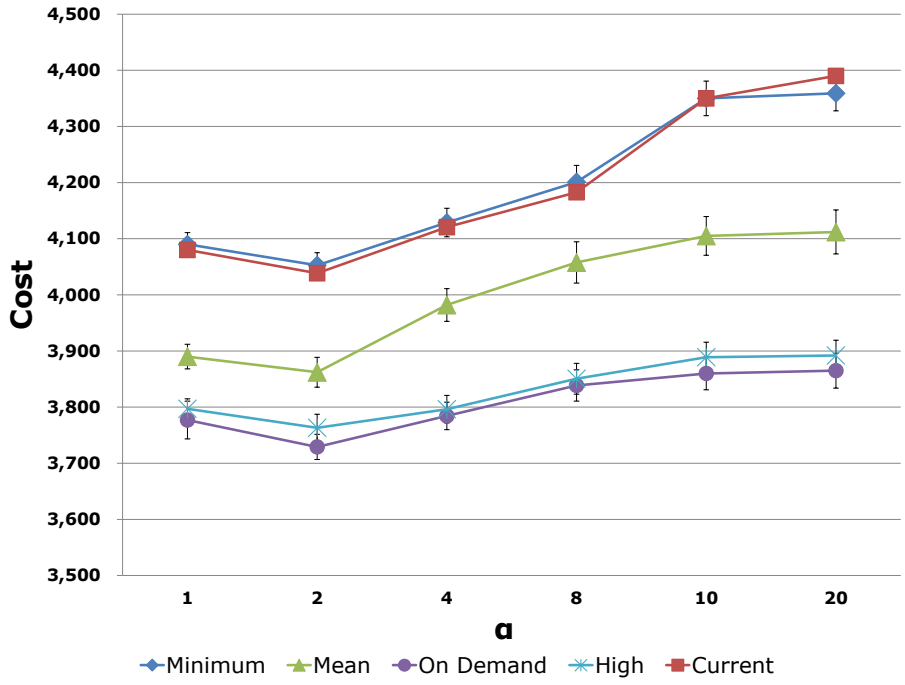
4.5.2 Effects of Bidding Strategies

In order to understand how bidding strategies work in isolation, an experiment was devised to gauge their effectiveness in a scenario where no fault tolerance mechanism is active. In this scenario a failed job must be restarted from the beginning after a failure.

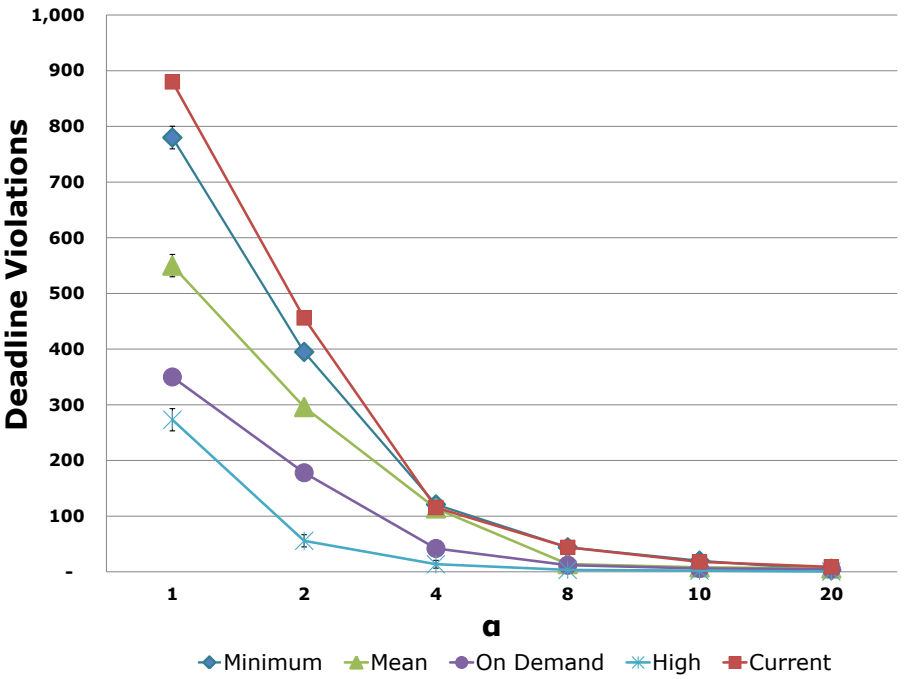
This experiment was aimed at quantifying each strategy's performance when paired with different values of α. Figure 4.1 shows the effect of the most aggressive ($\alpha = 1$) to the most conservative ($\alpha = 20$) urgency estimations under various bidding strategies.

Bidding strategies that produce higher bids tend to perform better, both in terms of cost and deadline violations. In particular, it can be observed that the "On-demand" strategy performs the best. It avoids failures that would otherwise have happened due to minor price increases and avoids incurring the cost of high prices above the on-demand price. This fact can be noticed by the performance comparison between "On-demand" and "High", which performs similarly, but incurs extra cost due its higher bids.

On the other end of the spectrum are strategies that aim at bidding at low values. They lead to the highest costs. This can be attributed to more failures, thus several job restarts, which in turn result in more loss of work. The "Current" strategy performs equally poor as compared to "Minimum", as they both bid at values that are likely to be out-of-bid at the slightest price increase.



(a) Monetary cost



(b) Deadline violations

Figure 4.1: Effect of aggressive and conservative urgency estimation modifier (α) under various bidding strategies

Use of the "Mean" strategy places costs midway between the others. In fact, this strategy does not cause as many failures as the low bidders, but falls short at avoiding most failures that occur when spot prices rise to the levels of the on demand fixed-price. Therefore, it can be concluded that, in the absence of a fault tolerance technique to mitigate out-of-bid situations, bidding relatively high proves to be the best strategy.

The value of α significantly influences both cost and deadline violations, consistently over all bidding strategies. Figure 4.1 indicates an optimal value of 2, which yields the lower costs, although for some bidding strategies, the difference between 1 and 2 is not statistically significant.

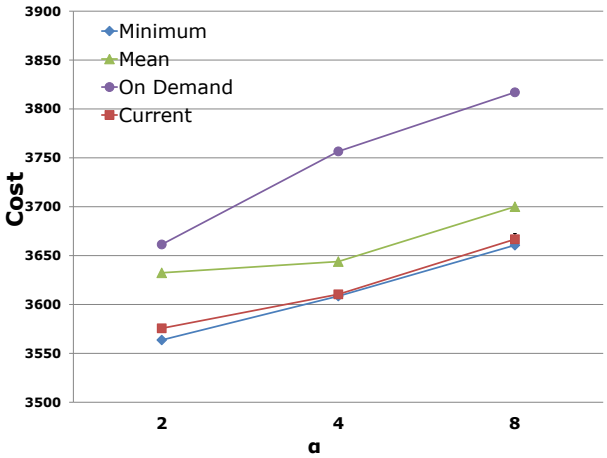
Regarding the deadline metric, 1 and 2 lead to many more deadline violations. This is due to the fact that lower values of α cause the algorithm to postpone more decisions, which in turn leads to the inability to provision resources "at the last minute". Conservative values, on the other hand, lead to virtually no violations, but significantly higher costs.

4.5.3 Migration, Checkpointing and Job Duplication

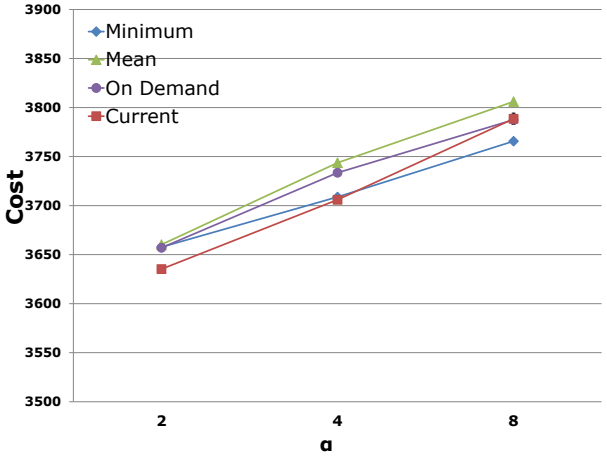
The results produced demonstrate the positive effects of fault tolerance mechanisms when paired with bidding strategies and urgency factor estimation. Figure 4.2 provides a comparison of migration, checkpointing and job duplication on the cost metric. It only shows the values of α set to 2, 4 and 8, which yield the best costs in all cases.

An interesting fact is that migration performs better when paired with bidding strategies that choose lower bid values, such as Minimum and Current, whilst checkpointing benefits from higher bid values, such as Mean and On-demand. This behaviour is coherent with the features of each mechanism. Migration tends to have more choices after an out-of-bid situation given its ability to choose other types of instances from multiple data centres under higher bids. Checkpointing, on the other hand, has a higher overhead and benefits from a higher chance of being in-bid most of the time.

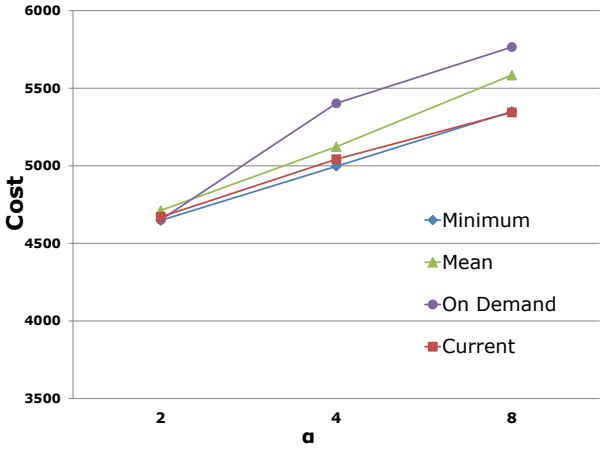
Job duplication performs poorly in all cases, yielding much higher costs when compared to the case when no fault tolerance exists. It's merit however, lies on its simplicity and the capability of replicating jobs across multiple data centres. Therefore, it can be useful in cases where an extra level of redundancy is required.



(a) Migration



(b) Checkpointing



(c) Job duplication

Figure 4.2: Performance of migration, checkpointing and job duplication on monetary cost

Table 4.3: Analysis of the dollars per useful computation metric

Rank	Fault tolerance	Bidding strategy	α	Dollars per useful computation	Worsening related to best (%)
1	Migration	Minimum	2	0.03578	0
2	Migration	Current	2	0.03588	0.29
3	Migration	Minimum	4	0.03613	0.978
4	Migration	Current	4	0.03614	1.006
5	Migration	Mean	2	0.03641	1.736
6	Checkpointing	Current	2	0.03647	1.881
7	Migration	Mean	4	0.03648	1.932
8	Migration	Minimum	8	0.03661	2.279
9	Checkpointing	On-demand	2	0.03663	2.330
10	Checkpointing	Mean	2	0.03666	2.412
...					
18	None	On-demand	2	0.03736	4.224

To help gauge a more precise metric, dollars per useful computation is defined as the ratio between the total cost and the number of jobs that finished within their deadlines. Table 4.3 ranks the 10 best factor combinations according to this metric.

The combinations that employ migration rank consistently higher, which makes these combinations good candidates for environments where strict deadlines must be respected. The migration technique, along with the Minimum bidding strategy and $\alpha = 2$ produced the lowest cost. However, $\alpha = 8$ produced the least number of deadline violations (30 out of 100,000 jobs). These results confirm that the trade-off between cost and deadline violations applies in this case.

These results demonstrate that it is the interaction of factors that influence the exact choice of bidding strategy. It is expected that, in absolute terms, more conservative urgency factors will lead to better adherence to deadlines but also a greater cost.

4.6 Summary

A multifaceted resource provisioning policy that reliably manages a pool of intermittent spot instances has been presented. This policy comprises multiple mechanisms supporting five bidding strategies, an adjustable urgency factor estimator, and three fault-tolerance approaches.

Extensive simulations under realistic conditions were performed, reflecting the behaviour of Amazon EC2 and based on a history of its prices. Results demonstrate that both cost savings and stricter adherence to deadlines can be achieved when properly combining and tuning the policy mechanisms. The fault tolerance mechanism that employs migration of VM state in particular provides superior results across virtually all metrics.

Chapter 5

Model-based Predictive Bidding Strategies and Alternate Volatility Scenarios

In this chapter, the use of a statistical model that captures the behaviour of spot prices in cloud computing data centres is explored. The model provides parameters for a Mixture of Gaussians (MoG) distribution that generates time-independent price patterns and an exponential distribution that generates time intervals between price changes. Distribution parameters are based on the price variation patterns of an actual Amazon's EC2 region.

