# PERFORMANCE, RESOURCE AND COST AWARE VIRTUAL MACHINE ADAPTATION

Lajanugen Logeswaran

148055X

Thesis submitted in partial fulfillment of the requirements for the degree Master of Science in Computer Science and Engineering

Department of Computer Science & Engineering

University of Moratuwa

Sri Lanka

August 2015

# DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

**Candidate**

..... .....

Lajanugen Logeswaran Date

The above candidate has carried out research for the Masters thesis under my supervision.

**Supervisors**

..... .....

Dr. H. M. N. Dilum Bandara Date

..... .....

Dr. A. S. Perera Date

# ABSTRACT

**Performance, Resource and Cost aware Virtual Machine Adaptation**

Cloud Computing has increasingly become an attractive paradigm for computing during recent years. In the current Infrastructure as a Service (IaaS) cloud landscape users pay for statically configured Virtual Machine sizes irrespective of usage. Although the auto-scaling features offered by current cloud providers enable cloud hosted applications to dynamically scale the amount of resources allocated, the adopted configurations are often sub-optimal owing to the lack of flexibility involved in resoure provisioning. This results in higher costs and difficulty in meeting performance targets for clients.

It would be more favorable for users to consume (and be billed for) just the right amount of resources necessary to satisfy the performance requirement of their applications. Although prior work have suggested a variety of approaches to the auto-scaling problem, the benefits of these approaches remain restricted to applications that mainly depend on CPU and memory. The reason is partly due to cloud operators not providing guarantees on resource types that are difficult to partition such as IO and networking performance in their typical VM offerings (although specialized instances for these types of resources are available).

We take a novel perspective in addressing this problem where we assume that the cloud operator exposes a small, dynamic fraction (for security and privacy reasons) of its infrastructure and the corresponding resource specifications and constraints to each application. Assuming such a scenario we propose a dynamic VM reconfiguration scheme which comprises an Application Performance Model, a Cost Model and a Reconfiguration algorithm. The performance model helps estimate the performance of an application given specific resources. The Cost model assigns a numerical cost value to resource candidates made available to the application considering the lease expense, reconfiguration penalty and operating income. A reconfiguration algorithm assisted by the cost model makes optimal reconfiguration decisions. Simulation results for the RUBiS and filebench-fileserver applications and the worldcup workload show significant cost savings can be achieved while meeting performance targets compared to rule-based scaling systems.

Our proposed framework has the advantages of being simple, generic and computationally efficient. This framework is also attractive from a cloud operator's perspective as it indirectly assists the operator with the problem of efficient datacenter utilization.

**Keywords**: Auto-scaling; Cloud Computing; Cost Model; IaaS Cloud;

# ACKNOWLEDGEMENTS

First and foremost, I express my sincere gratitude to my advisor Dr. Dilum Bandara. His continuous support, patience, guidance and advice made the successful completion of this research possible. I am thankful to him for regular meetings and productive discussions despite his busy schedule. I highly appreciate his tolerance during times of my slow progress due to health problems or other issues. It has been a wonderful experience working with him, during the course of which I have acquired and improved new knowledge and skills that will be useful to my career in future.

I would like to thank members of my review committee Dr. Srinath Perera and Dr. Chinthana Wimalasuriya for their helpful feedback during my progress reviews.

My sincere thanks goes to the Computer Science Department of the University of Moratuwa for facilitating me with the necessary resources throughout the course of my research. I am thankful to Dr. Dilum Bandara and Prof. Sanath Jayasena for providing me with office space necessary to carry out my research without any hindrance. Thanks are in order to the System Engineers of the Department, especially Mr. Sujith Fernando, for helping me acquire, setup and manage hardware resources. I also thank the department staff in general for their friendly interaction, making my time at the department a fruitful and pleasant one.

I thank the Senate Research Committee of the University of Moratuwa for supporting this research under Senate Research Grant award number SRC/LT/2014/01. I also thank the LK Domain Registry for supporting this research through the Prof. V. K. Samaranayake top-up grant.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| Amazon EC2 | Amazon Elastic Compute Cloud |
| CDF | Cumulative Distribution Function |
| IaaS | Infrastructure as a Service |
| IOPS | IO operations per second |
| KCCA | Kernel Canonical Correlation Analysis |
| MI | Millions of Instructions |
| MIPS | Millions of Instructions per second |
| PaaS | Platform as a Service |
| PDF | Probability Distribution Function |
| PM | Physical Machine |
| RL | Reinforcment Learning |
| RUBiS | Rice University Bidding System |
| SaaS | Software as a Service |
| SLA | Service Level Agreement |
| SVC | Support Vector Clustering |
| SVR | Support Vector Regression |
| VM | Virtual Machine |
| VMWare DRS | VMware Distributed Resource Scheduler |

# Chapter 1

# INTRODUCTION

## 1.1 Cloud Computing

Cloud computing has become an attractive paradigm for computing in the recent years. Cloud computing refers to the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services [1]. This model of computing offers numerous advantages, some of the most attractive ones being no upfront investment, elasticity and the pay-as-you-go model. It saves us the hassle of purchasing and maintaining hardware and software for personal/enterprise use, and instead provides compute capability as a utility.

There are three service models associated with cloud computing - Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). *IaaS* provides computing infrastructure (physical or virtual machines) and other resources such as storage and networking for lease. Major providers of IaaS include Amazon EC2, Windows Azure, Rackspace and Google Compute Engine. *PaaS* provides computing platforms which typically include an operating system, programming environments and tools to build and deploy applications, database, web server, etc. Major players in this market segment include Windows Azure, Heroku, Google App Engine, Apache Stratos, etc. *SaaS* provides access to application softwares often referred to as on-demand softwares. In this case the client does not need to be concerned about installation, setup and running of the application, which are done by the service provider and the user simply uses the software and pays for it. Examples include Google Apps, Microsoft Office 365 and SalesForce.

We concern ourselves with the IaaS service model. As mentioned above, in the IaaS setting users of the cloud may request for compute and other resources,

Instance Types Matrix

| Instance Type | vCPU | Memory (GiB) | Storage (GB) | Networking Performance | Physical Processor | Clock Speed (GHz) | Intel AVX† | Intel AVX2† | Intel Turbo | EBS OPT | Enhanced Networking† |
|---|---|---|---|---|---|---|---|---|---|---|---|
| t2.micro | 1 | 1 | EBS Only | Low to Moderate | Intel Xeon family | Up to 3.3 | Yes | - | Yes | - | - |
| t2.small | 1 | 2 | EBS Only | Low to Moderate | Intel Xeon family | Up to 3.3 | Yes | - | Yes | - | - |
| t2.medium | 2 | 4 | EBS Only | Low to Moderate | Intel Xeon family | Up to 3.3 | Yes | - | Yes | - | - |
| t2.large | 2 | 8 | EBS Only | Low to Moderate | Intel Xeon family | Up to 3.0 | Yes | - | Yes | - | - |
| m4.large | 2 | 8 | EBS Only | Moderate | Intel Xeon E5-2676 v3 | 2.4 | Yes | Yes | Yes | Yes | Yes |

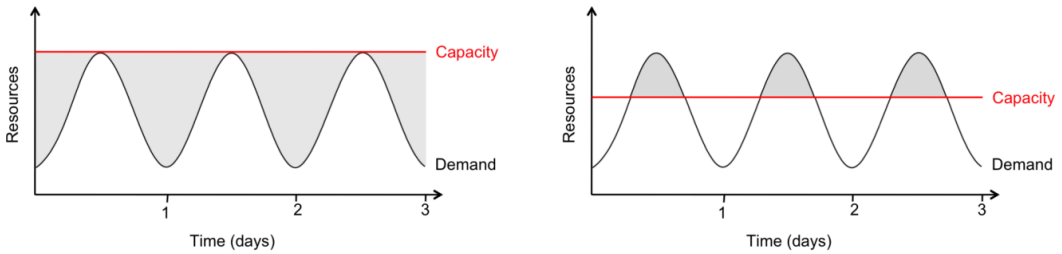
Figure 1.1: A few instance types offered by Amazon EC2.

for which they are billed based on usage. Figure 1.1 shows a few Virtual machine (VM) instance types offered by Amazon EC2 [2].

A cloud computing infrastructure is a complex system with a large number of shared resources and users. The cloud provider is faced with the challenge of accomodating and managing the large number of its user VMs effectively and efficiently on its hardware. The user is faced with such problems as how to determine the performance needs of his application, choose a particular VM size, deal with applications sensitive to networking and IO performance, etc. We discuss these challenges associated with resource provisioning in IaaS clouds next.

## 1.2 Resource Provisioning in the Cloud

In the current setting, users pay for statically configured VM sizes irrespective of the actual resources consumed by the hosted application. This makes it necessary for clients to choose a particular VM resource configuration in advance, which also requires a good understanding of the kind of workload to expect. As Figure 1.2 shows, underprovisioning leads to performance requirements not being met, and overprovisioning leads to higher operating costs to the user. It is desirable for user VMs to adapt based on actual performance needs. This benefits users because




(a) Provisioning for peak load (b) Underprovisioning

Figure 1.2: Overprovisioning and underprovisioning.

of the ability to meet performance needs at lower costs, as well as the provider because of increased customer willingness to use the cloud.

Although cloud providers do provide scaling mechanisms to cope with workload variations, these schemes are based on rules such as adding/removing VMs based on resource utilization [3]. They provide limited flexibility to the application in adapting different resource configurations over time and often produce suboptimal configurations.

Prior work have suggeted a variety of approaches to the auto-scaling problem beyond rule-based systems [4]. However, the benefits of these approaches remain restricted to applications that mainly depend on CPU and memory. The reason is partly due to cloud operators not providing guarantees on resource types that are difficult to partition such as IO and networking performance in their typical VM offerings.

Another main drawback of statically configured VMs is that applications co-located on the same hardware can have unpredictable impacts on each other [5]. This is typically a problem for types of resources which are hard to partition. As a result cloud providers are unable to provide guarantees for IO and network resource availability to offered VM's. Although some providers do provide specialized network/IO optimized instances with guaranteed discrete levels of service such as low, moderate and high [2]. The availability of these resources can vary over time depending on the applications running on the cloud. This situation would be better if it were possible for users to request and for providers to provide specific quantities of many types of resources.

## 1.3 Problem Statement

Consider the scenario of a client application deployed in an IaaS (Infrastructure as a Service) cloud. The client is interested in spending less for leasing resources while making sure that the application performance requirements are met. This requires the ability to accurately estimate resource requirements of an application, and a framework to provision the required amount of resources dynamically over time based on the workload.

The cloud being a highly dynamic environment with co-located applications exerting unpredictable impact on each other, a significant challenge involved is the ability to provision required amount of resources along resource dimensions that are hard to partition such as IO and networking performance. Furthermore, any such scheme has to be mutually agreeable between the client as well as the provider without conflicts of interest.

In summary, we attempt to *enable an application deployed in an IaaS cloud to consume just the right amount of resources to fulfill its performance needs while minimizing the cost of resources leased.*

## 1.4 Contributions

We propose a novel dynamic resource reconfiguration framework for cloud hosted applications. The proposed scheme enables applications to meet performance targets and maximize profits while minimizing the cost of resources leased from the cloud provider.

Prior approaches have looked at the resource scaling problem from the perspectives of the user as well as the provider. User space solutions exist such as scaling schemes driven by application performance models. Rule-based scaling is a provider perspective solution where the provider determines resource requirements and allocates VMs appropriately. We take a hybrid approach in which the user and cloud operator both provide some information regarding the

application and the cloud infrastructure respectively. This leads to a resource provisioning scheme that is beneficial to both the user and the provider.

Our specific contributions are as follows:

- We break away from prior approaches which limit themselves to the resource management capabilities of current cloud providers, and consider a scenario where the cloud provider makes available a relatively small pool of machines as choices (VMs/PMs) to each client application hosted by the provider. The main advantages of this scenario are:
  - It provides applications the flexbility to adopt favorable configurations along resource dimensions that are difficult to partition (such as IO and networking performance). This makes it possible for applications to make more accurate scaling decisions and offers room for cost savings.
  - It naturally exploits the co-hosted application interference problem in shared resources to the benefit of the client as well as the cloud vendor. Instead of determining and controlling performance interference due to co-located applications, the applications themselves figure out which of the vacant resource options are suitable to meet performance needs.
- Within this proposed framework, we propose a novel dynamic reconfiguration scheme for cloud hosted applications in an attempt to fill the gaps in prior work discussed in Section 1.1, which is comprised of the following components:
  - *Application Performance Model*

    Predicts the performance of an application given the workload and resources allocated to application.
  - *Cost Model*

    Assigns a numerical value to resource configurations considering performance predictions, monetary costs and reconfiguration penalties.
  - *Reconfiguration Algorithm*

    Makes use of the cost model to make favorable scaling decisions.