The model is used in two scenarios. First, it provides SpotRMS with a price prediction mechanism, which uses the model to generate future price variations. SpotRMS subsequently uses this information for resource provisioning decisions. The main objective of studying this scenario is to investigate whether significant gains could be obtained should an accurate price pattern model become available to SpotRMS. Second, by modifying certain model parameters in a controlled manner it is possible to create alternate models that generate price patterns of varying volatility. These are then used in scenarios set up to study the effectiveness of SpotRMS under volatile spot prices.

Specifically, this chapter introduces:

- A prediction-assisted resource provisioning policy;

- A study of the effect of price variation volatility on the performance of spot instance provisioning and job scheduling.

5.1 Background and Motivation

In the previous chapters, it has been demonstrated how spot instances can bring significant advantages to consumers of cloud computing resources provided that appropriate bidding strategies and fault tolerance mechanisms are employed. There, the decision-making process of SpotRMS was largely based on short-term historic spot price information. The bidding strategies used historic information to compute the bid (e.g. mean price for the past week), a value required at the time of provisioning new spot instances.

During the development of those bidding strategies, it became clear that it would be beneficial to obtain a model of the behaviour of a spot market, and in addition, understanding the trends and patterns of spot prices over time and their effect on resource management and job scheduling activities. The application of such a model aims at improving the efficiency of resource provisioning by predicting future prices, thus allowing more informed bidding decisions.

At present, Amazon's spot prices do not vary significantly over time. This is an indication that the provider possesses enough capacity to serve most requests without causing price spikes, even though users may choose to bid high. However, in a more contentious computational market, where providers operate more efficiently, consumers would need to bid judiciously and expect a more volatile price pattern. For this reason, it is necessary to study alternate models of spot price variation that simulate alternate patterns, while retaining the general behaviour as captured by the history-based model.

A model capable of closely mimicking the behaviour of spot instance price variation is useful from multiple perspectives. First, it helps users predict the near future, allowing them to make informed decisions on how much to bid, thus reducing the risk of instance failures, while keeping costs low. Second, it can serve as a powerful tool for researchers to study the effects of price variations in scenarios such as resource provisioning and scheduling. Finally, cloud providers themselves can draw insights on how their pricing mechanisms are behaving by observing patterns of usage and the effect of varying mechanism parameters when changes are required.

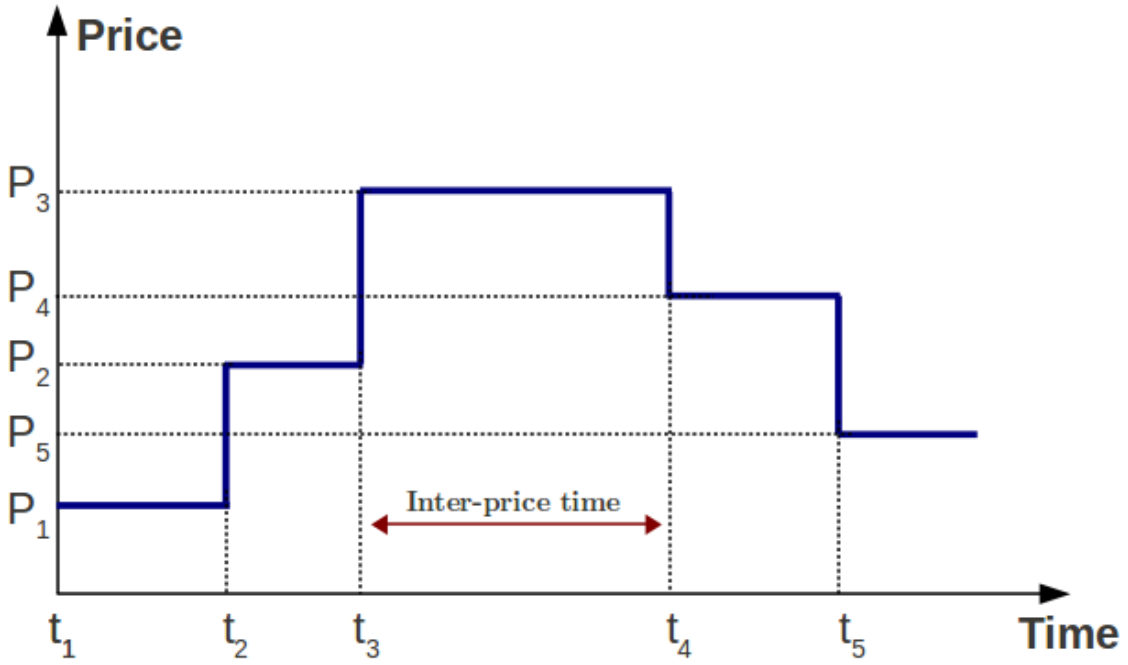


Figure 5.1: Spot price and the inter-price time of spot instances.

Therefore, the model applicability is investigated in two scenarios. First, it is used to predict future price variation patterns and this information be used to aid SpotRMS' mechanisms. Second, slight variations are applied to model parameters, thus causing volatility of spot prices to increase or decrease; this technique is then used to create alternate models which can themselves be used to study the effects of volatility on bidding strategies.

5.2 The Model

The model used in this study was described by Javadi et al. [51]. The modelling process and the resulting model itself are briefly described here.

Spot instances have two variables that are specified by the cloud provider: the spot price and inter-price time. A further variable (user's bid) is determined by consumers. The model is concerned with analysing and modelling spot prices and the inter-price time as two highly volatile system variables. These variables are illustrated in Figure 5.1 where P_i is the price of a spot instance at time t_i. So, the inter-price time is defined as $T_i = t_{i+1} - t_i$.

Table 5.1: Parameters of the mixture of Gaussians distribution with 3 components for spot price in US West.

Type	**Price MoG** ($k = 3$)		
	$\vec{p}$	$\vec{\mu}$	$\vec{\sigma^2}$
m1.small	0.999 0.001	3.982 9.667 9.667	0.013 0.056 0.056
c1.medium	0.194 0.712	7.710 7.978 8.349	0.005 0.021 0.003
m1.large	0.015 0.867	44.060 15.827 16.661	33.061 0.109 0.016
m1.xlarge	0.113 0.884	33.302 31.667 58.983	0.055 0.411 756.622
c1.xlarge	0.480 0.148	32.313 32.000 31.181	0.825 0.000 0.168

Table 5.2: Parameters of the exponential distribution for inter-price time in US West.

Instance type	**Exp(μ)**
m1.small	1.062
c1.medium	0.982
m1.large	0.895
m1.xlarge	0.836
c1.xlarge	0.868

The traces used to derive the model were composed of one year price history of all Amazon spot instances from February 2010 to mid-February 2011. The first 10 months (Feb-2010 to Nov-2010) were used in the modelling process, and the remaining 2-month period was used for model validation purposes. This model considers the spot instances offering the Linux operating system of Amazon's US-WEST (Northern California) data centre.

From observations of density functions of both time series using several distributions, Javadi et al. [51] have proposed a MoG distribution for price, and an exponential distribution for inter-price time, as good candidates for approximating such density shapes. Parameter estimation and goodness of fit tests were performed. Using this, a model with a MoG comprised of three components ($k = 3$) for spot price and an exponential distribution for inter-price time demonstrated a good fit. The model parameters for the price distribution, i.e. vectors of probability, mean, and variance are listed on Table 5.1. The parameter for the inter-price time distribution, i.e. the mean, is listed on Table 5.2.

5.3 Prediction-assisted Provisioning Strategy

In the previous chapters, the bid decision-making process of SpotRMS was based on historic spot price information. Here, the policy is augmented by adding a predictor component to the algorithm.

In the same way cloud providers bills for instances used, it is assumed that users are charged for a minimum lease period of 1-hour. Therefore, SpotRMS depends on predictions to optimise instance utilisation by reusing instances that are idle or may become idle for the remainder of an hour. This practice accommodates the execution of common workloads containing a significant number of short jobs, such as the workload used in the experiments conducted here.

As in previous chapters, a trade-off between cost minimisation and the risk of missing job deadlines is inherent to this scenario. A trivial way of approaching this issue is by submitting high bids, since the amount paid per hour is the current spot price at the beginning of the hour, regardless of the bid value. However as shown earlier, judicious bidding leads to considerably lower costs provided that out-of-bid situation are avoided or mitigated. Therefore, the hypothesis in this chapter is that prediction of future pricing conditions offers advantages over previous approaches, by calculating bids more precisely and allocating jobs to the most economical time slots.

A statistical model is applied to generate a distribution of price changes that resembles spot instance price variations for selected instance types. Given that the model is time-independent, prior to the beginning of its activities (time T_1), SpotRMS pre-generates price variation activity for a period of 60 days. This means that predictions are unaffected by actual price history during simulation. The result of this price generation process, for each instance type i, is an ordered set of price change events P_i, each represented by the tuple (t, p), where t is time in seconds since T_1 and p is the hourly spot instance price in dollars, e.g. $(0, 0.031)$. The set is ordered by t in ascending order.

The resource provisioning and job scheduling algorithm is very similar to Algorithm 1. Here, the complete algorithm is described with the additional prediction-assisted decision-making steps highlighted. The original algorithm periodically performs the following steps:

- The algorithm takes a job from the list of unscheduled jobs;

- It performs runtime estimations to predict the approximate runtime of the job on each available instance type;
- It then attempts to allocate the job to an idle instance (if any) with enough time to complete the job before a whole hour finishes;
- If unsuccessful, it attempts to allocate the job to a VM that is currently running jobs but is expected to become idle soon and will fit the job before a whole hour finishes;
- If the job still cannot be allocated, the algorithm will decide whether it is advantageous to either: postpone the allocation decision according to the job's urgency factor and pricing conditions; extend the current lease, or indeed start a new VM lease. Bids of new leases are computed according to predicted future prices;
- A time slot search algorithm is introduced in order to take model predictions into consideration when deciding to which instance, current or new, a job is best allocated.

5.3.1 Time Slot Search Algorithm

An additional decision-making step consists of using a greedy algorithm to perform a search for future time slots that minimise the monetary cost of a given execution.

The decision-making process is used for deciding whether to lease new spot instances or to extend an existing lease. The calculated monetary cost is expected to vary depending on when a lease starts, with possible starting points varying from the current time T to a job's maximum waiting time U_j.

The time slot search algorithm iterates over the vector (P_i) of predicted hourly price changes for each instance type i and computes the cost to run a job of a given estimated length (E_j).

At each fixed interval increment (k), the algorithm computes the cost of running job j over the price period $[P_i^k, P_i^k + E_j]$. It records the minimum cost obtained so far along with the respective instance type, period start (k), and the maximum price observed over the period, which will be used as the bid.

```
   input  : available instance types I
   input  : vectors of predicted price changes for all instances P_i
 1 C_min ← 100000
 2 foreach i ∈ I do
 3 |   E_j(i) ← estimated runtime of j on instance type i;
 4 |   k ← current time T;
 5 |   while k ≤ U_j do
 6 |   |   C_j ← cost of running job over period [P_i^k, P_i^k + E_j];
 7 |   |   if C_j < C_min) then
 8 |   |   |   C_min ← C_j;
 9 |   |   |   I_j ← i;
10 |   |   |   T_j ← k;
11 |   |   |   B_j ← maximum price observed over period [P_i^k, P_i^k + E_j];
12 |   |   k ← k + 300;
13 return (I_j, T_j, B_j)
```

Algorithm 2: Greedy algorithm that computes monetary cost of running a job j over multiple possible time slots

Finally, the result of the algorithm is the tuple (I_j, T_j, B_j), composed of the instance type chosen to run job (I_j), the new wait time until the job can start (T_j), and the bid itself (B_j). This activity is described in Algorithm 2.

5.3.2 Job Runtime Estimation

The time slot search algorithm relies on the runtime estimation of each job in order to find the spot of lowest predicted price in the interval between the job submission time and its maximum waiting time.

The runtime estimate E_j will determine the slot sizes that a job can fit, and combined with predicted spot prices on that slot, SpotRMS can determine the most economical slot to allocate the job.

Precise runtime estimation becomes even more valuable for the overall performance of SpotRMS. This section presents the results of applying the 5 proposed runtime estimation schemes previously discussed in chapter 3, namely 'Actual runtime", "Actual runtime with error", "User-Supplied", "Fraction of User-Supplied", and "Recent Average".

5.3.3 Bid Calculation

A single bidding strategy is used in this case. The bid computation process consists of selecting the maximum spot price observed over the time slot chosen by the algorithm as the most economical to run the job. This value is the maximum price predicted to be reached for the lifetime of the job. Therefore, the expectation of such a strategy is to completely avoid out-of-bid situations.

The assumptions in this case are that both the predicted job runtime and the predicted price levels are accurate. This implies that fault-tolerance mechanisms such as checkpointing are not required (or desired) due to their inherent overhead. For this reason, and to allow studying the effects of prediction-based strategies in isolation, for the experiments of this chapter, SpotRMS's fault-tolerance capabilities have been disabled. Nonetheless, should out-of-bid situations occur in practice, SpotRMS could apply the most effective mitigation strategies discussed in chapter 4.

5.3.4 Performance Evaluation

Simulations are used to evaluate the effectiveness of the prediction-based resource provisioning strategy. The experimental design is itself similar to that of the previous chapters, whereby SpotRMS's decision-making mechanisms used historical spot price information.

This section presents a comparison study of the effects of runtime estimation on the monetary cost metric. SpotRMS, equipped with prediction-assisted mechanisms, is compared with scenarios where prediction-assisted provisioning was not available. In the later case, SpotRMS was configured with the On-demand bidding strategy, and naturally, no future time-slot search was available.

Results depicted on Figure 5.2 indicate a clear advantage of using predictions. Positive effects on the cost metric are consistent over all runtime estimation methods. The "Recent Average" method provides more modest improvements because it already performed better in situations without the availability of prediction. The same observation is valid for the methods that use perfect information (e.g. "Actual runtime").

Estimation methods deemed to be less precise in previous results of chapter 3 ("User-Supplied" and "Fraction of User-Supplied"), when used in combination with

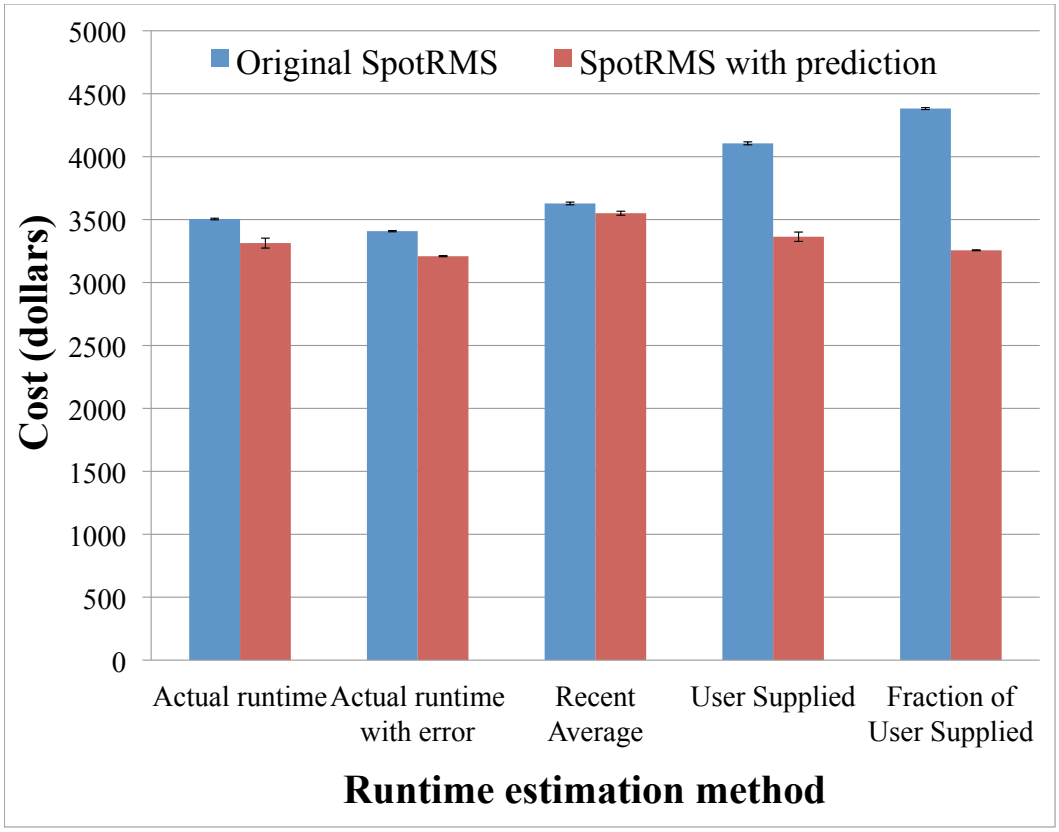


Figure 5.2: Effect of time slot search with spot price prediction on SpotRMS' performance: cost metric

prediction result in better costs. This indicates that the right combination of factors proves to be as important as in previous scenarios, and that the effect of prediction cannot be assumed to always improve cost equally across multiple estimation methods. The most benefit is obtained by using the "Fraction of User-Supplied" estimation, while when prediction was not available "Recent Average" was deemed as the ideal method. However, as explained in chapter 3, "Fraction of User-Supplied" assumes user estimates are overestimated by a certain amount, therefore relying on this fact for all jobs and workloads.

5.4 Effect of Price Volatility

As discussed, Amazon's spot prices do not vary significantly over time – a fact established by observing the history of spot prices from the periods studied in this work.

Table 5.3: Model parameters used to generate multiple volatility values

more/less volatile	Param 1	Param 2	Param 3
-30%	0.05	0.04	0.07
-20%	0.01	0.35	0.03
-10%	0.01	0.45	0.23
0% (original)	0.01	0.45	0.03
10%	0.01	0.45	0.52
20%	0.02	0.35	0.25
30%	0.15	0.44	0.52
40%	0.15	0.43	0.33
50%	0.18	0.42	0.54
60%	1.09	0.42	0.34
70%	0.12	0.41	0.31

This is an indication that the provider, at this time, possesses sufficient capacity to serve most requests without the need for price rises, even though users may choose to bid high. However, in a more contentious computational market, where providers operate more efficiently, consumers may decide to bid judiciously and as such a more volatile availability pattern would arise.

Similar to the volatility of stock prices in a share market, volatility represents a measure of risk. Therefore, the spot price volatility used here was based on a stock price volatility formula: namely the standard deviation of logarithmic returns over a period. The log return at time t, is computed according to equation 5.1.

$$x = \log\left(\frac{Q_t}{Q_{t-1}}\right) \tag{5.1}$$

Return is computed hourly, thus producing a set of n values, where n is the total number of hours over a period of 60 days of generated prices, i.e. 1440 values. The volatility (σ) is, therefore, computed as the standard deviation of the returns over this period (equation 5.2).

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{5.2}$$

In order to simulate scenarios of varying volatility, modifications were made to the model parameters to generate price variation patterns with a pre-defined volatil-

Table 5.4: Factors and their levels for volatility experiments

Factor	Possible values
Bidding Strategy	Mean, High, Current
α	2, 4
Fault tolerance mechanisms	None, Migration, Checkpointing

ity. As a baseline, the original model parameters for price (MoG) and inter-price time (exponential) for the US-WEST region were taken. The standard deviation (but not the mean) of the price distribution was steadily increased whilst the mean of the inter-price time distribution was steadily decreased, thus causing bigger price changes more often. Opposite changes to generate less volatile patterns were also simulated. The resulting model parameters are listed in Table 5.3.

5.4.1 Experimental Results

The effect of volatility on a variety of scenarios was evaluated, based on those devised in the previous chapters to evaluate SpotRMS. Specifically, 18 combinations of factors were selected, combined with 10 values of volatility, resulting in a total of 180 experiments. Table 5.4 lists the factors and their values used for this collection of experiments.

A selected number of factor combinations were chosen for the volatility experiments. Overall, the best performing factor combinations used in the previous chapters were included, considering the inherent trade-off between cost and deadline violations. Combinations of scenarios that were known to lead to high costs or high number of deadline violations were excluded, as they would invariably produce more deadline violations and higher costs given varying volatility.

A conservative (4) and an aggressive (2) urgency factors were also identified as the ones that led to fewer deadline violations and lower costs in previous results. Therefore, in this experiment we use those two values only.

The main objective of these experiments is to verify whether the behaviours observed in the previous chapters, obtained using real price patterns from Amazon EC2, still apply under volatile price patterns. It is important to understand if different bidding strategies and fault-tolerance mechanisms are more or less advantageous in volatile situations.

The expectation when designing these experiments is that fault-tolerance is needed for the more volatile scenarios, whereas for less volatile scenarios the overhead of these mechanisms would diminish their advantages. Nonetheless the migration mechanism, shown to impose less overhead than checkpointing could provide benefits even in cases where there is a low probability of out-of-bid situations.

In terms of cost, greater volatility may present obvious challenges, such as high price peaks. On the other hand, opportunities of using periods of lower prices also exist, provided that bids are kept relatively low and an efficient fault tolerance mechanism is present.

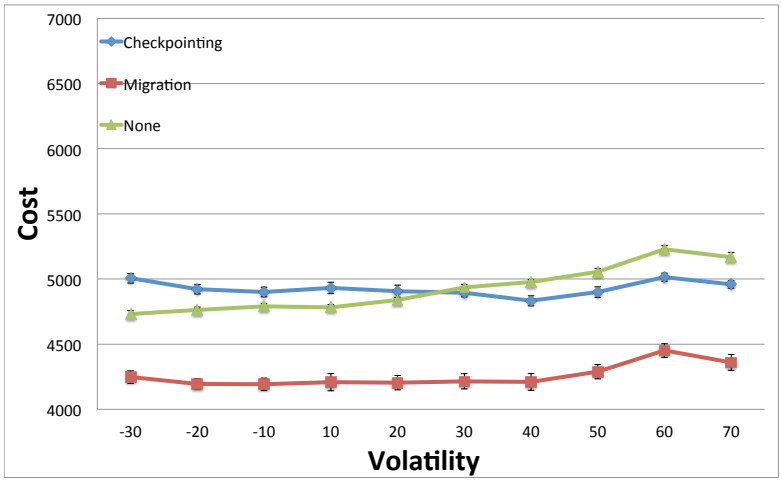
A number of observations may be derived from the results of the 180 volatility experiments.

The effect of volatility on the number of failures, and therefore in the cost of executing the workload without any fault tolerance technique is clear from these results, depicted in Figure 5.3. In most cases, more volatile scenarios lead to higher prices due to prolonged executions as multiple attempts to complete failed jobs are made.

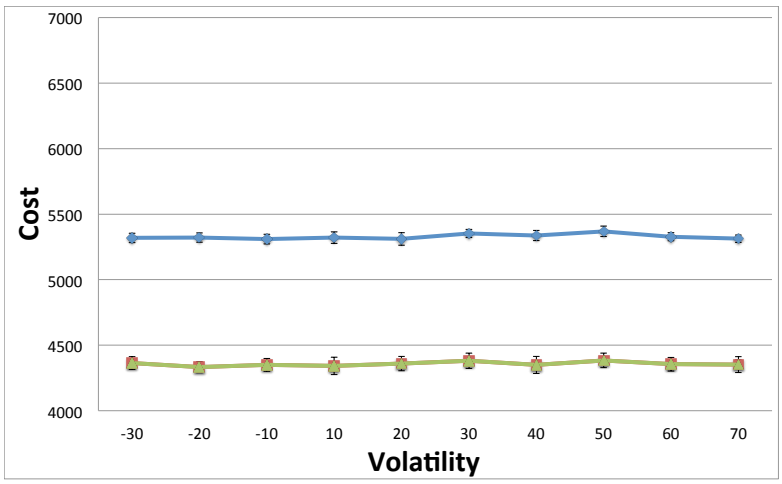
In almost all situations, the Migration fault tolerance technique helps the policy achieve lower overall cost, across both urgency factors, except when the High price bidding strategy is employed. Very high volatility modifiers ($> 50\%$) bring about slightly higher prices, due to more out-of-bid situations, therefore causing more recovery overhead. However, results are very similar for most other volatility values, indicating that the migration technique is capable of tolerating a range of failures with minimum overhead.

Results depicting the performance of checkpointing indicate the negative effects of overhead in low volatility scenarios (fewer failures/unnecessary checkpoints) and the positive effect of potential recoveries in more volatile cases (more failures/necessary checkpoints).

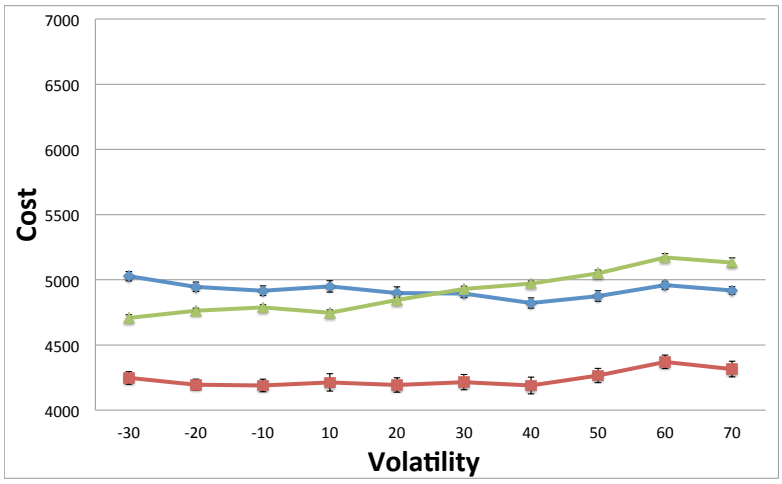
Moreover, in low volatility scenarios, prices remain mostly flat, leading to fewer opportunities of scheduling jobs on periods of lower prices, especially for short jobs. Regardless of the bid price, costs at any given slot tend to be close to the average price of the entire period.