- The proposed framework is generic and offers the flexibility to plug in specific choices of models for each of the components. We propose specific choices of models for these components, including a novel analytical cost model.

- We build a discrete event simulator and use it to validate our approach. Simulation results of different scenarios indicate that the proposed approach offers significant cost advantages while meeting application performance requirements compared to scaling schemes currently used by cloud providers.

## 1.5 Organization of the Thesis

The rest of the thesis is organized as follows. Chapter 2 reviews prior related work. In Chapter 3 we describe the proposed approach. We evaluate our approach in Chapter 4 - Section 4.1 describes the experimental setup, Section 4.2 describes the simulation setup and performance results are discussed in Section 4.3. We conclude and discuss future work in Chapter 5.

# Chapter 2

# LITERATURE SURVEY

We set out with the goal of devising a resource provisioning and management scheme that is desirable from both the client as well as cloud providers perspective. We reviewed several related problems addressed by previous work, as well as their approaches to the problem considered. Several ideas from these prior work formed the basis of our work. We discuss these ideas under the following major class of problems in an attempt to place our work in the context of previous studies. In Section 2.1 we discuss literature on the *Auto Scaling* problem. Sections 2.2 and 2.3 discuss the *VM Placement* and *Application Placement* problems. In Section 2.4 we discuss prior work on *Application Performance Modeling*. Section 2.5 describes cost models proposed in the literature. Finally, in Section 2.6 we discuss relevant simulation tools.

## 2.1 Auto Scaling

The *auto-scaling problem* has been widely studied. Auto-scaling refers to adding/removing resources to the operating pool of resources of an application dynamically depending on the workload. These scaling actions can be of two types; *horizontal scaling*: adding or removing a number of VM instances, and *vertical scaling*: adding more resources (CPUs/memory) to existing VMs on the fly, while they are running.

Approaches to auto-scaling dominantly fall under the following techniques: Rule-based systems, Reinforcement Learning, Queueing Theory and Control Theory. We give a brief overview of these techniques and discuss their pros and cons in the following sections. A comprehensive review of these techniques and related bibliography can be found in [4].

### 2.1.1 Rule Based Systems

Rule based systems are the most intuitive approach to auto-scaling [6, 7, 8]. Scaling decisions are made based on a set of rules for scaling up and down. Rules dictate adding or removing VMs based on a performance metric such as CPU load, request rate or average response time. Scaling actions are controlled by upper and lower thresholds, which correspond to scaling up and down respectively. To prevent frequent reconfigurations and system instability inertia durations are introduced during which scaling actions are forbidden. An example rule could be something like *If CPU utilization exceeds 90% throughout a duration of 100s add a new VM*. Further parameters have also been introduced for more flexibile rule-based systems.

An apparent difficulty associated with these systems is choosing a good set of parameters for a target application. However, usually adopted best practices are useful [9].

Despite all the sophisticated techniques for auto-scaling proposed in the past (which we discuss briefly below), cloud providers dominantly use rule-based autoscaling systems. The main reason being the simplicity and intuitive nature of rule-based systems and the skepticism about these methods due to the unrealistic models they employ and their lack of robustness [10].

### 2.1.2 Reinforcement Learning

Approaches to the auto-scaling problem based on reinforcement learning have been quite common [11, 12, 13]. A Reinforcement Learning (RL) formulation contains an intelligent agent that automatically learns from an environment. The main elements of the RL framework are states, actions and rewards. The agent interacts with the environment by applying an action and learning from the reward (positive or negative) awarded by the environment. The agent chooses actions based on a *policy*. The objective of the agent is to learn the optimal policy to achieve maximum reward in the long run.

Reinforcement Learning applied to the auto-scaling problem takes the following form: The states represent resource configurations (a set of VMs with specific resource parameters). Actions correspond to scaling decisions. Rewards depend on metrics such as whether performance requirements were met or amount of income obtained due to the scaling decision.

The main challenges associated with designing an RL agent for the auto-scaling problem is the large training time and huge state space. The system is initially running sub-optimally, and it may take a long time for the agent to learn good (state,action) pairs. Recent techniques have proposed techniques to overcome such problems through better initial functions and usage of non-linear function approximators instead of lookup tables for Q-learning [14, 15].

### 2.1.3 Control Theory

Control systems are a natural approach to the auto-scaling problem [16, 17]. Adaptive feedback and feedforward controllers have been widely suggested by prior work. The objective of the control system is to maintain the output (performance) of the target system (the application) to the desired level by adjusting the control input (resource configuration).

### 2.1.4 Queuing Theory

In techniques based on queuing theory [18, 19], applications are modeled using queuing models. The application hosting VMs are considered as VMs and the workload as a queue of requests. Assumptions are made regarding the properties of the workload such as requests come from a Poisson process which makes computing properties of the queuing model tractable. Properties of the system such as response time and throughput are calculated based on analytical treatments available in queuing theory. Queuing networks, which are a combination of several queues, are also used to model multi-tier applications, each tier typically being represented as a separate queue.

A drawback of analytic methods such as queuing models are assumptions made that are usually not realistic. Also, computations become less tractable as the system grows in complexity.

## 2.2 VM Placement

Another related problem is the *VM placement problem*, which requires determining on which hosts a given set of VMs need to be placed/instantiated with objectives such as maximizing VM consolidation ratios considering impact of co-located VMs and datacenter energy optmization. A bin packing formulation is commonly employed. The complexity of solving the bin packing problem increases exponentially with problem size. Proposed optimization objectives are NP-hard in general and heuristic based algorithms are usually proposed [20, 21, 22, 23, 24].

In [22] a dynamic server migration and consolidation algorithm is proposed for Service Level Agreement (SLA) satisfaction. The optimization objective considered is minimizing the time averaged number of active physical servers hosting virtual machines respecting the constraint that demands exceeds capacity in no more than a percentage $p$ of measurement intervals indicated by a threshold $p$ (SLA requirement).

The algorithm has three stages:

- Measure (past historical demand data).
- Forecast future demand.
- Remap: mapping of VMs to PMs minimizing the number of PMs required to support a workload at a specified rate of SLA violations and reduce rate of SLA violations for a fixed capacity.

VM placement is formulated as a bin packing problem and a heuristic based solution is proposed. The focus is only on one type of resource in this work.

A Minimum Cost Maximum Flow (MCMF) Algorithm for VM placement is proposed in [21]. Optimal VM placement for profit maximization is modeled as a maximum flow problem and a MCMF based solution is presnted. The proposed algorithm is claimed to be of low complexity.

The idea of exploiting demand correlation between VMs, also called VM multiplexing, for the placement problem is suggested in [25]. This work proposes joint VM provisioning - instead of allocating resources on a per VM basis, allocating resources to a group of VMs exploiting statistical multiplexing among their workload patterns. Demand patterns differ across different VMs. While demand for one peaks, the demand for another may dip, in which case it placing both VMs on the same PM will not result in contention for resources. The main objective is making use of statistical multiplexing to consolidate VMs, thereby achieving CPU capacity savings. The authors propose algorithms for identifying group of VMs with most compatible demand patterns and allocating resources to the group as a whole making sure that the SLAs of individual VMs are met.

Along similar lines [23] argue that previous work have ignored the dynamic nature of the incoming stream of VM deployment requests. They suggest that the demand correlation between VMs in the past can be used for future prediction. The placement objective is formulated as a multi-dimensional bin packing problem and a heuristic based algorithm is proposed. The constraint considered is an upper bound on a weighted sum of the following:

- Amount of CPU allocated to VMs with respect to desired amount (sum of allocated resources/sum of actual demands).
- Number of relocations (minimize number of VM migrations).
- Difference between most and least loaded hosts (load balancing).

VM deployment, undeployment and migration decisions are made based on the algorithm. Our proposed approach indirectly employs the idea of VM multiplexing whereby the application decides whether moving to a particular physical host will satisfy its performance requirements.

In [26] the authors (from VMware) discuss challenges invovled in cloud scale resource management. They discuss features of the VMware Distributed Resource Scheduler (DRS) and highlight that DRS does not scale well to the cloud environment and that it works well only in a small scale. They state that what we really need is a combinination of two strengths: EC2's elasticity and DRS's resource management. The main aspects/challenges to be considered in

the design of such a scheme are described as scale, resource heterogeneity, high frequency of operations and failure tolerance.

A few proposals for a solution are also pointed out by the authors such as hierarchical scaling, flat scaling and statistical scaling. Of these, we found the idea of statistical scaling particularly attractive. The key idea is to obtain large-scale resource management by doing small-scale optimizations. The authors propose doing this by creating small clusters dynamically (sampling random hosts) and to run DRS on individual clusters. Our proposed approach exploits a similar idea to the benefit of both the cloud provider as well as the client.

It is claimed in [27] that the two widely used VM placement products - xenserver and VMware DRS, are not network-aware. The authors show how VM placement could be better informed by network knowledge and propose algorithms that improve performance by exploiting network traffic and topology knowledge. This work demonstrates incremental benefits of adding network knowledge to today's CPU-only algorithms. A minimax objective is considered - Minimizing the utilization of the maximally utilized resource (either a PMs CPU or a network link). Two greedy algorithms and four variants of Simulated Annealing are proposed as solvers.

## 2.3 Application Placement and VM Provisioning

While the VM placement problem looks at how a set of VMs should be placed on PMs to satisfy a certain objective, the combined placement of applications and VMs has the following objective: given a set of machines and applications, determine how many machines to run for each application and on which physical machines to place them, respecting resource constraints. This is a datacenter level problem of concern to the cloud provider, as in the case of VM placement.

In [28] a *Scalable Application Placement Controller* for datacenters is proposed. Given a set of machines and web applications with dynamic demands, an online application placement controller decides how many instances to run for each app and where to put them. Multiple optimization objectives are considered:

- Maximize total satisfied application demand.
- Minimize total and of application starts and stops (overhead).
- Balance load across machines (utilization of individual machines must stay close to the utilization of the entire system).

A placement restriction matrix (whether a given application can run on a particular machine or not) as well as CPU and memory capacity constraints are the constraints considered. The problem is formulated as a variant of the Class Constrained Multiple-Knapsack problem and an onnline approximation algorithm is proposed as a solver.

A dynamic resource provisioning and VM placement scheme, considering application level SLAs and resource exploitation costs is proposed in [20]. The following breakdown into two sub-problems is considered:

- VM provisioning - Determines the number of each of the different types of VMs to allocate to each application with the objective of maximizing a weighted sum of utilities which relate to the application SLAs.
- VM packing - Pack the VMs suggested by the previous stage minimizing the number of active PMs.

These problems are modeled as separate CSPs and solved using Constraint Programming. This work considers heterogeneous applications and workloads (batch oriented and enterprise online applications).

The following problem is addressed in [24]. Consumers submit requests for allocation/deaallocation of VMs of different capacities with associated SLAs. The provider has to manage placement, instantiation and migration of the VMs on its PMs. A reveneue maximization objective is formulated considering the following - Availability SLA (Availability Model is built), maximum number of VM migrations and VMs deployed on external cloud (decreases revenue). A hill climbing heuristic search algorithm is proposed to solve this constrained optimization problem.

## 2.4 Application Performance Modeling

A performance model models the relationship between the performance of an application and the parameters that determine performance such as allocated resources and workload. Given a particular quantity of workload and a specific resource allocation, the model would predict what performance will be exhibited by the application. Performance is represented using an appropriate performance metric, some of the common ones being mean response time and throughput.

Performance models enable us to make accurate scaling decisions in a production system as they enable us find how performance would vary across different resource allocations for a particular level of observed workload. Performance modeling has been studied in both virtualized and non-virtualized environments. Approaches based on machine learning have been the most common. Simple regression based techniques were used in the past and recent work have shown that more sophisticated models are necessary to model the complex relationship between different types of resources and performance.

The work [29] focuses on dynamic workload performance prediction and resource allocation. The modelling technique proposed is as follows:

1. A basic set of applications which represent the diversity of applications (CPU intensive, memory intensive, etc.) in a datacenter are chosen.

2. Performance models are built for these primitive applications using Support Vector Regression (SVR).

3. During operation, for a new workload encountered estimates are derived for the probabilities that this workload is similar to each of the primitive set of apps using Support Vector Clustering (SVC).

4. The performance model of the new workload is derived as a weighted summation of the models built in (2), the weights being the probability estimates derived in (3).