(a) Current price bidding (aggressive urgency factor)

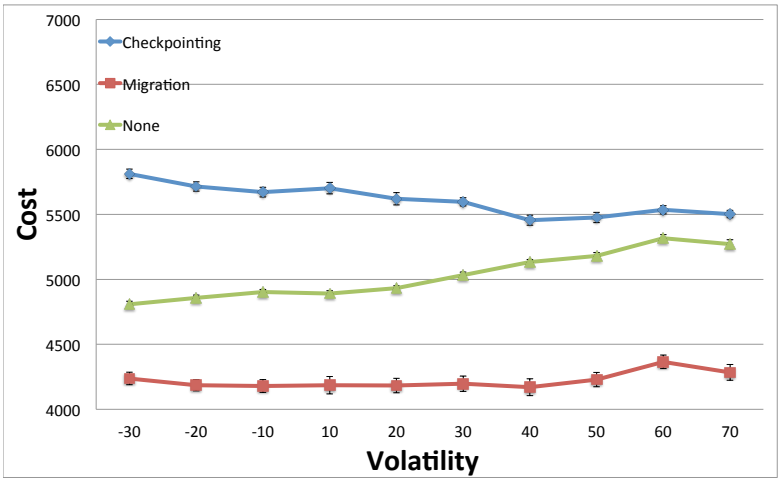


(b) High price bidding (aggressive urgency factor)

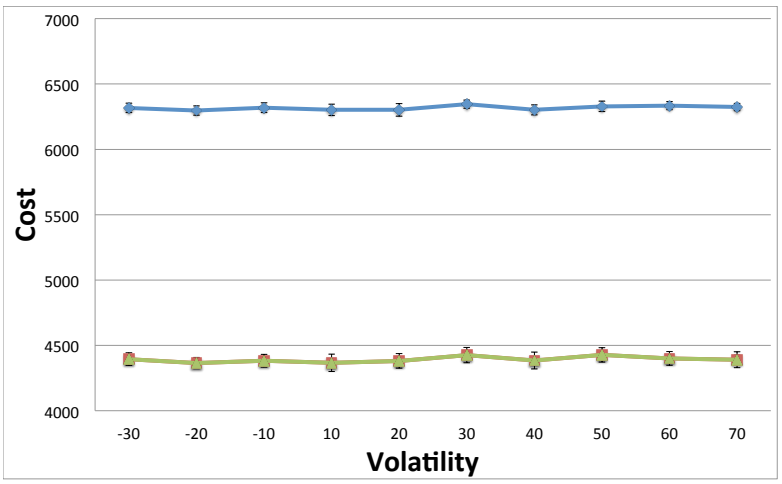


(c) Mean price bidding (aggressive urgency factor)

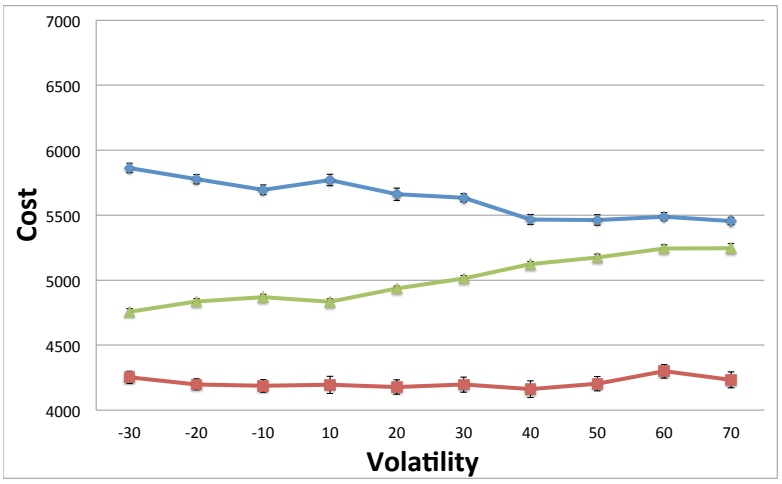
Figure 5.3: Effect of volatility under different bidding strategies and fault tolerance mechanisms, with an aggressive provisioning policy



(a) Current price bidding (conservative urgency factor)



(b) High price bidding (conservative urgency factor)



(c) Mean price bidding (conservative urgency factor)

Figure 5.4: Effect of volatility under different bidding strategies and fault tolerance mechanisms, with a conservative provisioning policy

High bidding leads to higher costs, but costs are not significantly affected by volatility. This fact is also an indication that, over the entire period, average prices are similar regardless of volatility. Furthermore, high bids cause SpotRMS to make use of both low and high price spots, whereas other policy combinations would favour lower price periods.

Overall, these results demonstrate that, depending on the combination of bidding strategy and fault tolerance mechanism used, volatility significantly affects how economically SpotRMS is able to provision resources.

5.5 Summary

In this chapter a model of spot prices was applied in two use cases: on prediction of future price variation patterns and use of this information to aid SpotRMS' mechanisms, and on volatility studies, where volatility of spot prices were decreased and increased by applying slight variations to model parameters.

Spot price prediction was shown to be largely beneficial to SpotRMS, but its combination with runtime estimation methods that performed well in previous experiments did not provide significantly superior results. This behaviour reinforces previous observations that less precise runtime estimation methods do not necessarily perform worse. In this sense, prediction provides more benefit when combined with less precise estimation methods.

Volatility had varying effects over the 180 factor combinations. High volatility has significant effect on prices unless low overhead fault tolerance is used. As expected, high bidding leads to higher costs, but ensures jobs get an average cost regardless of volatility.

Chapter 6

Viability of Live Migration of Virtual Machines as a Fault Tolerance Mechanism

Live virtual machine (VM) migration can bring many benefits such as improved performance, manageability, and especially fault tolerance, whilst allowing workload movement with a short service downtime. However, it is the case that service levels of applications are likely to be negatively affected during a live migration. For this reason, a better understanding of its effects on system performance is desirable. s In this chapter, the effects of live migration of VMs on the performance of applications running inside Xen-based VMs is evaluated. Results show that, in most cases, migration overheads are acceptable but cannot be disregarded, especially in systems where availability and responsiveness are governed by strict Service Level Agreements (SLAs). Despite this, there is a high potential for live migration applicability in cloud-based data centres, which serve a variety of workloads, most notably, modern Internet applications. The results presented are based on workloads covering the domain of multi-tier Web 2.0 applications. This choice was deliberate since the current research literature lacks a practical investigation of live VM migration effects on the performance of modern Internet applications, such as multi-tier Web 2.0 applications.

The primary focus of this chapter is a case study that quantifies the effect of VM live migration for a representative modern Internet application. This study is

applicable to environments where metrics, such as service availability and responsiveness, are driven by SLAs. In such systems, service providers and consumers agree upon a minimum service level and non-compliance to such agreement may subsequently incur penalties to providers [10]. More importantly, an SLA can directly reflect how end-users perceive the quality of service being delivered.

The quantification of live migration overhead is useful to understand the technique's viability as a fault tolerance mechanism to support SpotRMS' reliable provisioning of spot instances. In the same manner, the overhead identified in this chapter was used directly on the experiments of chapter 5 as a realistic quantification of the impact on jobs running in a migrating spot instance.

Specifically, this chapter introduces:

- A characterisation of modern Internet applications;
- A practical investigation of live VM migration effects on the performance of modern Internet applications;

The contents of this chapter are derived from a published research paper:

Voorsluys, W., Broberg, J., Venugopal, S., and Buyya, R. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *Proceedings of the 1st IEEE International Conference on Cloud Computing (CloudCom 2009)* (Berlin, Heidelberg, Sept. 2009), vol. 5931 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 1–12

6.1 Background and Motivation

VM technology has emerged as an essential building-block of modern data centres due to its capabilities of isolating, consolidating and migrating workload [11]. Altogether, these features allow a data centre to serve multiple users in a secure, flexible and efficient way. Consequently, virtualised infrastructures are considered a key component used to drive the emerging Cloud Computing paradigm [17].

Migration of VMs can be used to improve the manageability, performance and fault tolerance of systems. VM migration in a production system can help balance system load, e.g. by migrating VMs out of overloaded/overheated servers, and/or

to selectively bring servers down for maintenance after migrating their workload to other servers.

The ability to migrate an entire operating system overcomes the difficulties that make process-level migration a complex operation [26, 68]. The applications themselves and their corresponding processes do not need to be aware that a migration is occurring. Popular hypervisors, such as Xen and VMWare, allow migrating an OS as it continues to run. Such a procedure is termed as "live" or "hot" migration, as opposed to "pure stop-and-copy" or "cold" migration, which involves halting the VM, copying all its memory pages to the destination host and then restarting the new VM. The main advantage of live migration is the possibility to migrate an OS with near-zero downtime, an important feature when live services are being served [26].

6.1.1 Migration Techniques

Clark et al. [26] identify that live migrating a VM (on Xen) consists of transferring its state, i.e. its memory contents, from a source server to a destination server. To live migrate a VM, the hypervisor pre-copies memory pages of the VM to the destination without interrupting the OS or any of its applications. The page copying process is repeated in multiple rounds in which dirty pages are continuously transferred. Normally, there is a set of pages that is modified so often that the VM must be stopped for a period of time, until this set is fully transferred to the destination. Subsequently, the VM can be resumed in the new server.

It has been observed that live migration of VMs allows workload movement with near zero application downtime. Nevertheless, the performance of a running application is likely to be negatively affected during the migration process due to the overhead caused by successive iterations of memory pre-copying [26]. For the duration of the pre-copying process extra CPU cycles are consumed on both source and destination servers. An extra amount of network bandwidth is consumed as well, potentially affecting the responsiveness of Internet applications. Furthermore, as the VM resumes after migration, a slowdown is expected due to cache warm-up at the destination [70].

Downtime and application performance are also likely to be affected in different ways for different applications due to varying memory usages and access patterns. Previous studies have found that actual downtime may vary considerably between

applications, ranging from as low as 60ms when migrating a Quake game server [26] to up to 3 seconds in case of particular HPC benchmarks [70]. Regarding the overhead due to migration activity, earlier studies have shown that slowdown ranged between 1% and 8% of wall-clock time for a particular set of HPC benchmarks [70].

In other scenarios using Xen, a 12% to 20% slowdown on the transmission rate of an Apache Web server running a VM with 800MB of memory and serving static content was reported [26]. In the case of a complex Web workload (SPECWeb99), the system under test could was able to meet benchmark metrics [26]. In all cases, it has been concluded that, for the particular set of applications considered, the bad effects of migration were acceptable and/or negligible in contrast to the potential benefits of system fault tolerance [70].

6.1.2 Characteristics of Target Applications

The domain of applications that can potentially take advantage of the Infrastructure as a Service (IaaS) paradigm is broad. For instance, Amazon [95] reports several case studies that leverage their EC2 platform, including video processing, genetic simulation and Web applications.

Modern Internet Applications

Multi-tier Web applications generally include a static content Web server (e.g. Apache), an application server/dynamic content generation layer (e.g. PHP, Java EE, Ruby on Rails), and a backend database (e.g. MySQL, Oracle, CouchDB). Virtual machine technology adds extra flexibility to scaling of such Web applications, by allowing dynamic provisioning and replication of VMs to host additional instances for one of the application tiers.

Social networking websites are perhaps the most notable example of highly dynamic and interactive Web 2.0 applications which have gained popularity over the past few years. Their increasing popularity has spurred demand for a highly scalable and flexible solution for hosting applications. Many larger sites are growing at 100% a year, and smaller sites are expanding at an even more rapid pace, doubling every few months [88]. These web applications present additional features that make them different from traditional static workloads [83]. For instance, their social networking features make each users' actions affect many other users, which makes static load partitioning unsuitable as a scaling strategy. In addition, through blogs, photostreams

and tagging, users can now publish content to one another rather than just consume static content.

Altogether, these characteristics present a new type of workload with particular server/client communication patterns and that can dynamically impact on server load. However, most available performance studies use extremely simple static file retrieval tests to evaluate Web servers, often leading to erroneous conclusions [88]. This work has taken this trend into account during the workload selection process, resulting in the selection of the Olio web application [5] as the basis for generation of a realistic workload.