5. An optimization problem which minimizes the total cost of resources with respect to allocated resources, under the constraint that the realized performance is equal to the expected performance is solved, and an approximate solution is obtained.

The first two tasks are done offline beforehand, and the rest take place periodically online. We found the modeling technique attractive. The choice of primitive applications did not seem to be very methodical however. The resources considered seems to be number of CPU threads and memory, but the optimization formulation is general.

The distribution of application response time given allocated resources is learned in [30]. Following is the model considered:

$$p(t|a) = \sum_{i,j,k} p(t|a,i,j,k).p(i,j,k|a) \tag{2.1}$$

where the symbols denote the following: $t$ - response time, $a$ - resource allocation, $i, j, k$ - utilization (consumed resources/allocated resources) values of the three application tiers. All these are discretized values, discretized using fixed-with binning. The paper focuses only on CPU allocation. Parameters of the two probability distributions on the right of equation 2.1 are learned using training data. Some of the apparent drawbacks of the approach are the following:

- Only CPU is considered.
- Model is too simplistic. More variables could be taken into account in the model.
- Discretization reduces model effectiveness. Only four values for the allocation variable a are considered (25%, 40%, 70%, 100%) in the experimentation.
- Contention for resources is not explicitly addressed in model.

A more powerful and widely applicable model may be built by considering multiple resources and their interdependencies, and by using discrete distributions with a large number of bins. We address these gaps in our performance model.

In [31] the authors address the problem of predicting performance metrics of database queries. They evaluate several statistical machine learning algorithms for this prediction problem and show that Kernal Canonical Correlation Analysis (KCCA) performs well on the task. KCCA is a technique used to correlate points in two sets of data, the datasets in this case being database query feature vectors and performance metrics of the corresponding queries. The key idea is to project both sets of points to spaces with same dimensionality such that the distances between corresponding point pairs in the two sets in the projection space are similar. Prediction involves given a point in one of the sets, projecting the point to its projection space, finding a corresponding point in the projection space of the other set, and then mapping it back to its original space. We initially considered KCCA as a candidate for our performance model, but later opted for other methods as it was unsuitable to the performance metric we considered (response time distributions).

In [32] the authors claim that it is difficult to capture the complex relationship between performance and resource allocation using a single global model. They demonstrate that fitting different models to different parts of the input space can produce accurate predictions. The authors consider CPU, memory, and storage I/O as resource types. The approach is validated using the RUBiS [33] and filebench [34] applications. They further use these models to recommend VM instances suitable from a cloud provider such as Amazon EC2 or Rackspace for a given workload.

Performance models are applicable directly to the VM instance recommendation problem as well, where the user wants to choose a static VM size to suit his application. The work discussed above [32, 29] also evaluate the strengths of their models in terms of how well the recommendations are.

## 2.5 Cost Model

Our approach uses an analytical model to compare different possible scaling actions. In this section we discuss relevant models proposed by prior work.

Table 2.1: Components of cost function in [35].

| Symbol | Description |
|---|---|
| $W_r$ | Penalty for SLA violation |
| $W_c$ | Cost of leasing a machine per hour |
| $W_f$ | Cost of reconfiguring application |
| $R_{SLA}$ | SLA given response time |
| $R$ | Maximum response time observed |
| $M_i$ | Number of machines used in the $i$th interval |

The cost model proposed in [35] is a linear combination of the following:

- Penalty for violation of SLA bounds (e.g., extent by which desired performance is violated, such as increase in response time)
- Cost of leasing a machine
- Cost of reconfiguring the application (defined as a change in number of machines used before and after reconfiguration. This is a penalty term which discourages horizontal scaling).

Equation 2.2 shows the model.

$$Cost = W_r \times (R_{SLA} - R) + W_c \times M_k + W_f \times ||M_k - M_{k-1}|| \tag{2.2}$$

where the symbols used are described in Table 4.2.

The reconfiguration term is defined as the change in number of machines used before and after reconfiguration, and is used to penalize horizontal scaling. We observed that the proposed reconfiguration cost in this model is too simple to be realistic. This simplistic model motivated our cost model, although we handle SLA fulfillment in a different manner and our reconfiguration cost also takes into account VM migration.

### 2.5.1 Cost of VM migration

Moving to a VM in a physical cost different than the one in which the application is running is one of the scaling actions we consider in our model. Moving a VM

from a source host to a destination cost is termed *VM migration*. Migrating a VM between hosts can be done in one of two ways:

- Offline migration: The running VM is stopped, its image is copied to the destination host, and resumed there.
- Online/live migration: Copying the VM image (part of it) while the VM is operational.

Offline migration completely disrupts the application since the machine is shutdown until resumption in the destination, while live migration does some of the copying while the VM is running. Several hypervisors used today such as xen [36] and vmware [37] support live migration.

Live migration operates in two phases. During the *Pre-copy phase* while the VM is running, its memory pages are copied over to the destination iteratively. The reason for copying the pages iteratively is because of the possibility for some of the already transferred pages to become invalid because of the corresponding pages getting modified in the source VM. An invalid page is termed a *dirty* page. After the pre-copy phase has progressed for some time (as determined by heuristics such as number of iterations/number of remaining dirty pages), the running VM is stopped and the remaining dirty pages are copied to the destination.

During the pre-copy phase, bandwidth is shared by the application as well as the migration procedure. During copy phase, the service is entirely down, and the complete bandwidth is used for migration. We discuss below prior work that address the performance penalty incurred due to live VM migration.

In [38] the authors discuss the impact of vertical scaling and VM migration on performance. Although they do not provide an analytical cost model, they provide a set of observations which can be useful in devising one. Their observations with regard to live migration are as follows:

- If there are no resource constraints, the duration of migration for an application varies linearly with the active memory of the VM. The migration duration varies across applications with same memory footprint.

- Live migration requires spare CPU resources on the source server. If spare CPU is not available, it impacts the duration of migration and the performance of the VM being migrated.
- The amount of CPU required for live migration increases with an increase in the number of active pages of the VM being migrated.
- A co-located VM impacts a VM being migrated by taking away resources from the physical server. The co-located VM does not suffer from CPU contention but may suffer from cache contention based on its cache usage pattern.

In [39] the authors conclude based on their experiments that although migration overhead is acceptable, it cannot be disregarded when strict SLAs need to be adhered.

A migration cost model based on available bandwidth, dirtying rate and VM memory size is proposed in [40]. This work develops analytical models for the SLA penalty incurred due to migration, and also presents a migration scheme that minimizes this SLA penalty cost by determining optimal values for the following parameters- Amount of bandwidth to use for copying during pre-copy, when to stop VM and initiate copy phase. They compare the performance of their proposed migration scheme with Xen's migration using the developed cost model. The following modeling assumptions are made - constant request rate, requests follow a queuing model with constant request and service distributions, probability of a pre-copied page getting dirty is modeled using a Bernoulli distribution. We use the analytical expression developed in this work for the probability that response time exceeds a certain threshold in our model.

### 2.5.2 Income Model

We consider a profit maximization objective in our approach in addition to performace satisfaction. In this section we briefly review some relevant models proposed in the literature.

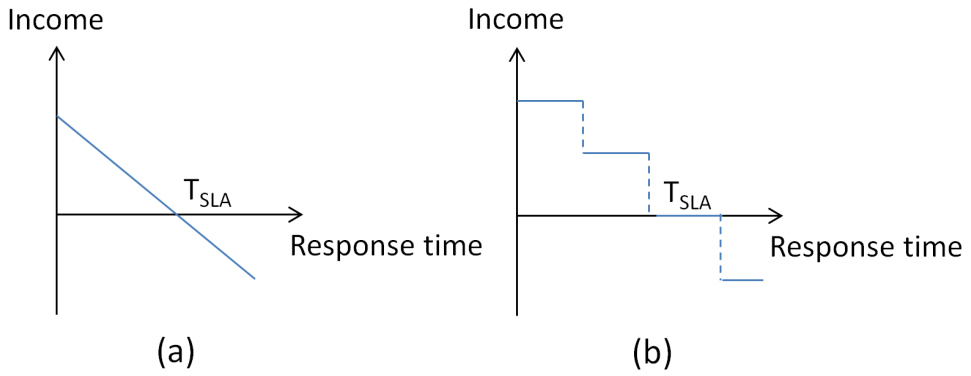



Figure 2.1: Utility models.

Assuming the target performance to be keeping the response time under a specific response time delay $T_{SLA}$ with high probability, in [41, 42] revenue schemes with the following properties are proposed:

- Smaller the response time, higher the revenue.
- Strongly discourage response time falling below Tsla by setting utility to be negative.

In [41] the following model is considered

$$P(1 - \frac{r}{T_{SLA}}) \tag{2.3}$$

where $P$ is a constant and $r$ is the mean response time observed (see Figure 2.1(b)). In [42] a piecewise constant function is used to define utility (see Figure 2.1(a)), a multiple of which is defined to be revenue. The purpose of these revenue models is more towards discouraging large response time than reflecting actual revenue. We use the model in equation 2.3 for our experiments.

## 2.6 Simulation Tools

Several simulation tools have been developed in the past for validating cloud resource management algorithms. We discuss a few such simulators below, highlighting some of their main features, primarily the models of task execution.

### 2.6.1 CloudSim [43]

Cloudsim assumes provisioned VMs to be predictable and stable in performance. The basic unit of workload in Cloudsim is called a *cloudlet*. Workload is defined by a sequence of cloudlets. The time required to execute a given cloudlet on a VM solely depends on the cloudlet's length (in MI (Millions of Instructions)) and VM's processing power (in MIPS (Millions of Instructions per second)). A cloudlet is defined by a set of The execution time of cloudlets (a task) is calculated as follows:

$$\text{Execution Time} = \frac{rl}{capacity \times cores} \tag{2.4}$$

where,

- $rl$ is total number of instructions the cloudlet needs to execute on a processor
- $cores$ is the number of cores (processing elements) required by the cloudlet
- $capacity$ of a host with np processing elements is given by

If cloudlets are scheduled in space-shared policy,

$$capacity = \sum_{i}^{np} \frac{cap(i)}{np} \tag{2.5}$$

If cloudlets are scheduled in time-shared policy,

$$capacity = \frac{\sum_{i}^{np} \frac{cap(i)}{np}}{max(\sum_{i}^{cloudlets} cores(i), np)} \tag{2.6}$$

$cap(i)$ is the processing strength of individual elements and $cores(i)$ is the number of cores required by cloudlet $i$.

### 2.6.2 DynamicCloudSim [44]

DynamicCloudSim introduces external bandwidth and file I/O as additional performance characteristics of tasks, VMs and hosts. It takes into account all performance requirements of tasks for determining how long it takes to execute the task on a given VM.

Execution Time is defined to be:

$$\text{Time} = \frac{\text{Cloudlet Length}}{\text{Assigned MIPS}} \tag{2.7}$$

where the quantities invovled are computed as follows:

$$\begin{aligned} \text{Cloudlet Length} &= mi + io + bw \\ ioTime &= \frac{io}{iops} \\ bwTime &= \frac{bw}{bwps} \\ miTime &= \frac{mi}{min(mips, cores \times \frac{mips}{cpus})} \end{aligned} \tag{2.8}$$

Assuming the larger of the above three to be $bwTime$,

$$\text{Assigned MIPS} = bwps + mi \times \frac{iops}{io} + io \times \frac{bwps}{bw} \tag{2.9}$$

The symbols used indicate the following:
$mi, io, bw$ - CPU, IO and networking requirements of a cloudlet.
$mips, iops, bwps$ - Capabilities of the VM to process each type of resource requirement.

### 2.6.3 CloudSim extension for Three-Tier Applications

In [45] the authors propose an analytical performance model of 3-tier applications. Workload is represented in terms of user sessions unlike in CloudSim and DynamicCloudSim where the basic unit of workload is a cloudlet. Each session incurs performance load on the application and database servers. Sessions make use of CPU time and RAM on the application servers and CPU, RAM, disk I/O on the database servers (disk operations on application servers assumed to be negligible).

A session generates variable workload over time unlike jobs. A session is formally defined using the following variables.

- Ideal session duration: Session duration given all resources for its execution are available
- Data Item: Represents an entity stored on disk (e.g., files, portions of database records). Assume finite number of data items $d_1, d_2, ..., d_n$.
- Step size
- CPU load of application server $v_{as}(t)$
- Memory load of application server $m_{as}(t)$
- CPU load of database server $v_{db}(t, d_k)$
- Memory load of database server $m_{db}(t, d_k)$
- Disk I/O load $d_{db}(t, d_k)$

The last 5 are step (piecewise constant) functions.

Within each step, a session has constant resource requirement in terms of CPU, RAM, I/O. Behavior of a session within a step is represented using one cloudlet executed by AS server and several cloudlets on the DB servers. Each DB cloudlet is associated with the data item it uses. The step functions define the corresponding cloudlets' CPU, RAM, I/O resource requirements.

A session is represented with 2 equal sized queues - one for AS server and one for DB servers. Elements of queue correspond to steps.

Defining the resource consumption of sessions that make up workload involves averaging monitored performance data from actual executions. Performance logs can be used for this purpose.

Execution time for a single cloudlet is computed as follows:

$$\begin{aligned}
\text{Estimated CPU Time} &= \frac{\text{Cloudlet Length}}{\text{capacity} \times \text{cores}} \\
\text{Estimated IO Time} &= \frac{\text{CLoudlet IO Length}}{\text{IO capacity} \times \text{Number of HDDs used by cloudlet}} \\
\text{where IO capacity} &= \frac{\text{IOPS of hard disk}}{\text{number of cloudlets using it}} \\
\text{Estimated Execution Time} &= \min(\text{Estimated CPU Time, Estimated IO Time}).
\end{aligned} \tag{2.10}$$

A survey of the above as well as other simulation tools revealed the following major obstacles in using them for our purposes:

- The workload characterization parameter in our experiments is the number of active user sessions. Most simulators do not provide a way of modeling session based workload. The closest we came across that facilitates this was the third of the simulators discussed above [45].
- Task execution is based on analytical models. For this reason it was difficult to get the performance observations reported by the simulator to agree with the actual performance numbers observed in hardware.

We opted to build our own simulator for these reasons.

Our work also addresses some of the shortcomings in previous work such as assumptions of constant workload and consideration of only controllable resource types such as CPU and memory. Our framework also accomodates those resource dimensions which are difficult to partition such as IO and networking and attempts to exploit them to the benefit of both the cloud provider as well as the client.

# Chapter 3

# PROPOSED FRAMEWORK

We assume that the cloud operator makes available a resource candidate pool to each client application (see Figure 3.1). The candidate pool can be a set of Virtual or Physical machines. This will be a relatively small fraction of the whole cloud infrastructure for privacy and security reasons. An application can assess the options made available to it and choose to make use of a subset.

Figure 3.2 shows an overview of our proposed scheme. Given the workload, the application performance model is used to predict the performance offered by each resource candidate. The cost model assigns a numerical cost to each candidate as well as the current configuration considering the specifications of the resource candidates, their prices and the performance predictions. The reconfiguration algorithm makes use of the cost values to make reconfiguration decisions. This reconfiguration scheme runs periodically to maintain required level of performance.

The individual components are detailed in the following sections. In the rest of this thesis, we will consider the case where the application is hosted on a single machine which can move around (migrate) or expand (vertically scale). The multi-VM (horizontal scaling) situation can be easily accomodated, but we defer this and the relevant experimental validation for future work.

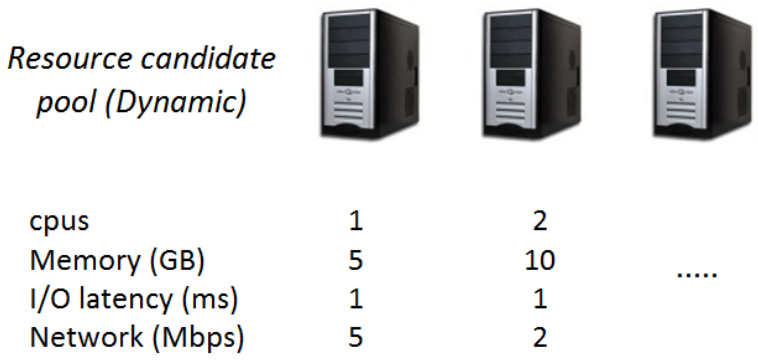



Figure 3.1: Resource candidate pool made available to an application.

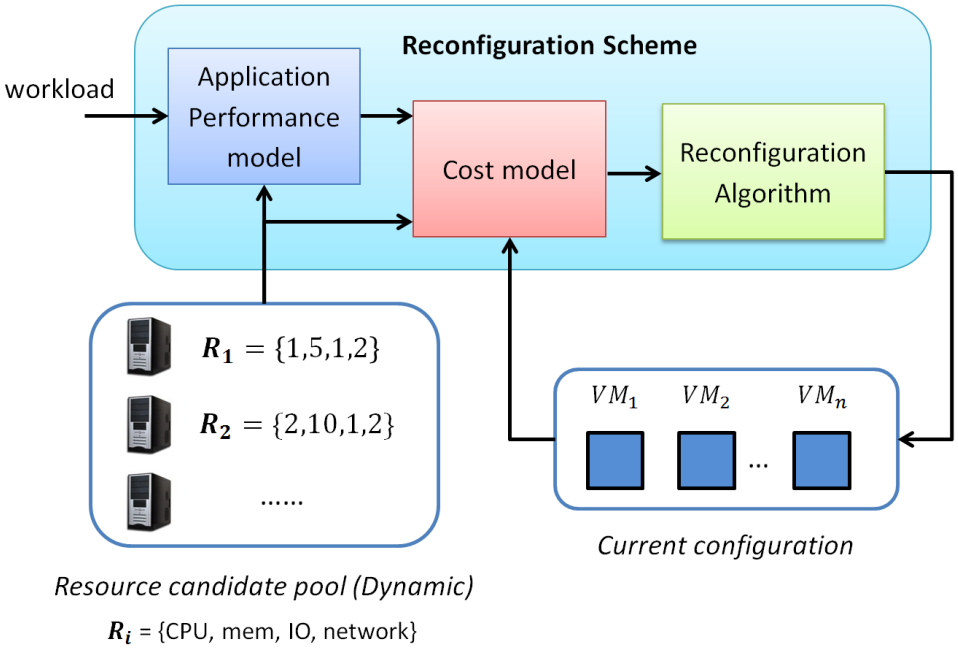



Figure 3.2: Solution overview.

## 3.1 Application Performance Model

The performance of an application at a given point in time depends on the type and quantitiy of workload and the hardware resources available to the application to serve this workload. Capturing the complex relationship between these influencing factors and application performance allows us to make more accurate scaling decisions to suit requirements. Given a particular resource configuration $R$ and workload characterization parameter $w$, an application performance model predicts the performance of the application $Perf\,(R, w)$.

An apparent choice to building such a model is through learning, by observing the performance of the application in an offline experimental environment when subjected to different levels of workload intensity and resource availabilities. Although offline training can be expensive in terms of time and money, an accurate performance model produces significant performance and cost benefits at production time. Training can also be adjusted to work online, at the expense of suboptimal decisions during the beginning. Performance predictions made by the model are used by the cost model to assess different resource configurations.

Table 3.1: List of symbols.

| Symbol | Description |
|---|---|
| $Cost(R)$ | The estimated cost of adopting resource configuration $R$ |
| $Cost_{res}(R)$ | Cost of leasing resources $R$ |
| $Cost_{rec}(R)$ | Cost associated with reconfiguring to a resource configuration $R$ |
| $Income(P,T)$ | Income obtained when application is running with performance $P$ for $T$ period of time |
| $g$ | Monetary loss associated with a particular response time value exceeding $T_{SLA}$ |
| $n$ | Number of active users |
| $Perf(R,w)$ | Predicted performance of the application when provided resources $R$ and subjected to workload $w$ |
| $Perf_{SLA}$ | Level of performance demanded by SLA |
| $P(r > r_0)$ | Probability that response time exceeds $r_0$ |
| $r$ | A particular respone time value |
| $R, R_i$ | A particular resource candidate/configuration |
| $R_{curr}$ | Current configuration |
| $R_{opt}$ | Optimal configuration among options made available to application |
| $T$ | Time period ahead during which we analyze the cost/profits |
| $T_{mig}$ | Time taken for pre-copy phase of migration |
| $T_{down}$ | Downtime during migration |
| $T_{SLA}$ | Target SLA response time |
| $w$ | Workload intensity |
| $\lambda$ | Request arrival rate |

## 3.2 Cost Model

The cost model governs the relationship between a resource allocation/reconfiguration decision and the associated penalty/cost incurred by the application host. It comprises of the following components:

- *Resource Cost* ($Cost_{res}$) – Cost of leasing specified resources.
- *Reconfiguration Cost* ($Cost_{rec}$) – Captures the service degradation or downtime due to reconfiguration actions.
- *Income* – The monetary income associated with providing a particular level of service.

Using these components, we formulate two types of models. The first corresponds to satisfying a performance target specified by a Service Level Agreement (SLA). This encompasses a wide range of applications such as mail servers, search engines, etc.

- Attempt to satisfy SLA requirement at minimal cost

$$Cost(R) = Cost_{res}(R) * T + Cost_{rec}(R) \tag{3.1}$$

From an income perspective, while the above cost model is suitable in a situation where the application host earns a particular income for respecting an SLA, consider an application for which different levels of service have different associated incomes. For an e-commerce website such as ebay, we could assume that income is correlated to perceived performance - better performance facilitates faster purchases. In this case we are interested in maximing profits to the application host.

- Maximize profit (equivalently, minimize cost)

$$Cost(R) = Cost_{res}(R) * T + Cost_{rec}(R) - Income(Perf(R, w), T) \tag{3.2}$$

where the symbols used are described in Table 3.1.

Together these components decide whether the benefits of moving into a new configuration outweigh the momentary fixed cost incurred due to reconfiguration.

### 3.3 Reconfiguration Algorithm

As the workload changes, we decide whether a scaling decision needs to be triggered based on the following optimization objective. For SLA satisfaction, we consider the objective

$$min_R Cost(R) \text{ subject to } Perf(R, wl) > Perf_{SLA} \tag{3.3}$$

whereas for profit maximization, we consider the same objective with the constraint dropped:

$$min_R Cost(R) \tag{3.4}$$

If there exists a configuration ($R_{opt}$) with a lower operating cost than the current one ($R_{curr}$), we move to the new configuration.

To avoid system instability we introduce cooldown periods during which the operating configuration cannot change. These periods are set to expire at a rate proportional to the tradeoff between operating in the configurations $R_{curr}$ and $R_{opt}$ as in equation 3.5.

$$\text{Scale Up Timer} \mathrel{+}= (\text{Target SLA confidence} - \text{Current confidence}) \tag{3.5}$$

This makes sure that we do not reside in unfavorable configurations for too long and that we do not make frequent reconfigurations.

## 3.4 Specific Choices of Models

In the previous section we described a framework for dynamic resource reconfiguration in the cloud. Our approach is modular in that one may plug in specific choices for each of the components depending on the application and requirements. Now we discuss particular choices for these components and evaluate the reconfiguration scheme for these choices.

### 3.4.1 Application Performance Model

We discuss our performance model for a web server application, and use response time as a performance indicator. While most prior work have considered mean and percentile response time values as well as throughput as metrics [32], we consider the response time Cumulative Distribution Function (CDF) as a performance metric for reasons we describe below. Workload is characterized by the number of active user sessions.

The application is deployed in an experimental VM. Performance statistics of the application are collected across different VM configurations and different levels of workload intensity (we refer to these experiments as emulation experiments in the following discussion). The ranges of these parameters are chosen such that they encapsulate the typical production system scenario: The VM configurations can be those offered by the cloud provider, or the options available in a private cloud. Minimum and maximum workload may be determined based on client knowledge of estimates.

The number of such different scenarios can easily be quite large in practice, which makes it time consuming to evaluate each individual configuration. However, sophisticated sampling techniques could be used to choose a manageable subset which provides reasonably good estimates for parts of the problem space which were not considered. For simplicity, we collect statistics for uniformly sampled configurations between the minimum and maximum values for each of the resource types as well as workload.

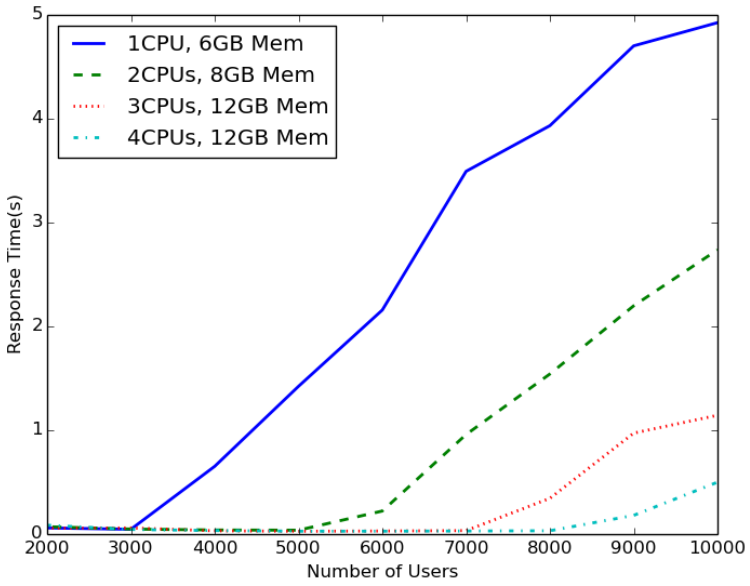



Figure 3.3: Mean response time observed for the RUBiS application in different VM configurations.