6.2 Evaluation of Live Migration Cost

This study aims at achieving a better understanding of live migration effects on modern Internet applications. Benchmarking experiments have been designed to evaluate the effect of live migration on a realistic Web 2.0 application hosted on networked VMs.

6.2.1 Testbed Specifications

The testbed used is a group of 6 servers (1 head-node and 5 virtualised nodes). Each node is equipped with Intel Xeon E5410 (a 2.33 GHz Quad-core processor with 2x6MB L2 cache and Intel VT technology), 4 GB of memory and a 7200 rpm hard drive. The servers are connected through a Gigabit Ethernet switch. The head-node runs Ubuntu Server 7.10 with no hypervisor. All other nodes (VM hosts) run Citrix XenServer Enterprise Edition 5.0.0. The choice for a commercial hypervisor was based on the assurance of an enterprise class software in accordance with the needs of target users, i.e. enterprise data centres and public application hosting environments.

All VMs run 64-bit Ubuntu Linux 8.04 Server Edition, para-virtualised kernel version 2.6.24-23. The installed web server is Apache 2.2.8 running in pre-fork mode. PHP version 5.2.4-2 is used with MySQL with Innodb engine – version 5.1.32.

6.2.2 Workload

Olio [5] was used as the Web 2.0 application, combined with the Faban load generator [89] to represent an application and workload set. Olio is a Web 2.0 toolkit that helps

developers evaluate the suitability, functionality and performance of various Web technologies devised by Sun Microsystems from its understanding of the challenges faced by Web 2.0 customers [88]. It has been successfully deployed and evaluated in several reasonably sized high-end server infrastructures [88], as well as in rented resources from Amazon EC2 [83].

The Olio Web application represents a social-events website that allows users to perform actions such as loading the homepage, logging into the system, creating new events, attending events and searching for events by date or tag. It provides implementations using three technologies: PHP, Ruby on Rails and J2EE. For the experiments given here, the Olio's PHP implementation was used, including the popular LAMP stack (Linux Apache MySQL PHP).

Faban is an open-source Markov-chain load generator that can be used to drive load against Olio. It is composed of a master program which spawns one or more load drivers, i.e. multi-threaded processes, that simulate actual users. The master presents a Web interface through which it is possible to submit customised benchmark runs and monitor their results. This Olio/Faban combination was originally proposed as part of the Cloudstone benchmark [83].

The load of a given application may be varied by changing the number of concurrent users to be served by the application. The total time for each run is configured by adjusting three different durations, namely: ramp-up; steady state, and ramp-down. Resulting metrics reported by Faban only take into account the steady state period.

The main metric considered in the experiments given here is related to an SLA defined in Cloudstone. Specifically this SLA defines minimum response times for all relevant user actions. Thus, at any 5-minute window, if a certain percentile of response times exceeds the maximum, an SLA violation is recorded. The 90th and 99th percentiles are considered in this study, representing a more relaxed and a stricter SLA, respectively. Table 6.1 lists the details of the SLA.

6.2.3 Benchmarking Architecture

The architecture of the benchmarking setup is depicted in Figure 6.1. Based on the observation that MySQL tends to be CPU-bound when serving the Olio database, whereas Apache/PHP tends to be memory-bound [83], the system under test (SUT)

Table 6.1: Cloudstone's SLA: The 90th/99th percentile of response times measured in any 5-minute window during a steady state should not exceed the following values (in seconds):

User action	SLA	User action	SLA
Home page loading	1	User login	1
Event tag search	2	Event detail	2
Person detail	2	Add person	3
Add event	4		

splits the workload into two networked VMs, hosted on different servers. This allows to better partition the available physical resources.

All nodes share a Network File System (NFS) mounted storage device, which resides in the head-node and stores VM images and virtual disks. In particular, a local virtual disk is hosted in the server that hosts MySQL.

The load itself is driven from the head-node, where the multi-threaded workload drivers run together with Faban's master component.

6.2.4 Experimental Design

The overall objective of these experiments is to quantify the slowdown and downtime experienced by the (executing) application when VM migrations are performed. Specifically, the experiment aims to quantify the application slowdown based on values reflected in the above-mentioned SLA.

In all experiments, the servers and their interconnection are dedicated to the application under test. A migration experiment consists of migrating a single VM between two dedicated physical machines. In each run, the chosen destination machine was different from the source machine in the previous run, i.e. a series of runs did not consist of migrating a VM back and forth between the same two machines.

Preliminary Benchmarking of Experiments

Initially the application ran without performing any VM migration. The load against Olio was gradually increased by increasing the number of concurrent users between runs (by 100 user increments), whilst using 2 identically sized VMs containing 2 vCPUs and 2GB of memory, running on distinct Xen Server nodes. By analysing the SLA (both 90th and 99th percentile of response times for all user actions), it was

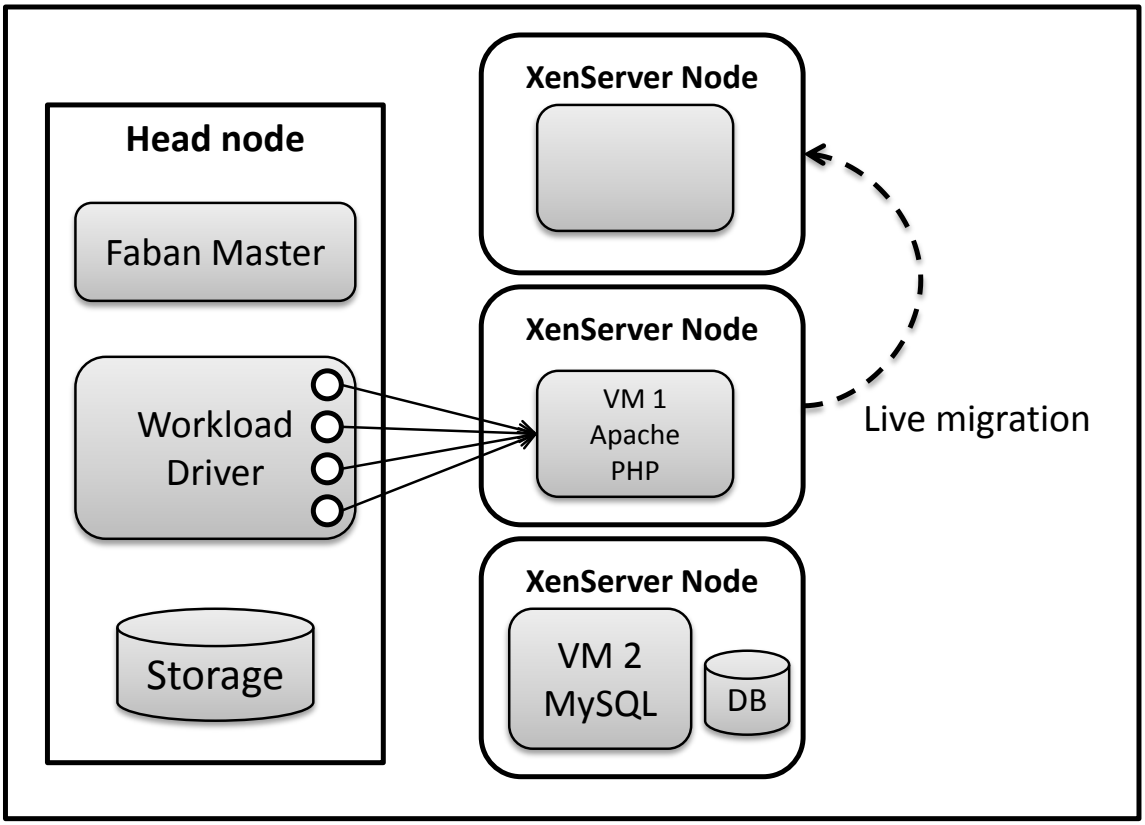


Figure 6.1: Benchmarking architecture

found that up to 600 concurrent users could be served by the SUT. The memory and CPU usage was measured to find the minimum VM sizes capable of serving 600 users. The vCPUs and memory usage was then set to the minimum required to serve 600 users. In the final configuration, the first VM hosted Apache/PHP and had 1 vCPU with 2GB of memory, whilst the second VM hosted MySQL and had 2 vCPUs with 1GB of memory each.

However in the preliminary experiments, performance issues arose when hosting the MySQL server on NFS. Specifically the application would not scale to more than 400 concurrent users, which lead to hosting the MySQL database on a local disk, thus scaling up to 600 concurrent users. For this reason, the experiments do not include migrating the VM hosting the database server, since XenServer requires all storage devices to be hosted in a network storage in order to perform live migrations.

Migration Experiments

In the first set experiments with Olio, 10-minute and 20-minute benchmark runs were performed with 600 concurrent users. During these experiments, live migration of the Web server VM was performed. The goal of this experiment was twofold: to evaluate how the pre-defined SLAs could be violated when the system was nearly oversubscribed, but not overloaded, and to quantifying the duration of migration effects and potential downtime experienced by the application.

In a second round of experiments, the benchmark was run with smaller numbers of concurrent users, namely 100, 200, 300, 400 and 500. This experiment aimed at finding a "safe" load level upon which migrations could be performed at lower risks of SLA violation, especially when considering the more stringent 99th percentile SLA.

6.3 Migration Experimental Results

Overall, the experimental results show that the overheads due to live migration are acceptable but cannot be disregarded, especially in stringent SLA-oriented environments requiring more demanding service levels.

Figure 6.2 shows the effect of a single migration performed after five minutes in a steady state of one run. A downtime of 3 seconds is experienced near the end of a 44 second migration. The highest peak observed in response time takes place immediately after the VM resumes in the destination node; 5 seconds elapse until the system can fully serve all requests received during the downtime. In spite of this, no requests were dropped or timed out due to the application's downtime. The downtime experienced by Olio when serving 600 concurrent users was well above the expected millisecond level, previously reported in the literature for a range of workloads [26]. This result suggests that workload complexity imposes an unusual memory access pattern, increasing the difficulty of live migrating the VM.

Figure 6.3 presents the effect of multiple migrations on the homepage loading response times. These results correspond to the average of 5 runs and report the 90th and 99th percentile SLAs. As can be seen, the more stringent 99th percentile SLA is violated a short moment after the first migration is performed indicating that when 600 concurrent users are being served, a single VM migration is not ac-

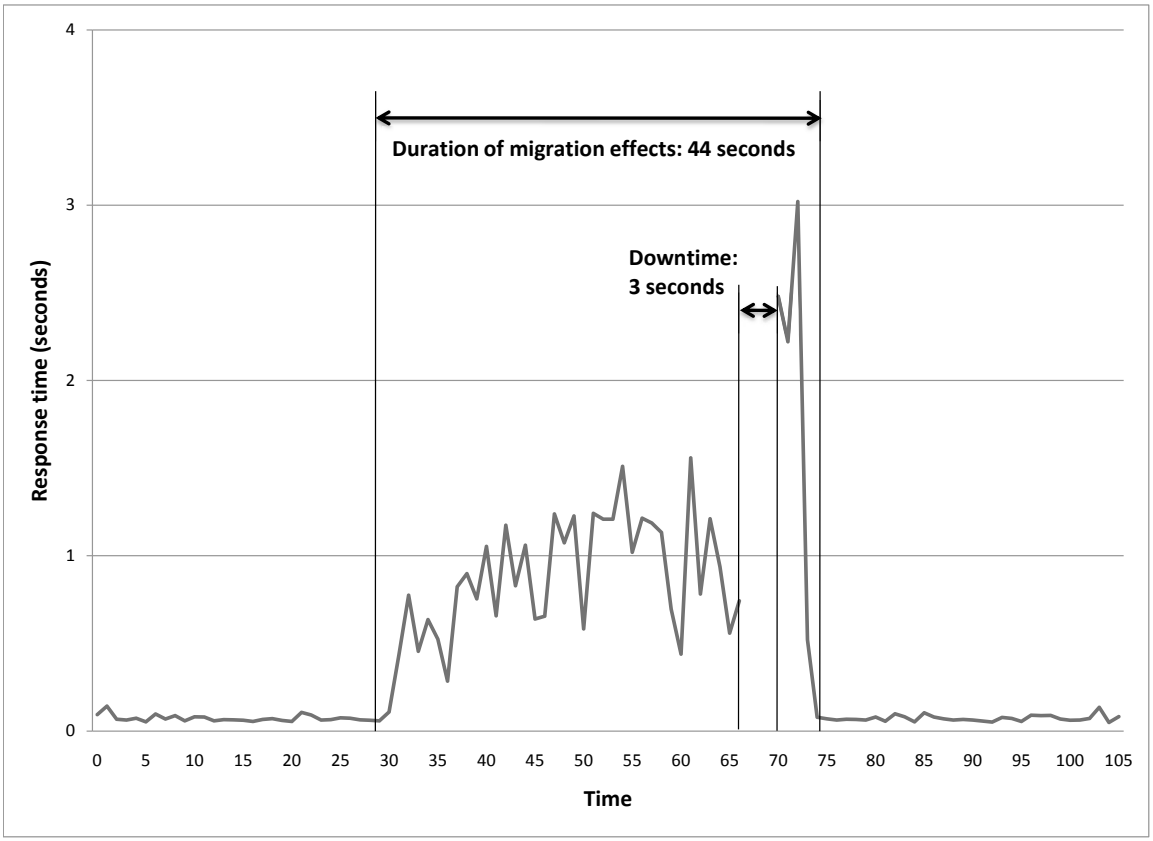


Figure 6.2: Effects of a live migration on Olio's homepage loading activity

ceptable. The 90th percentile SLA is not violated when a single migration occurs, but is violated when two migrations are performed in a short period of time. This figure also indicates that more than one migration might not cause violation of the 90th percentile SLA. A way of preventing such violation is by allowing sufficient spacing between migrations in order to allow the SLA formula to generate normal response time levels. Thus, it is paramount that this information is employed by SLA-oriented VM-allocation mechanisms with the objective of reducing the risk of SLA non-compliance in situations when VM migration is inevitable.