Figure 3.3 shows the mean response time curves observed for the RUBiS application in several VM configurations. We observe the following behavior

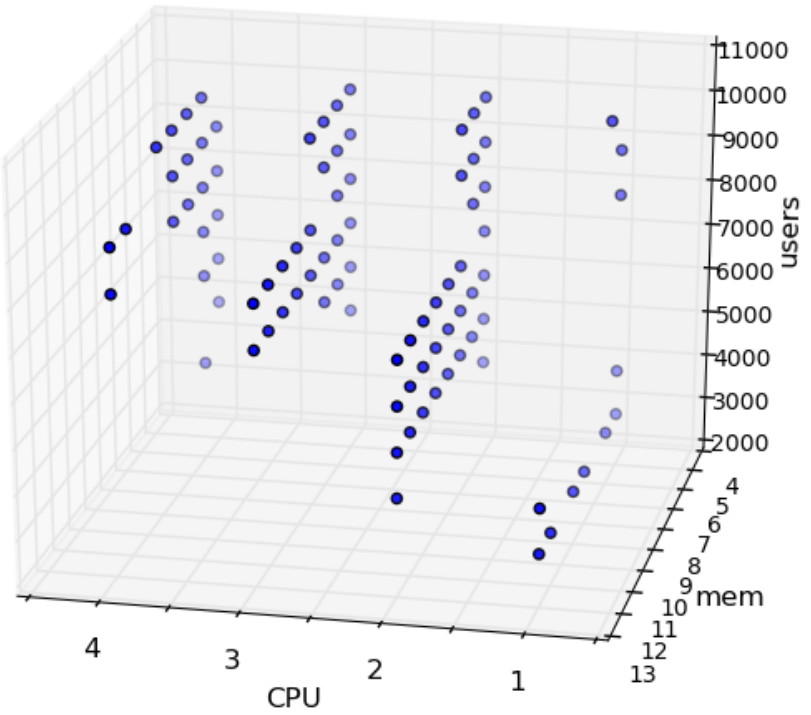



Figure 3.4: The configurations with highest prediction error when a single global model is fit to the data.

consistently in each case. As we increase the number of users, performance stays good until the point where one of the resources becomes a bottleneck, at which point the performance starts degrading rapidly. In effect, we may consider two regions, corresponding to (a) neither of the resources types is fully utilized and (b) at least one of the resource types is fully utilized, in which the performance behavior is consistent within a region but not across. We further made the observation that the response time distributions in the former region were unimodal, whereas it was bimodal in the latter. We observed that this was because requests with modest requirements such as requesting for the home page can still be served fairly quickly when the server is over-utilized, whereas requests involving complex queries were penalized severely. These observations also shows why mean response time is not a very appropriate performance metric in that it fails to capture some crucial aspects of performance.

Figure 3.4 shows VM configurations and number of users for which the largest performance prediction errors were observed when we fit a single global nearest neighbor model. Performance predictions were made using a linear interpolation of all nearby configurations in the training database, and the predictions with kl-divergence [46] error metric $> 0.1$ were plot. We observe from the plot that the largest errors are associated with regions where one of the resources is starting to become a bottleneck, and then beyond that to some extent.

The above observations motivated us to fit different models to the two regions. This idea of fitting different models to different parts of the input space was also advocated by [32] where they analyze the benefit of this approach compared to using a single global model. Within each region, we consider a simple instance-based model for making performance predictions. For a given workload and resource configuration, we predict the response time distribution to be a linear interpolation of nearby configurations.

This simple predictor works quite well. Considering uniformly sampled points from the emulation database to comprise the test set, and a KL-divergence error metric which measures the distance between true and predicted distributions, we observe a mean score of 0.026 for data points in the first region. The fit was not as good in the second region, where we observe a mean score of 0.1. This is because the second mode of the bi-modal distributions observed for configurations in this region moves further to the right as the number of users increases, and this cannot be easily captured by superposing neighboring distributions (whose modes happen to be located quite far away). Capturing this behavior would involve a horizontal shifting or stretching of the neighboring distributions. However, our concern is mainly about modeling accuracy in the former region, since it is where we want to be operating most of the time when the reconfiguration algorithm is operational.

### 3.4.2 Reconfiguration Cost

In the following we overload the term *migration* and use it to mean moving from a particular VM operating configuration to another. This may involve VM migration (in the usual sense) and/or vertical scaling. Live migration refers moving a VM to a different host while it is running, and it takes place in three main stages - the pre-copy phase, stop-and-copy phase and resume phase. During the pre-copy phase, memory pages of the VM are copied to the destination while it is running. This takes place in multiple iterations as some of the already copied pages can become invalid (dirty) and they need to be re-copied. After sufficient data has been copied over (as determined by the particular migration algorithm), the VM is stopped and the remaining dirty pages are copied, after which the VM is resumed at the destination.

Let $T_{SLA}$ be the target SLA time. Let $T_{mig}$ be the time taken for the pre-copy stage of migration. Let $T_{down}$ be the downtime. For virtualization platforms which do not support live migration, we can consider $T_{mig}$ to be 0 and $T_{down}$ to be the total copy time. Assume that $T_{down}$ includes the time period taken by a VM to either resume after migration or reboot after vertically scaling. Consider an operating time horizon $T$ into the future. We evaluate the operating cost during the timespan $T' = T_{mig} + T_{down} + T$ in the following.

As workload increases, if the current VM configuration is no longer able to satisfy the SLA requirement due to insufficient resources, a migration action to a VM with more resources may be considered. There will initially be a service interruption during the migration period, but the benefit gained by migrating to a better VM might compensate for this interruption.

Assume that there is a monetary loss ($g$) associated with a response time value exceeding $T_{SLA}$. This could be, for instance, a monetary penalty associated with the server host violating promised QoS, or a qualitative penalty that captures the fact that large response times will cause user dissatisfaction/churn, etc. We

define the cost of reconfiguration as:

E[Number of user requests which experience a response time greater than $T_{SLA}$ during the $T'$ time period that follows] $\times\ g$ (3.6)

We assume that user requests are distributed according to a Poisson process with arrival rate $\lambda$ and the the number of users $n$ is constant over the duration $T'$ which is a reasonable assumption since $T'$ is in the range of a few minutes.

If we continue to operate in the current configuration, the above quantity is

$$Cost_{rec}(R_{curr}) = [P(r > T_{SLA})(n\lambda T')] \times g \tag{3.7}$$

where P is the response time distribution for current configuration serving the workload over the $T'$ time period and $r$ is a particular response time observation. This is computed from CDF predictions made by our application performance model.

If we choose to migrate,

- During the pre-copy phase which lasts for $T_{mig}$ time, let the perceived response time disribution be $P'$
- During the down time $T_{down}$, service is down
- Response times will be distributed according to distribution $P''$ during the time $T$ in which the new VM is serving load

The cost of a new configuration $R$ is then

$$Cost_{rec}(R) = [P'(r > T_{SLA}) \cdot (n\lambda T_{mig}) + (n\lambda T_{down}) + P''(r > T_{SLA}) \cdot (n\lambda T)] \times g \tag{3.8}$$

We use the analytical model proposed in [40] to capture $P'(r > T_{SLA})$. $P''$ is predicted by the application performance model.

The main factors that influence $T_{mig}$ and $T_{down}$ have been identified as the size of VM memory, memory page dirtying rate and the bandwidth available

for migration [47]. Although analytical models for migration behavior based on these factors have been proposed, dirtying rate of an application as well as the networking performance of cloud VMs can vary significantly over time, which makes it difficult to use these models. Based on empirical observations made by previous studies [39, 48], we found that migration and down times can reach up to 100s and 3s, respectively. We use values sampled from the vicinity of these values for $T_{mig}$ and $T_{down}$. Since we consider the worst case scenario, this provides a lower bound on the performance achievable by our approach. We use VM boot time values reported in [49].

### 3.4.3 Income Model

The client may use any suitable income model based on his requirements. Linear functions and piecewise constant functions are naive choices. For instance, consider a model [41] such as

$$I \times (1 - \frac{r}{T_{SLA}}) \tag{3.9}$$

where $I$ is a constant representing monetary income for ideal performance and $r$ is the mean response time. This is a straight line with negative slope hitting the time axis at $T_{SLA}$ and the income axis at $I$.

# Chapter 4

# PERFORMANCE ANALYSIS

We first describe the experimental setup in Section 4.1. The simulation setup is discussed in Section 4.2. Performance validation using the simulator is described in Section 4.3.

## 4.1 Experimental Setup

### 4.1.1 Hardware and Software Setup

Hardware experiments were performed on the Xen 4.3 virtualization platform running on a machine with Intel i7-4710U CPU (4 cores, 8 threads, 2GHz base frequency), and 16GB RAM. The libvirt 0.9.1 virtualization API [50] was used for VM creation and management. Performance data was collected for VM configurations with CPU and memory varying through 1-4 and 1-12GB, respectively.

### 4.1.2 Application Setup

The RUBiS auction site web application [33] was used as a prototypical application in our experiments. We used the RUBiS PHP incarnation. Application and Database servers were setup on the same VM for simplicity and to avoid network effects. The RUBiS browsing mix was used for our experimental purposes. We used the database dump provided in the RUBiS website as the database. We found the RUBiS setup guide blog post [51] useful for setting up RUBiS.

Apache version 2.4.7 was used as the web-server. The *MaxClients* and *ServerLimit* parameters in Apache were set to sufficiently high values to make sure that the capacity of server VM is not constrained by these parameters. The

*max-connections* parameter (maximum number of concurrent sessions) in MySQL was similarly set to a large value.

### 4.1.3 Workload Emulation

A scalable workload emulator was required to emulate the range of users that we required. The RUBiS Client Emulator is a thread-based Java application which uses one thread per user. As a result the tool did not scale very well, and we had difficulties emulating even a modest number of users properly.

A more scalable option was the RAIN workload generation toolkit [52]. It is customizable to suit any given application and has a more flexible design in that it employs a shared thread pool to emulate users. A RUBiS extension was under development at the time of this work, but was incomplete. (The same authors seem to have completed it recently [53]).

Next we attempted to use httperf [54], a high performance sequential emulator implemented in C and widely used by several previous work for workload emulation. The authors of [55] do a study on the accuracy of results produced by the RUBiS Client emulator. Sessions produced by the Client emulator are captured in a file and replayed using httperf. The results reported by both tools are then compared. They make the observation that the java library used for networking purposes in the client emulator is causing inaccuracies in the results reported. The authors modify httperf to log response time values of requests and to take session inter arrival times as input. The patch is publicly made available in [56].

We encountered difficulties in using the httperf builds from most repositories in emulating a large number of users. Increasing the file descriptor limit in these sources and building causes httperf to crash with glibc buffer overflow errors. This was most likely due to mismatch in the value of FD_SETSIZE between httperf and glibc. The patch [57] works around this issue. We used this patch along with some features of patch [56] such as logging response time values for our experimentation.

The RUBiS Client Emulator was used to generate request sessions, which were logged into a file. The log replay feature was used to replay logs with different numbers of users. A single Intel Core i5 machine was able to emulate upto 6000 users using this version. The emulator machine was running Ubuntu 12.04, and we followed the instructions in [58] to set the system limit parameters appropriately. We were able to emulate upto 10,000 users with two machines.

### 4.1.4 Collecting Performance Data

Each VM configuration was subjected to a load of constant number of sessions. The number of users was varied from 1,000 to 10,000 in steps of 1,000. Each experiment lasted for a duration of 400s with a steady state of 300s (Ramp-up + Ramp-down = 100s). The sysstat package [59] (version 8.0.3) was used to collect performance data from the server VM. We modified the httperf patch to log response time values of individual requests.

Figure 4.1 shows performance statistics collected for a particular VM configuration across different number of users. A uniform increase in utilization is observed for CPU, memory and network as we uniformly increase the number of users to the point where one of the resources becomes a bottleneck. Figure 4.2 shows response distributions and the corresponding cumulative distributions observed for certain configurations. These plots show the behavior of distributions starting to become bi-modal as one of the resource types become a bottleneck.

### 4.1.5 Workload

To evaluate the performance of our alorithm against a realistic workload, we use the widely used World Cup '98 workload trace [60]. It comprises user requests logged over a period of 92 days. We pre-process the trace logs so that the statistical nature of the workload match the RUBiS application. The workload generated in this fashion for the whole span of 92 days is depicted in Figure 4.3. The plot shows the number of active sessions against time. We discuss the

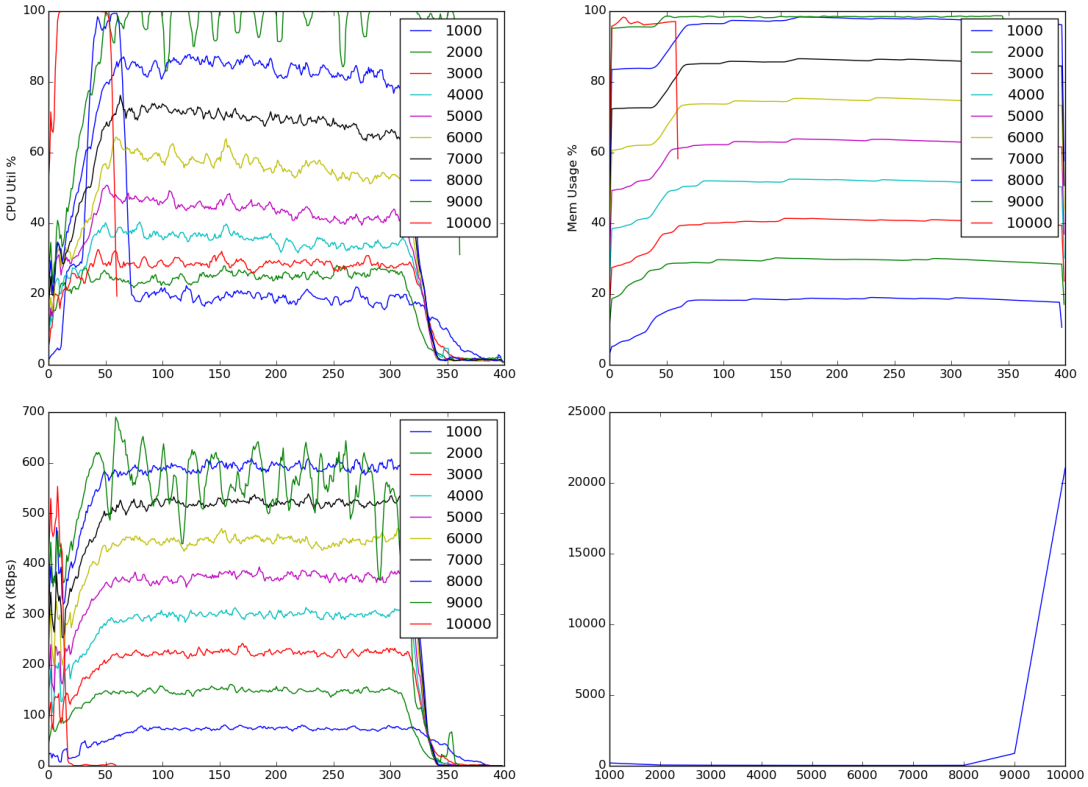



Figure 4.1: Performance Statistics collected for different number of active users under a particular VM configuration.

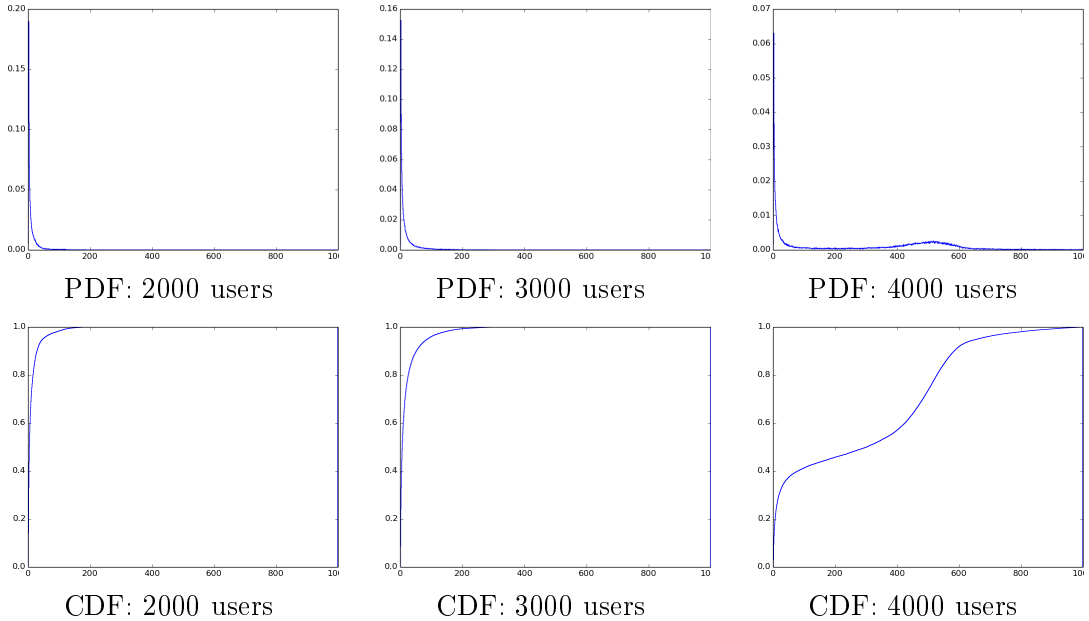



Figure 4.2: Response time distributions observed for different number of active users under a particular VM configurations.

performance of our proposed scheme for the cases of a single day's workload as well as the complete workload.

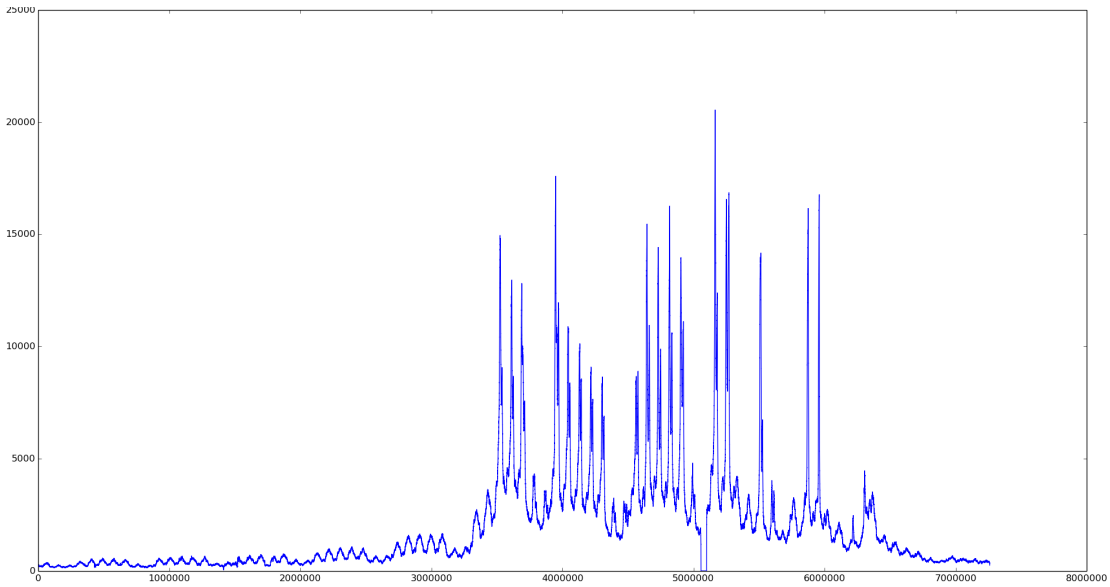



Figure 4.3: Sessions generated from World Cup '98 trace.

## 4.2 Simulation Setup

Simulation models of hardware systems provide a convenient means of anayzing the performance of algorithms. Several simulators have been built to aid cloud research in the past, but we found them unsuitable to our purpose for the following reasons. Most of these simulators are based on analytical models for VM task execution and servicing workloads. With analytical models it is difficult to reflect the performance behavior exhibited by hardware in the simulation model. For instance, it is difficult to embed behavior such as caching, whose influence is evident from our hardware emulation experiments. As a result, we would find our performance prediction model producing results that do not align well with the task execution model of these simulators.

For these reasons we built a discrete event simulator to validate our proposed scaling framework. We use the performance observations recorded from our hardware experiments to model the request servicing behavior in the simulator. We split the simulation duration into distinct intervals, for each of which

performance statistics are computed using the above data. These statistics are then aggregated to compute the overall statistics for the simulation. Note that, while the complete set of performance observations made during emulation experiments are used to build the simulator, only a subset (25%) is used to train the performance model. This assures that the simulation results are not misleading.

Components of the simulator are modeled as follows:

- **Users**: A variable keeps track of the number of users at any given moment.
- **Workload Pattern**: A desired workload variation pattern can be modeled by appropriately varying this variable. For instance, to model simple ramp-up, steady load, and then ramp-down, the variable is increased at required rate until ramp-up time, kept unchanged for steady duration, and decreased for ramp-down time.
- **Web-server VM**: A VM is modeled by a specific set of values for CPU, memory, IO latency and networking performance. The application hosted VM will have particular values for these parameters until change is demanded by a scaling algorithm.
- **Recording Performance Statistics**: Performance over time will be recorded in the following form - $(t_1, N_1, R_1), (t_2, N_2, R_2), (t_3, N_3, R_3), ...$ where each $(t_i, N_i, R_i)$ indicates that an average of $N_i$ users were served by the application with $R_i$ resources during interval $t_i$.

  The intervals are chosen such that:

  1. There is no significant deviation of number of users from $N_i$, and similarly the resources from R (i.e., Number of users $\in (N_i - \delta, N_i + \delta)$)
  2. Our training database has the response time distribution for $(N_i, R_i)$

  As shown in Figure 4.4 for each of these time intervals $t_i$, we have a corresponding response time distribution $dist_i$. The assumption made here

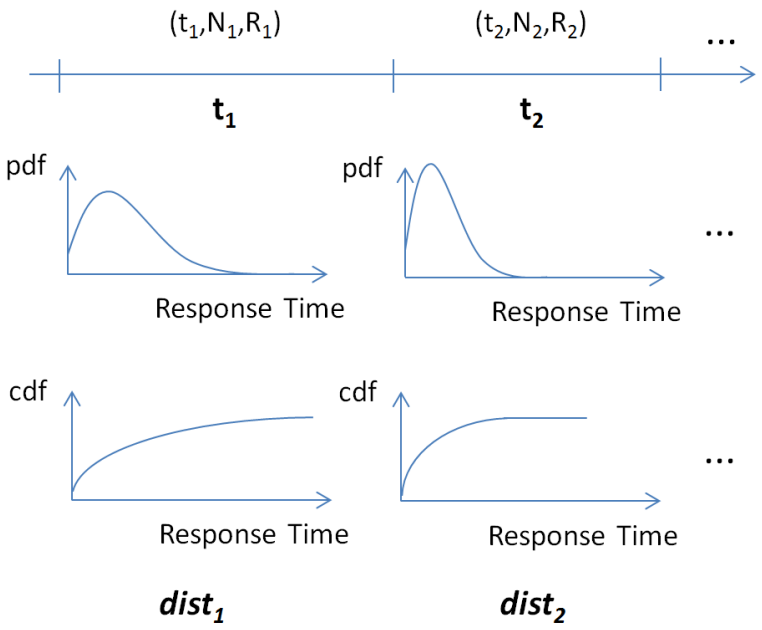



Figure 4.4: Collecting performance statistics in the simulator.

is that the response time distribution of requests does not vary much for small changes in number of users/resources.

- **Overall Performance Statistics**: The overall statistics for the simulation are computed using the statistics for each interval as follows. Assuming that user requests are distributed according to a Poisson process, the expected number of requests made by $N$ users in time period T $= N\lambda T$, where $\lambda$ denotes the arrival rate.

  Over the whole set of response time values observed during an experiment, probability $(p_k)$ that a particular response time value $r$ came from distribution $dist_k$ is given by

$$\frac{N_k \lambda t_k}{\sum_i N_i \lambda t_i} = \frac{N_k t_k}{\sum_i N_i t_i} \tag{4.1}$$

  The overall respose time distribution is then computed as

$$\sum_i p_i \times dist_i \tag{4.2}$$

which is a weighted summation of the distributions $dist_i$, the weights being proportional to the number of requests which presumably came from those distributions.