From the above results it can be concluded that despite a significant slowdown and downtime caused by live migration of VMs, the SUT is resilient to a single migration when the system responsiveness is governed by the 90th percentile SLA. In other words, provided that migrations are performed at correct times, there is no cost associated with them. However, this is not the case for the 99th percentile SLA. For this reason, a new series of experiments was performed with a smaller number

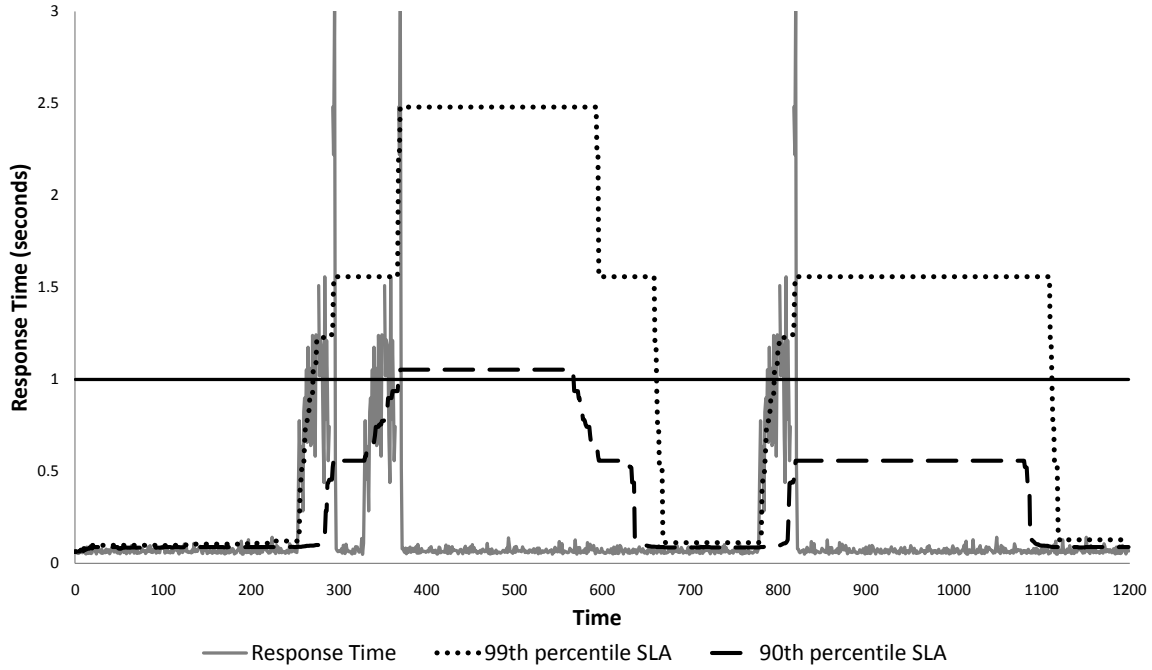


Figure 6.3: 90th and 99th percentile SLA computed for the homepage loading response time with 600 concurrent users. The maximum allowed response time is 1 second

of concurrent users. The objective of such experiments is to gauge a safe level upon which a migration could be performed with low risk of SLA violation.

Table 6.2 presents more detailed results listing maximum response times for all user actions as computed by the 99th percentile SLA formula when one migration was performed in the middle of a 10 minute run. In these runs the load varies from 100 to 500 users. In all cases, the SUT was able to sustain an acceptable performance even in the presence of a live migration of the Web server. For instance, the maximum value observed for loading the homepage was 0.32 seconds, which corresponds to approximately 1/3 of the maximum value allowed, i.e. 1 second. The maximum value observed for adding a new person to the system was 2.28 seconds – more than half of the maximum allowed, but this still does not pose a risk to SLA violation. These results indicate that a workload of 500 users is the load level at which a live migration of the Web server should be carried out (e.g. to a lesser loaded server) in order to decrease the risk of SLA violation.

Table 6.2: Maximum recorded 99th percentile SLA for all user actions when one migration is performed for 500, 400, 300, 200 and 100 concurrent users

Action	500	400	300	200	100
HomePage	0.32	0.18	0.25	0.25	0.13
Login	0.32	0.33	0.42	0.28	0.14
TagSearch	0.46	0.32	0.35	0.39	0.29
EventDetail	0.48	0.27	0.22	0.24	0.14
PersonDetail	1.53	0.62	0.69	0.61	0.32
AddPerson	2.28	1.00	1.51	1.73	0.66
AddEvent	2.26	1.02	1.30	1.81	0.98

6.4 Summary

Live migration of VMs is a useful capability for data centres. It allows more flexible management of available physical resources by making it possible to load balance and undertake infrastructure maintenance without entirely compromising application availability and responsiveness.

A series of experiments was performed to evaluate the cost of live migration of VMs in a scenario where a modern Internet application was hosted on a set of VMs. Live migration experiments were carried out in scenarios where several levels of load were driven against the application.

The results show that in an instance of a nearly oversubscribed system (serving 600 concurrent users), live migration causes a significant downtime (up to 3 seconds) – a larger value than expected based on results previously reported in the literature for simpler, non Web 2.0 workloads. This service disruption causes a pre-defined SLA to be violated in some situations, especially when two migrations are performed in a short period of time. On the other hand, the most stringent SLA (99th percentile) can still be met when migrations are performed when the system load is slightly decreased to fewer concurrent users, by approximately 20%, i.e. 500 in the above case study.

Chapter 7

Conclusions and Future Directions

Cloud computing technology has been a driving force in the switch from in-house, single-user computing servers to multi-tenant, centrally hosted computing power, delivered as a service. The reliable delivery of cloud computing services has materialised in recent years especially due to adoption of this model by industry players such as Amazon, Microsoft, and Google. Full-fledged markets of computing, storage, and application services have been set up giving consumers significant advantages and challenges. In particular, the cloud computing spot market, as introduced by Amazon's EC2 service, has made price variation and its effects into primary concerns in the process of resource provision and task scheduling.

This thesis has investigated the effects of variable resource pricing in the task of building dynamic clusters of cloud resources for the purpose of running computational jobs. SpotRMS, a complete architecture and system for resource management and scheduling was presented and the performance of its policies and mechanisms was evaluated. To conclude this work, we present here a synthesis of this thesis' findings with a focus on how they address the following objectives, previously set out in section 1.4:

- Identifying challenges and potential advantages of using dynamically priced resources in the task of resource management and scheduling in cloud computing;

- Understanding the impact of out-of-bid situations on application execution, especially regarding time and cost constraints;

- Evaluating bidding strategies designed to prevent failures and incorporating fault-tolerance mechanisms to mitigate their effect;

- Analysing the effectiveness of the proposed mechanisms in a variety of scenarios, including studying the effect of price volatility;

- Evaluating the potential overheads of fault-tolerance mechanisms on the performance of applications;

7.1 Using Dynamically-priced Cloud Resources

This study aimed at identifying challenges that the use of variable priced resources imposes on consumers of these resources, i.e. the end-users who utilise spot instances to perform computational tasks. Consumers aim at completing their tasks at the lowest possible cost and within an acceptable time frame. The design and evaluation of SpotRMS and its associated mechanisms has provided insights into those challenges.

The most notable challenge is that of price variation itself, characterised by unpredictable price levels and time intervals between changes. In the absence of the prediction techniques introduced in chapter 5, the effect of variations was found to be a significant factor that prevented users of achieving low costs and meeting job deadlines.

Identifying approximate lower and upper price bounds, by analysing history of spot price variations, was found to be essential in devising the bidding strategies introduced in chapter 4 (section 4.2).

The lower bound for an instance type spot price is the reserve price set by the cloud provider. Even though cloud providers do not publish their reserve prices, they could be determined by simply noting a pattern regarding the minimum historic spot price. For Amazon EC2 instance, the reserve price was found be about 1/3 of the on-demand price. This price level can be observed for extended periods of time.

In terms of an upper bound, it was observed that prices very rarely exceeded the on-demand price, an expected behaviour, given the clear incentive of bidding at or below the on-demand price, based on the fact that an instance of identical type can be obtained at fixed price from the cloud provider.

In view of price range information, bidding strategies were devised taking into account the risk and rewards of bidding at particular levels. For example the "Minimum" strategy aimed at identifying the lower possible price at a given period and bidding at just above that value, thus restricting provisioning of resources to favourable periods. That particular strategy, however, performed poorly in the absence of a fault-tolerance mechanism, due to more frequent loss of work linked to out-of-bid situations. On the other hand, strategies that yielded higher bid values avoided most failures, thus leading to lower costs and deadline violations.

Therefore, several mechanisms to manage the price/deadline violations trade-off were identified, developed, and evaluated. However, finding the right combination of these mechanisms was identified as a challenge. Nonetheless, experimental results of chapters 4 and 5 were helpful in this sense, bringing up ways of transforming challenges into advantages.

7.2 Impact of Out-of-bid Situations

Out-of-bid situations occur whenever the current spot price goes above the user's maximum bid. They cause spot instances to be terminated by the provider without prior notice to the end user, resulting in total loss of running jobs.

Some experiments designed in this work aimed at quantifying the effect of spot price variation on the deadline violations metric. Ultimately this metric indicates job usefulness, making it especially sensitive to situations that cause loss of work. Except in experiments where the provisioning process of SpotRMS only made use of high bids, out-of-bid situations played a significant role in causing deadline violations.

Factor combinations devised to achieve lower costs, such low bids and setting the value of the urgency modifier (α) to aggressive values (≤ 4) resulted in more deadline violations. SpotRMS' aggressive provisioning fashion aimed at postponing jobs as much as possible to maximize reuse of instances and obtain lower spot prices. The resulting deadline violations were attributed to situations when SpotRMS postponed jobs until it was necessary to have resources available immediately, i.e. when the urgent factor U_j approached 0. In this scenarios out-of-bid situations were even more detrimental.

By managing the impact of out-of-bid situations that result from low bids it is possible to obtain low costs while keeping deadline violations under control. This has been achieved to some extent with judicious bidding strategies, and primarily when fault-tolerance mechanisms, such as checkpointing and migration were introduced in 5.

7.3 Pairing Bidding Strategies and Fault Tolerance Mechanisms

The main conclusion drawn from experimental results from chapter 5 is that bidding strategies alone cannot bring about the advantages of using spot market resources. It was initially identified that submitting high bids is a good way of achieving cheaper cost and fewer deadline violations. The "On-demand" strategy, which uses the published on-demand price as the bid has consistently performed the best. With this strategy, SpotRMS was able to avoid failures that would otherwise have happened due to minor price increases and also avoid incurring the cost of high prices above the on-demand price. Bidding low, contrary to what would be expected led to higher cost due to loss of work caused by out-of-bid situations.

Fault tolerance techniques are helpful to mitigate out-of-bid situations. Results demonstrated positive effects of fault tolerance mechanisms when paired with bidding strategies that submitted low bids and aggressive urgency factor. For instance, the migration technique performed better when paired with the Minimum and Current bidding strategies, given that it provides superior mitigation, thus tolerating frequent failures cause by using low bids.

Although able to similarly tolerate failures, checkpointing was shown to impose more overhead, which in turn impacted its effectiveness, especially in the cost metric. Nonetheless, checkpointing benefited from being paired with bidding strategies that produced higher bid values, as failures still occurred, albeit less frequently.

7.4 The Effect of Price Volatility

By analysing the history of spot prices used as traces for simulations of this work, it could be observed that, even though prices vary over time, their volatility is relatively low. For this reason, new experimental scenarios were devised to test SpotRMS

performance when faced with significantly volatile price profiles. The objective of these new scenarios was to observe if all the advantages brought by SpotRMS still held true if greater or lower volatility was introduced to price traces.

Several new experimental scenarios were performed, simulating 10 different volatility levels. When faced with high price volatility the use of fault-tolerance mechanism was shown to be essential in enabling the use of spot resources.

7.5 Benefits of Prediction-assisted Provisioning

As discussed, judicious bidding leads to considerably lower costs provided that out-of-bid situations are avoided or mitigated. However, mixed results were obtained when employing bidding strategies alone, without assistance of fault tolerance mechanisms. For this reason, this work also evaluated a prediction assisted time slot search heuristic that seeks to calculate bids more precisely and allocate jobs to the most economical time slots.

Predictions were made using a statistical model that captures the behaviour of spot prices in cloud computing data centres, through a mixture of Gaussian distribution that generates time-independent price patterns and an exponential distribution that generates time intervals between price changes. All distribution parameters were based on the price variation patterns of an actual Amazon data centre.

The hypothesis in that case was that prediction of future pricing conditions offers advantages over previous approaches, which relied on historic price information. Indeed, through the application of job runtime estimations, the time slot search heuristic was able to place jobs on spot instances significantly more efficiently than previous approaches. Across all runtime estimation techniques, there was significant improvement in the cost metric.

7.6 Future Directions

It is expected that the research subject of this thesis will continue to evolve at a fast pace, along with the constant innovations of the cloud computing field. Therefore, this section outlines practical steps to advance this research, as well practically apply this thesis' development and findings on real-world software technologies.

7.6.1 Simulation of Additional Workloads

The workload trace used in the works of chapters 3, 4, and 5, was obtained from the LHC Grid at CERN, through the Parallel Workloads Archive [34]. It was composed of grid-like embarrassingly parallel tasks.

While that workload provided a representative set of high-performance computing jobs, we intend to expand the workload representativeness to include business logic components of modern Internet applications, MapReduce jobs, and workflows.

In this direction, it will be necessary to consider other instance types, in addition to the CPU-optimised ones used in the experiments of this thesis. At present, Amazon EC2 offer various other types of instances suitable to varying application performance profiles, including memory-optimised, storage-optimised and GPU instances.

7.6.2 Spot Market Extension to OpenStack

A system model of a cloud provider spot market has been implemented into Spotsim to support simulations of SpotRMS' mechanisms. We intend to transport an extended system model into the OpenStack architecture, particularly augmenting its "Ceilometer" module to allow accounting and billing of variable pricing resources. However, the pricing mechanism used in this thesis was trace-based, thus only useful for simulations. Therefore, this component will need to extended.

As discussed in chapter 4, the pricing scheme thought to be used by Amazon is a generalisation of the Vickrey model for multiple divisible goods – the standard uniform price auction upon which the provider assigns resources to users starting with the highest bidder until all bids are satisfied or there are no more resources. The price paid by all users is the value of the lowest winning bid (sometimes, the highest non-winning bid) [109]. The planned OpenStack extension can include a Vickrey-auction pricing mechanism.

7.6.3 General Purpose Broker Service

Similarly to the spot market system model, the brokering architecture of SpotRMS, described in section 3.2 has been implemented into Spotsim and contains actual logic to perform spot instance provisioning and job execution on spot instances. This logic

will be basis of general purpose broker that will be implemented as Web-service, aimed at brokering access to, but not limited to, Amazon EC2 spot market.

The broker service will be built using modern enterprise technologies aimed at producing a production-level, scalable service. Technologies that are suitable to such implementation include:

- Reactive architecture: the Reactive Manifesto [1] conceptually describes a set of paradigms (reactive traits) that are paramount for the development of responsive, scalable, event-driven, and resilient applications. These concepts, when applied to our brokering software architecture will enable a system that reacts to high-load traffic conditions, can tolerate failures of its own components, and can itself be run on spot instances.

- RESTful web services API: The broker service programmatic interface will follow Representational State Transfer (REST) [35], a modern architectural style.

- Cloud provider agnostic provisioning: Generic cloud provisioning libraries such as Apache jClouds [2] will allow the broker to provision resources from multiple cloud providers, thus providing ample to access to compute services such as Amazon Web Services, Rackspace, and Joyent.

7.6.4 Refinement of Fault-tolerance Techniques to Reduce Overhead

The fault-tolerance techniques introduced in chapter 4 were able to positively contribute to achieve lower costs and better adherence to deadlines throughout most experimental scenarios. However, the performance of techniques such as checkpointing and job duplication were at times hindered by their imposed overhead.

We plan we tackle that limitation by applying more judicious decisions to avoid an overly cautions policy. In production environments where application performance requirements are regulated by varying types of Service Level Agreements (SLAs), SpotRMS could selectively apply fault-tolerance operations, aiming at satisfying particular SLAs. In this thesis, SpotRMS' heuristics was strict, aiming at

[1] http://www.reactivemanifesto.org/
[2] http://jclouds.apache.org/

obtaining the lowest possible cost and preventing all deadline violations. In this sense, we plan to investigate the effect of adhering to less stringent SLAs.

7.6.5 Policies for Using a Mix of On-demand and Spot Instances

In this work, SpotRMS was specialised in assembling virtual clusters composed solely of spot instances. We contend that in some situations, using a mix of on-demand and spot instances would be beneficial. For example, additional reliability could be provided by running urgent jobs on those fixed price instances, on which case an additional cost would be justified. SpotRMS can be easily extended to manage the provision of on-demand instances. In this sense, we plan to develop new policies and mechanisms and compare their effectiveness to this work's approach.

7.6.6 Enhanced Prediction Techniques

We intend to extend the work done on spot price and availability predictions by employing mathematical modelling techniques for prediction correction and improved certainty, such as the use of stochastic tasks and sensitivity analysis.

7.6.7 Evaluations with Other Workload Traces

Newer workload traces, such as Google Cluster Data [29] have recently become available. These traces contain workload profiles of large-scale MapReduce jobs over periods of several months. These traces may more closely reflect modern cloud computing workloads than the grid workload trace used in this work. Therefore, we plan to also evaluate the effectiveness of our techniques using those traces.

7.6.8 Evaluation of Fault-tolerance Mechanisms Under Varying Application Loads

As discussed in chapter 6, migration time of virtual machines is affected by the characteristics of running applications. In order to design load sensitive migration techniques, we plan to further evaluate the fault tolerance mechanisms proposed in chapter 4 to include workload profiles that include varying load patterns.

References

[1] Akoush, S., Sohan, R., Rice, A., Moore, A. W., and Hopper, A. Predicting the performance of virtual machine migration. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on* (2010), IEEE, pp. 37–46.

[2] Amazon Inc. Amazon EC2 SLA, 2012.

[3] Andrzejak, A., Kondo, D., and Yi, S. Decision Model for Cloud Computing under SLA Constraints. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (Aug. 2010), INRIA, IEEE, pp. 257–266.

[4] Apache Software Foundation. Apache CloudStack, Open Source Cloud Computing.

[5] Apache Software Foundation. Olio. http://incubator.apache.org/olio.

[6] Apparao, P., Iyer, R., Zhang, X., Newell, D., and Adelmeyer, T. Characterization & analysis of a server consolidation benchmark. In *Proceedings of the 4th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments - VEE '08* (New York, New York, USA, Mar. 2008), ACM Press, p. 21.

[7] Appistry. Cloud Platforms vs. Cloud Infrastructure. Tech. rep., Appistry, 2009.

[8] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., and Stoica, I. Above the clouds: A berkeley view of cloud computing. Tech. rep., Technical Report

UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[9] Ausubel, L. M., and Milgrom, P. The Lovely but Lonely Vickrey Auction. In *Combinatorial Auctions*, P. Cramton, Y. Shoham, and R. Steinberg, Eds. MIT Press, 2006, ch. 1, pp. 17–40.

[10] Barbosa, A. C., Sauvé, J., Cirne, W., and Carelli, M. Evaluating architectures for independently auditing service level agreements. *Future Generation Computer Systems 22*, 7 (Aug. 2006), 721–731.

[11] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03* (New York, New York, USA, 2003), ACM Press, p. 164.

[12] Ben-Yehuda, O. A., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. Deconstructing Amazon EC2 Spot Instance Pricing. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science* (Nov. 2011), IEEE, pp. 304–311.

[13] Broberg, J., Venugopal, S., and Buyya, R. Market-oriented grids and utility computing: The state-of-the-art and future directions. *Journal of Grid Computing 6*, 3 (Dec. 2007), 255–276.

[14] Buyya, R., Abramson, D., and Venugopal, S. The Grid Economy. *In Proceedings of the IEEE 93*, 3 (Mar. 2005), 698–714.

[15] Buyya, R., Pandey, S., and Vecchiola, C. Cloudbus Toolkit for Market-Oriented Cloud Computing. In *Proc. of the First International Conference on Cloud Computing (CloudCom 2009))* (Beijing, China, 2009), M. G. Jaatun, G. Zhao, and C. Rong, Eds., vol. 5931 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 24–44.

[16] Buyya, R., Ranjan, R., and Calheiros, R. N. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In *Proceedings of the 2009 International Conference on High Performance Computing & Simulation* (June 2009), IEEE, pp. 1–11.

[17] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems 25*, 6 (2009), 599–616.

[18] CA Technologies. CA AppLogic Cloud Platform.

[19] Carr, N. *The Big Switch.* W.W. Norton & Company, 2008.

[20] Casazza, J. P., Greenfield, M., and Shi, K. Redefining server performance characterization for virtualization benchmarking. *Intel Technology Journal 10*, 3 (2006), 243–251.

[21] Catlett, C. The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)* (2002), Ieee, pp. 8–8.

[22] Cherkasova, L., and Gardner, R. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. In *ATEC '05: Proceedings of the USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), USENIX Association, pp. 24–24.

[23] Chohan, N., and Castillo, C. See spot run: using spot instances for mapreduce workflows. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (2010), USENIX Association, pp. 7–14.

[24] Cirne, W., and Berman, F. A Model for Moldable Supercomputer Jobs. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium* (Los Alamitos, CA, USA, 2001), IEEE Computer Society.

[25] Citrix Systems. The three levels of high availability âĂŞ Balancing priorities and cost. Tech. rep., Citrix Systems Inc, 2009.

[26] Clark, C., Fraser, K., Hand, S., Hansen, J. G. J., Jul, E., Limpach, C., Pratt, I., and Warfield, A. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation* (2005), USENIX Association, pp. 273–286.

[27] Danak, A., and Mannor, S. Resource allocation with supply adjustment in distributed computing systems. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on* (2010), IEEE, pp. 498–506.

[28] de Assuncao, M. D., di Costanzo, A., and Buyya, R. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Proceedings of the 18th ACM international symposium on High performance distributed computing - HPDC '09* (New York, New York, USA, June 2009), ACM Press, p. 141.

[29] Di, S., Kondo, D., and Cirne, W. Characterization and comparison of cloud versus grid workloads. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on* (2012), IEEE, pp. 230–238.

[30] Diego, S., Sapuntzakis, C., Brumley, D., Chandra, R., Zeldovich, N., Chow, J., Lam, M. S., and Rosenblum, M. Virtual Appliances for Deploying and Maintaining Software. In *Proceedings of the 17th USENIX conference on System administration (LISA 03)* (Oct. 2003), USENIX, pp. 181–194.

[31] Distributed Management Task Force. Open Virtualization Format Specification. Tech. rep., Distributed Management Task Force, 2009.

[32] Dogan, A., and Ozguner, F. Scheduling independent tasks with QoS requirements in grid computing with time-varying resource prices. In *Grid Computing, 2002 3rd IEEE/ACM International Conference on Grid Computing* (Baltimore, 2002), M. Parashar, Ed., Springer, pp. 58–69.

[33] Downey, A. A model for speedup of parallel programs. Tech. Report UCB/CSD-97-933, U.C. Berkeley, 1997.

[34] Feitelson, D. Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload.

[35] Fielding, R. T. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[36] Forrest, W. Clearing the Air on Cloud Computing. Tech. Rep. March, McKinsey & Company, 2009.

[37] Foster, I. The grid: computing without bounds. *Scientific American 288*, 4 (2003), 78–85.

[38] Foster, I. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology 21*, 4 (July 2006), 513–520.

[39] Foster, I. The Open Grid Services Architecture, Version 1.5. Tech. rep., Global Grid Forum, 2006.

[40] Foster, I., and Tuecke, S. Describing the elephant: The Different Faces of IT as Service. *Queue 3*, 6 (July 2005), 26–34.