Distributions corresponding to periods of migration are also calculated as described in Section 3.3 and taken into account in this computation.

To further validate the computation of this overall response time distribution, we do the following. We derive $N_k \lambda t_k$ samples from distribution $dist_k$ for each $k$ and compute the distribution of the collection of all sampled values. This is done several times, and we use the KL-distance metric to verify that the distributions obtained from different simulations are not very different from one another.

## 4.3 Performance Validation

We compare the performance of our reconfiguration scheme against the following schemes that are offered by cloud providers: Static Provisioning and Rule Based Scaling. The Rule Based Scaling mechanism we consider is the following: for each type of resource, if utilization stays higher than a given threshold for a given inertial period, a scale up action is triggered. To scale up, we instantiate new VMs which are of the same type as those in the auto-scaling group.

Parameters of the Rule-based system are set as per the recommendations made by Netflix in [9]. The simulator computes response time distributions assuming that incoming workload is evenly split between the VMs in the auto-scaling group. The cooldown parameters in our reconfiguration algorithm are set identical to the rule-based system.

### 4.3.1 VM sizes from Amazon EC2

We first consider the case where the available VM sizes are those offered by Amazon web services. Instances with a maximum of 4 CPUs and 16 GB memory are considered (shown in Table 4.1) as these were the largest values considered for VMs during performance data collection. Note that prices depend on not

Table 4.1: VM types from Amazon EC2 considered and their prices. These prices vary over time and the figures below are as of Apr 01, 2015.

| VM size CPU cores, Memory(GB) | Cost $/hr |
|---|---|
| (1,1) | 0.013 |
| (1,2) | 0.026 |
| (2,4) | 0.052 |
| (2,7.5) | 0.14 |
| (4,7.5) | 0.28 |
| (2,3.75) | 0.116 |
| (4,15) | 0.232 |

just the amount of CPU and memory but also other factors such as the type of storage. We assume that all these VM sizes are made available as options to the application at all times. The RUBiS application is used for this experiment. Since RUBiS is largely insensitive to IO latency, we assume that the application performance is independent of VM IO performance. Networking performance is omitted as well. The following performance target is considered: Keep response time under 20 ms with 0.9 probability.

Figure 4.5 demonstrates the performance of the proposed approach as well as the rule-based approach with different VM sizes for a particular day's worldcup workload. The different cases considered for rule-based scaling correspond to different types of VMs used in scaling group. For a particular type chosen, the scaling algorithm adds/removes VMs of the same type to the scaling group. The corresponding expense figures and confidence level at which each scaling scheme keeps the response time under the target are shown in Table 4.2 under Workload 1.

Figure 4.6 shows the response time CDFs observed for the complete worldcup workload for the various schemes. The corresponding figures are displayed in Table 4.2 under Workload 2.

We observe that when a small VM size is used for rule-based scaling, the scaling decisions need to be quite frequent for demanding performance requirements, and less so as the VM size increases. The downside of using

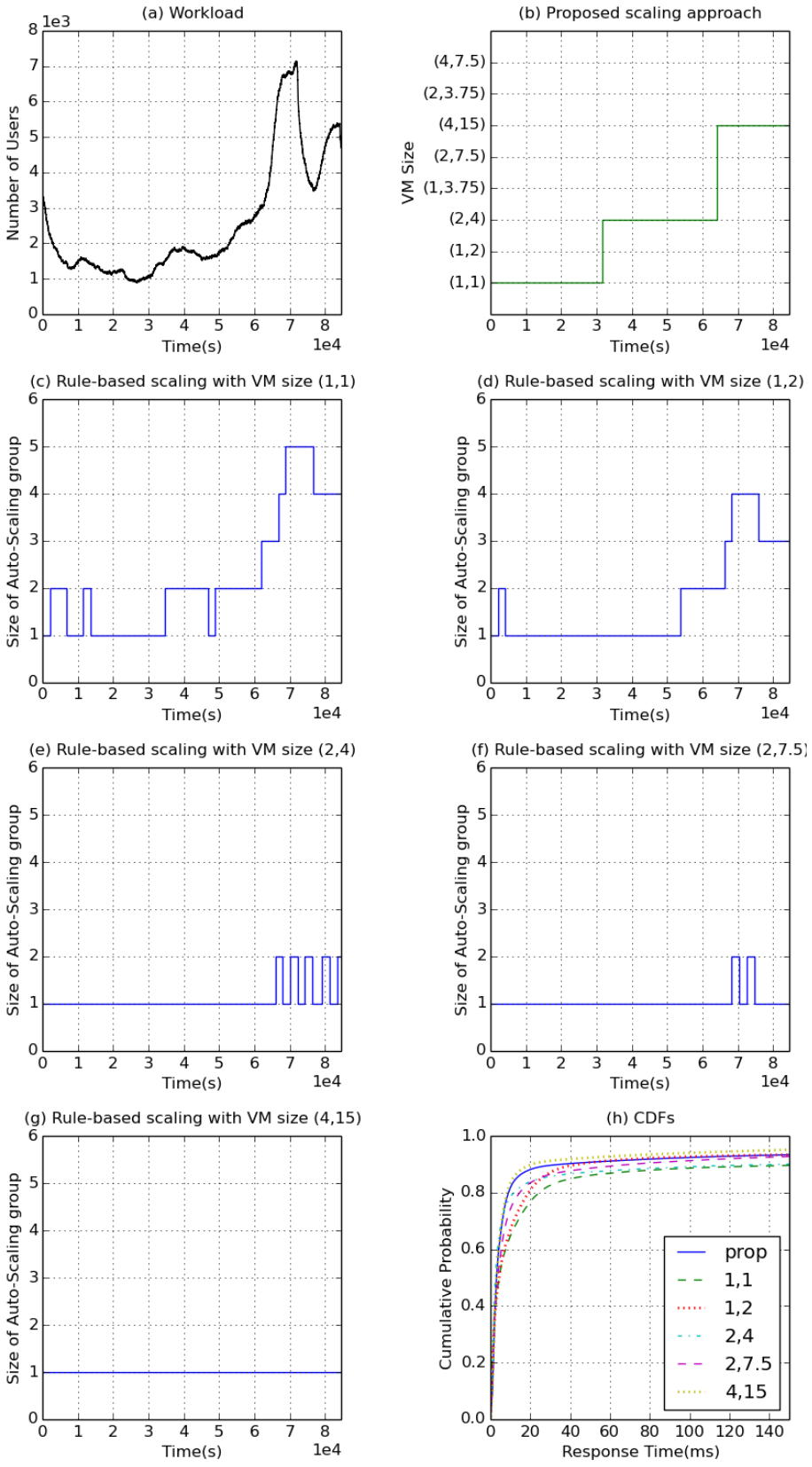



Figure 4.5: Comparison of scaling schemes for a particular day's worldcup workload.

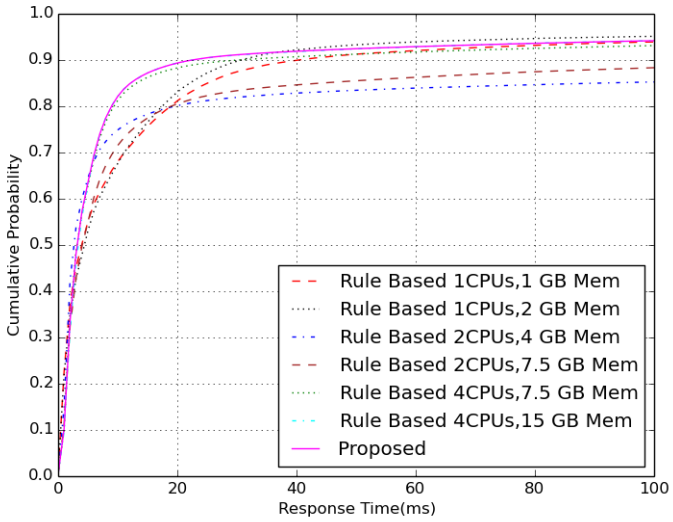



Figure 4.6: Comparison of CDFs for rule-based and proposed scaling schemes - RUBiS application, entire worldcup workload.

Table 4.2: Performance comparison of rule-based scaling with different VM types against the proposed scheme.

| | ***Workload 1*** | | ***Workload 2*** | |
|---|---|---|---|---|
| **VM type (CPUs,Mem(GB))** | **Lease Cost($)** | **P(r < 20ms) %** | **Lease Cost($)** | **P(r < 20ms) %** |
| **Rule-based scaling with different VM types** | | | | |
| (1,1) | 0.70 | 76.8 | 49.63 | 81.20 |
| (1,2) | 1.08 | 81.0 | 79.58 | 83.07 |
| (2,4) | 1.40 | 83.9 | 111.74 | 80.13 |
| (2,7.5) | 3.52 | 83.5 | 290.89 | 80.52 |
| (4,7.5) | 5.70 | 89.4 | 480.96 | 88.30 |
| (4,15) | 6.72 | 89.8 | 564.62 | 89.36 |
| **Proposed scaling scheme** | | | | |
| - | 2.32 | 88.5 | 443.79 | 89.35 |

a larger VM size is an increase in the mean lease cost. As the resonse time distributions and confidence probabilities show, although rule-based scaling can achieve the required performance target with specific instance type choices, in practice we do not know in advance which choice to pick. Our approach is able to maintain a proper balance between cost and performance. A point to note

here is that the proposed scheme achieves this performance with a single VM. Better configurations can be achieved when horizontal scaling is also considered.

The advantages of using a performance model are readily observed. Whereas a rule-based system can identify if a resource bottleneck occurs and increase it, it does not know precisely by what amount to increase the resources. But a performance model knows exactly what amount of resources of each type are required to maintain service to the required level of satisfaction. Although a better set of parameters for the rule-based system may yield a cost benefit, finding a good set of parameters for a given application can be often hard in practice. The proposed scheme falls a little short of the performance target (88.5%, 89.35%), which is due to workload peaks for which even the largest VM considered does not satisfy the performance target. This level of performance was achieved despite migration and downtime influences.

### 4.3.2 A private cloud scenario

Next we consider a private cloud scenario where VM sizes with CPU varying from 1 to 4 and memory from 1 to 16 GB in steps of a unit can be instantiated. In this experiment we model our proposed scenario of a small pool of resources being made available to an application among all vacant options: We assume that a random subset of these VM sizes are made avaialble as options to applications at any given time. We assume a pricing model that linearly varies with the amount of each resource type.

Figure 4.7 shows the variation of CPU and memory as well as the corresponding cost for a particular day's worldcup workload. The corresonding response time CDF is shown in figure 4.8. Following are some observations we made with regard to how the pool of choices made available to an application influences its performance based on the above information.

Since we restrict ourselves to the single VM case in this work, there was a need for sufficiently large instances to be consistently available to the application to maintain the required level of performance. Figure 4.8 shows that the

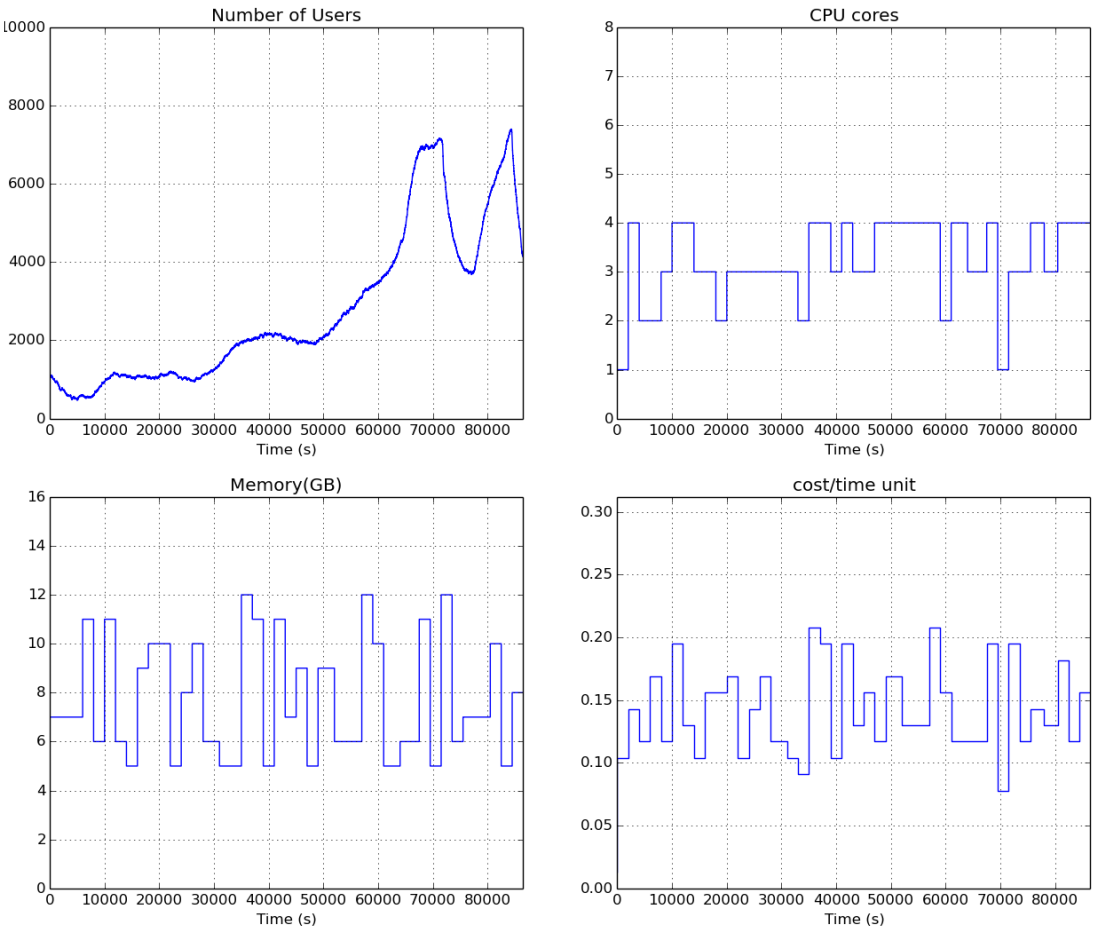



Figure 4.7: A private cloud scenario - Dynamic reconfiguration of CPU and Memory and variation of the corresponding cost.