[41] Gagliardi, F., Jones, B., Grey, F., Bégin, M.-E., and Heikkurinen, M. Building an infrastructure for scientific Grid computing: status and goals of the EGEE project. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences 363*, 1833 (Aug. 2005), 1729–42.

[42] Garg, S. K., Buyya, R., and Siegel, H. J. Time and cost trade-off management for scheduling parallel applications on Utility Grids. *Future Generation Computer Systems 26*, 8 (Oct. 2010), 1344–1355.

[43] Gmach, D., Rolia, J., Cherkasova, L., and Kemper, A. Capacity management and demand prediction for next generation data centers. In *Web Services, 2007. ICWS 2007. IEEE International Conference on* (2007), HP Laboratories, IEEE, pp. 43–50.

[44] Goldberg, R. R. Survey of virtual machine research. *IEEE Computer 7*, 6 (1974), 34–45.

[45] Hayes, B. Cloud computing. *Communications of the ACM 51*, 7 (July 2008), 9.

[46] Hill, Z., and Humphrey, M. A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: The death of the local cluster? In *Proceedings of the 10th IEEE/ACM International Conference on Grid Computing* (oct. 2009).

[47] Huebscher, M. C., and McCann, J. A. A survey of autonomic computing-degrees, models, and applications. *ACM Computing Surveys 40*, 3 (2008), 1–28.

[48] IBM. An architectural blueprint for autonomic computing. Tech. Rep. June, International Business Machines, 2006.

[49] Isard, M., Budiu, M., and Yu, Y. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS European Conference on Computer Systems 2007* (2007).

[50] Jaeger, P., Lin, J., Grimes, J., and Simmons, S. Where is the cloud? Geography, economics, environment, and jurisdiction in cloud computing (2009). *First Monday 14* (Aug. 2010), 4–5.

[51] Javadi, B., Thulasiram, R. K., and Buyya, R. Characterizing spot price dynamics in public cloud environments. *Future Generation Computer Systems 29*, 4 (2013), 988–999.

[52] Keahey, K., Figueiredo, R., Fortes, J., Freeman, T., and Tsugawa, M. Science clouds: Early experiences in cloud computing for scientific applications. In *Cloud computing and applications* (2008), vol. 2008, pp. 825–830.

[53] Keahey, K., Foster, I., Freeman, T., and Zhang, X. Virtual workspaces: Achieving quality of service and quality of life in the Grid. *Scientific Programming 13*, 4 (Oct. 2005), 265–275.

[54] Keahey, K., and Freeman, T. Contextualization: Providing One-Click Virtual Clusters. In *2008 IEEE Fourth International Conference on eScience* (Dec. 2008), IEEE, pp. 301–308.

[55] Kintala, C. Checkpointing and its applications. *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers* (1995), 22–31.

[56] Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium* (2007), vol. 1, pp. 225–230.

[57] Kozuch, M., and Satyanarayanan, M. Internet suspend/resume. *Proceedings Fourth IEEE Workshop on Mobile Computing Systems and Applications*, Vm (2002), 40–46.

[58] Krauter, K., Buyya, R., and Maheswaran, M. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience 32*, 2 (Feb. 2002), 135–164.

[59] Kreger, H. Fulfilling the Web services promise. *Communications of the ACM 46*, 6 (June 2003), 29.

[60] Lee, M., Krishnakumar, A. S., Krishnan, P., Singh, N., and Yajnik, S. Hypervisor-assisted application checkpointing in virtualized environments. In *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)* (June 2011), IEEE, pp. 371–382.

[61] Li, H., Muskulus, M., and Wolters, L. Modeling job arrivals in a data-intensive grid. In *Job Scheduling Strategies for Parallel Processing* (2007), Springer, pp. 210–231.

[62] Mattess, M., Vecchiola, C., and Buyya, R. Managing Peak Loads by Leasing Cloud Infrastructure Services from a Spot Market. In *Proceedings of 12th IEEE International Conference on High Performance Computing and Communications (HPCC)* (Sept. 2010), IEEE, pp. 180–188.

[63] Matthews, J., Garfinkel, T., Hoff, C., and Wheeler, J. Virtual machine contracts for datacenter and cloud computing environments. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds* (New York, New York, USA, June 2009), ACM, pp. 25–30.

[64] Mazzucco, M., and Dumas, M. Achieving Performance and Availability Guarantees with Spot Instances. In *13th International Conference on High Performance Computing and Communications HPCC2011* (2011), no. ii.

[65] Mell, P., and Grance, T. The NIST Definition of Cloud Computing. Tech. Rep. 6, National Institute of Standards and Technology, 2011.

[66] Meyer, D. h. t., Aggarwal, G., Cully, B., Lefebvre, G., Feeley, M. h. j., Hut hinson, N. C., and Warfield, A. Parallax: virtual disks for virtual machines. *ACM SIGOPS Operating Systems Review 42*, 4 (Apr. 2008), 41.

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

[68] Milojicic, D., Douglis, F., Paindaveine, Y., Wheeler, R., and Zhou, S. Process migration. *ACM Computing Surveys 32*, 3 (Sept. 2000), 241–299.

[69] Milojicic, D., Llorente, I. M., and Montero, R. S. Opennebula: A cloud management tool. *IEEE Internet Computing 15*, 2 (2011), 11–14.

[70] Nagarajan, A. B., Mueller, F., Engelmann, C., and Scott, S. L. Proactive fault tolerance for HPC with Xen virtualization. In *Proceedings of the 21st annual international conference on Supercomputing - ICS '07* (New York, New York, USA, June 2007), ACM Press, p. 23.

[71] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on* (2009), IEEE, pp. 124–131.

[72] OpenStack Project. OpenStack Open Source Cloud Computing Software.

[73] Overview, S., Iaas, E. E., and Platform Computing. Platform ISF Datasheet, 2009.

[74] Pandey, S., Jin, C., Voorsluys, W., Rahman, M., and Buyya, R. Gridbus Worklow Management System on Clouds and Global Grids. In *Proceedings of the 4th IEEE International Conference on eScience* (2008), IEEE, pp. 323–324.

[75] Pandey, S., Voorsluys, W., Niu, S., Khandoker, A., and Buyya, R. An autonomic cloud environment for hosting ECG data analysis services. *Future Generation Computer Systems 28*, 1 (May 2011), 147–154.

[76] Pandey, S., Voorsluys, W., Rahman, M., Buyya, R., Dobson, J., and Chiu, K. A Grid Workflow Environment for Brain Imaging Analysis on Distributed Systems. *Concurrency and Computation: Practice and Experience 21*, 16 (2009).

[77] Papazoglou, M. P., and Heuvel, W.-J. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal 16*, 3 (2007), 389–415.

[78] Perlman, R. *Interconnections: bridges, routers, switches, and internetworking protocols*. Addison-Wesley Professional, 2000.

[79] Raman, R., Livny, M., and Solomon, M. Matchmaking: distributed resource management for high throughput computing. In *Proceedings. The Seventh International Symposium on High Performance Distributed Computing* (1998), IEEE Comput. Soc, pp. 140–146.

[80] Rappa, M. A. The utility business model and the future of computing services. *IBM Systems Journal 43*, 1 (2004), 32–42.

[81] Red Hat's Emerging Technology. oVirt.

[82] Singh, A., Korupolu, M., and Mohapatra, D. Server-storage virtualization: Integration and load balancing in data centers. In *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov. 2008), no. November, IEEE, pp. 1–12.

[83] Sobel, W., and Subramanyam, S. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *Proceedings of 1st Workflow on Cloud Computing and Applications* (2008).

[84] Song, Y., Zafer, M., and Lee, K.-W. Optimal bidding in spot instance market. In *INFOCOM, 2012 Proceedings IEEE* (March 2012), pp. 190–198.

[85] Sotomayor, B., Keahey, K., and Foster, I. Combining batch execution and leasing using virtual machines. In *Proceedings of the 17th international symposium on High performance distributed computing* (2008), ACM, pp. 87–96.

[86] Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. Resource Leasing and the Art of Suspending Virtual Machines. *2009 11th IEEE International Conference on High Performance Computing and Communications* (2009), 59–68.

[87] Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing 13*, 5 (Sept. 2009), 14–22.

[88] Subramanyam, S., Smith, R., van den Bogaard, P., and Zhang, A. Deploying web 2.0 applications on sun servers and the opensolaris operating system. Tech. rep., Sun Microsystems, 2009.

[89] Sun Microsystems. Project Faban. http://faban.sunsource.net.

[90] Tanenbaum, A. S. *Computer networks*. Prentice Hall PTR, 2003.

[91] Travostino, F., Daspit, P., Gommans, L., Jog, C., de Laat, C., Mambretti, J., Monga, I., van Oudenaarde, B., Raghunath, S., and Wang, P. Y. Seamless live migration of virtual machines over the man/wan. *Future Generation Computer Systems 22*, 8 (2006), 901–907.

[92] Tsafrir, D. Modeling user runtime estimates. In *Proceedings of the 11th Job Scheduling Strategies for Parallel Processing* (2005), Springer Berlin Heidelberg, pp. 1–35.

[93] Tsafrir, D. Using Inaccurate Estimates Accurately. In *Job Scheduling Strategies for Parallel Processing* (2010), Springer, pp. 208–221.

[94] Uhlig, R., Neiger, G., Rodgers, D., Santoni, A., Martins, F., Anderson, A., Bennett, S., Kagi, A., Leung, F., and Smith, L. Intel virtualization technology. *Computer 38*, 5 (2005), 48–56.

[95] Varia, J. Best Practices in Architecting Cloud Applications in the AWS Cloud. In *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. M. Goscinski, Eds. Wiley Press, 2011, pp. 459–490.

[96] Venugopal, S., Broberg, J., Buyya, R., and Engineering, S. OpenPEX: An open provisioning and execution system for virtual machines. In *17th International Conference on Advanced Computing and Communications (ADCOMâĂŹ09)* (2009), Technical Report, CLOUDS-TR-2009-8, CLOUDS Laboratory, The University of Melbourne, Australia, Citeseer.

[97] Verma, A., Ahuja, P., and Neogi, A. pMapper: power and migration cost aware application placement in virtualized systems. In *Proceedings of*

the 9th ACM/IFIP/USENIX International Conference on Middleware (2008), Springer-Verlag New York, Inc., pp. 243–264.

[98] VMWare Inc. Vmware.

[99] VMWare Inc. VMWare Virtual Appliance Marketplace.

[100] VMWare Inc. VMware High Availability. Tech. rep., 2009.

[101] VMWare Inc. VMware vSphere, the First Cloud Operating System. Tech. rep., 2009.

[102] VMWare Inc. VMWare vStorage APIs for Data Protection, 2009.

[103] Voorsluys, W., Broberg, J., and Buyya, R. Introduction to Cloud Computing. In *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. Goscinski, Eds. Wiley, 2011, ch. 1, pp. 3–42.

[104] Voorsluys, W., Broberg, J., Venugopal, S., and Buyya, R. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *Proceedings of the 1st IEEE International Conference on Cloud Computing (CloudCom 2009)* (Berlin, Heidelberg, Sept. 2009), vol. 5931 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 1–12.

[105] Voorsluys, W., and Buyya, R. Reliable Provisioning of Spot Instances for Compute-intensive Applications. In *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications* (Fukuoka, Japan, Mar. 2012), IEEE, pp. 542–549.

[106] Voorsluys, W., Garg, S. K., and Buyya, R. Provisioning Spot Market Cloud Resources to Create Cost-effective Virtual Clusters. In *Proceedings of the 11th IEEE International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-11)* (Melbourne, Australia, 2011), Y. Xiang, A. Cuzzocrea, M. Hobbs, and W. Zhou, Eds., vol. 7016 of *Lecture Notes in Computer Science*, Springer, pp. 395–408.

[107] Yi, S., Kondo, D., and Andrzejak, A. Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud. In *Proceedings of the 3rd IEEE International Conference on Cloud Computing* (July 2010), no. January, IEEE, pp. 236–243.

[108] Youseff, L., Butrico, M., and Da Silva, D. Toward a Unified Ontology of Cloud Computing. In *2008 Grid Computing Environments Workshop* (Nov. 2008), IEEE, pp. 1–10.

[109] Zhang, Q., Zhu, Q., and Boutaba, R. Dynamic Resource Allocation for Spot Markets in Cloud Computing Environments. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing* (Dec. 2011), IEEE, pp. 178–185.

[110] Zhao, M., and Figueiredo, R. J. Experimental study of virtual machine migration in support of reservation of cluster resources. In *VTDC '07: Proceedings of the 3rd International Workshop on Virtualization Technology in Distributed Computing* (New York, NY, USA, 2007), ACM, pp. 1–8.