performance target was not achieved, and this was due to the unavailability of sufficiently large instances during certain periods of time. In the general case, more than one instance will need to be instantiated if a suffciently large option is not available. This relationship between the need for large instances and horizontal scaling needs to be taken into account both from the application's perspective while optimizing for the best reconfiguration and from the provider's point of view considering efficient datacenter utilization. If, for instance, the provider wants to minimize the number of hosted instances for management reasons, it is preferable for the provided set of options to roughly span the range of possible resource values.

Another observation that we made is that with rapid variation in the pool of resources, the number of reconfigurations is high (as in Figure 4.7). This is

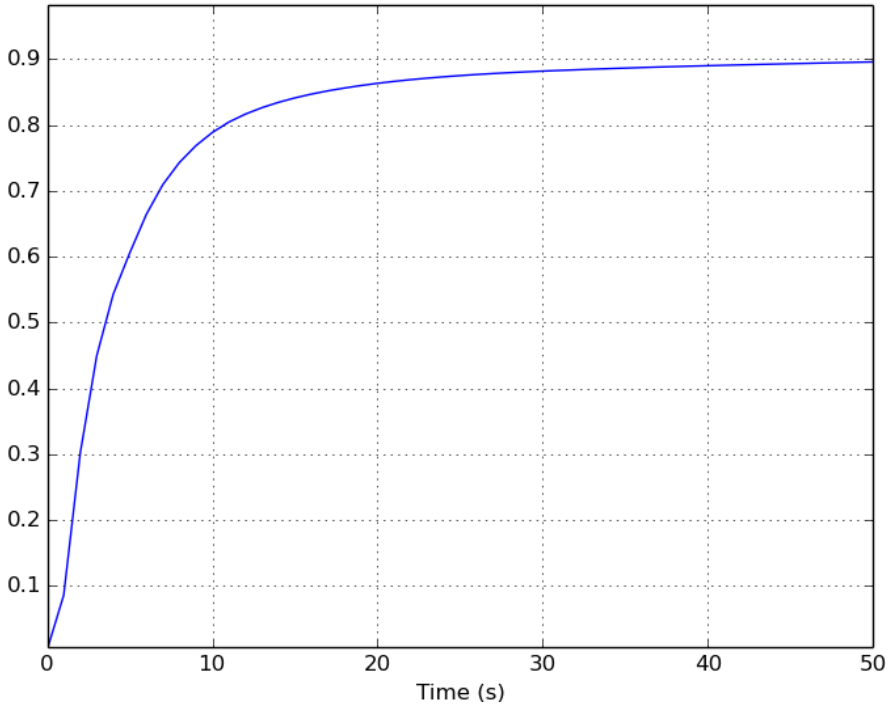



Figure 4.8: A private cloud scenario - Response time CDF.

because a particular configuration fails to remain optimal over a considerable period of time when there is such variation. With a large number of applications with distinct characteristics sharing the cloud, we can expect the available resources, and hence those made available to applications, to be highly dynamic in nature (especially those resource dimensions that are difficult to control). This motivates the cost model to have a term that explicitly penalizes reconfigurations in addition to the cooldown periods in the reconfiguration algorithm. We hope to further pursue this idea rigorously in future.

### 4.3.3 Application affected by IO performance

We use the filebench-fileserver benchmark workload [34] to assess the potential benefits of our approach with respect to IO availability and performance impact. We modify the filebench application to log the response time values for individiual IO requests and derive the read/write latency distributions. To change the perceived IO latency of the application hosted VM, we run fio [61], an IO workload generation tool, on a different VM to create competing IO workload

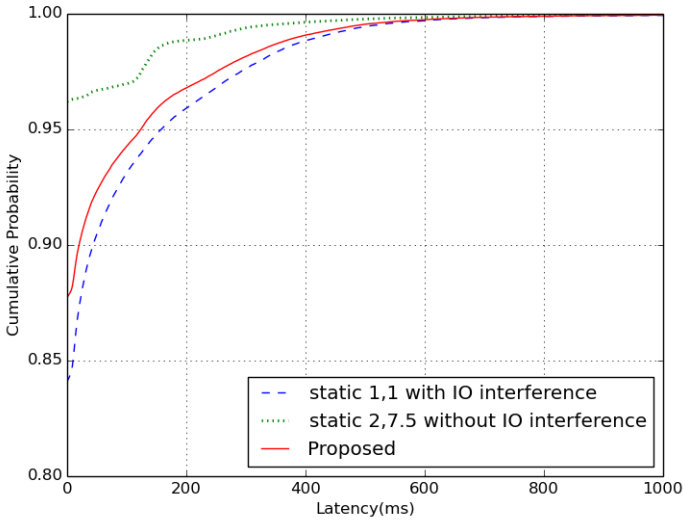



Figure 4.9: Comparison of CDFs for static provisioning and proposed scaling scheme - filebench-fileserver application driven by simple synthetic workload.

as in [32]. Two levels of IO stress are considered - a) no competing IO and b) fio emulating random IO with 10 workers. Performance data was collected for fileserver workload varying from 10 to 40 threads.

The EC2 VM sizes were used for this experiment. The IO performance of these VMs is modeled to fluctuate randomly between two levels corresponding to the two levels of IO stress discussed above. We consider a basic workload pattern that oscillates within the range of workload values for which performance data was collected. We accomodate IO performance in the pricing model by pricing a VM at its full price when there is no IO interference and half its price when IO stress exists. The performance target considered is: $90^{th}$ percentile latency $<$ 20ms.

It was observed during performance data collection that CPU and Memory utilizations were relatively low, and the workload was mainly bounded by IO performance. Hence, we compare the proposed scheme with the extreme static provisioning cases. Figure 4.9 compares the response time CDFs. Corresponding performance and cost figures are presented in Table 4.3. We observe that the small VM with IO interference is unable to meet the performance target. The proposed scheme achieves the performance target and produces cost savings of 10% relative

to the best static VM and 30% relative to the largest static VM according to the aforementioned pricing policy. Although this is a very preliminary proof of concept that our approach works well in the case of IO bound applications, it strongly suggests that we should further explore this venue using experiments in hardware.

Table 4.3: Different schemes and corresponding performance, cost figures.

| Scheme | P(latency < 20ms) | Lease Cost ($) |
|---|---|---|
| Static (1,1) with IO interference | 87% | 0.16 |
| Static (1,2) without IO interference | 90% | 0.62 |
| Static (2,7.5) without IO interference | 97% | 3.36 |
| Proposed | 90% | 0.56 |

### 4.3.4 Profit Maximization

We now consider profit maximization as the objective instead of meeting performance targets. We consider the simple linear income model discussed previously in equation 3.9. As for the choice of parameter $I$, this very much depends on the specific client application. It is the monetary income attributable to a single user request. This quantity varies across different types of applications, as well as across time for the same application. For instance [62] reports that LinkedIn gets $1.30 in revenue for every hour and Facebook gets 6.2 cents. The mean income observed per request for a running application till that point in time is a reasonable choice for the parameter.

For this experiment we use a constant value of 1 cent for the constant $I$. The overall profits for a simulation are calculated as the following expected value

$$\sum_{i}^{k}(n_i \lambda t_i) I (1 - \frac{E_i[r]}{T_{SLA}}) \tag{4.3}$$

where $n_i$ represents the average number of users observed during the $i$th interval, $t_i$ denotes the length of the interval, and $E_i$ is the expectation taken over the $i$-th interval.

Table 4.4: Profit Maximization - Performance comparison of rule-based scaling with different VM types against the proposed scheme.

| VM type (CPUs,Mem(GB)) | Profit ($) |
|---|---|
| **Rule-based scaling with different VM types** | |
| (1,1) | 384 |
| (1,2) | 233 |
| (2,4) | 250 |
| (2,7.5) | 235 |
| (4,7.5) | 204 |
| (4,15) | 175 |
| **Proposed scaling scheme** | |
| - | 313 |

Table 4.4 shows a comparison of the income observed for rule-based scaling with different types of VMs as well as the proposed scheme. These figures were produced by deducting the overall VM lease cost from the expression in equation 4.3. Rule-based scaling with the (1,1) type VM produced the most income. The proposed scheme follows next, and is ahead of rule-based scaling with other VM types. There is marginal difference between the rule-based scaling scenarios with the other VM types. Although rule-based scaling does not explicitly attempt to maximize income, the correlation between resource utilization, good performance and profits enables it to produce figures comparable with the proposed scheme.

The possibility of employing multiple VMs enables rule-based scaling to produce better results than the proposed scheme. As mentioned in the other performance analysis sections as well, although a particular case of rule-based scaling does produce good results, we do not know this information in advance. On the other hand, considering the multi-VM case in our proposed framework can potentially yield better results.

# Chapter 5

# CONCLUSIONS AND FUTURE WORK

## 5.1 Conclusions

Cloud computing has become an attractive venue to enterprises and users alike during recent times. This growth in interest has been accompanied by the need to address problems related to resource scaling and datacenter management.

Our novel framework enables cloud hosted user applications to dynamically scale while consuming just the adequate amount of resources along different types of resource dimensions. We consider a scenario where the cloud operator exposes a pool of resource options to each application, of which the application has the choice to make use of a subset. The application reconfigures itself guided by performance and cost models. We showed how this provides more flexibility to applications in adopting favorable resource configurations and offers cost advantages. As more efficient techniques for live migration and vertical scaling evolve, our approach becomes increasingly attractive.

We demonstrated that our approach is

- Computationally simple - Each application considers only a handful of resource options to reconfigure itself.
- Generic - Different types of models can be plugged into the application performance, income and cost models. We perform experiments with specific choices for these models and discuss performance results.
- Beneficial to both the cloud provider as well as the user - Our scheme naturally exploits the co-hosted application interference problem in shared resources. Instead of determining and controlling performance interference due to co-located applications, the applications themselves figure out which of the vacant resource options are suitable to meet performance needs.

The proposed approach achieves cost savings of more than 20% relative to the rule-based scaling schemes for the world cup workload. Our preliminary experiments with an application sensitive to IO performance show cost savings of 10% relative to the best static VM configuration. It should be noted that these figures were achieved assuming a single operating VM. Considering the multi-VM case can potentially lead to even better results.

We identified several significant next steps that can be taken from our work that we were unable to address. We discuss these possible improvements and other related future avenues of research in the next section.

## 5.2 Future Work

One of the restrictions in our current implementation of the proposed approach is that we only consider the case where the application is hosted in a single VM. The multiple VM case is more representative of the real situation. We can expect to see significant benefits in this case as the single VM case by itself offers considerable advantages. The main challenges involved in doing this would be accurately modeling the tradeoffs between horizontal and vertical scaling as well as migration in the reconfiguration cost component, and handling the exponential increase in number of possible configurations.

A significant next step would be to validate our approach in hardware. Testing using a local environment would involve setting up a hardware cluster and the virtualization platform. The simulation experiments need to be run in hardware and the results have to be compared against the simulation results. This would provide more credibility to the simulation results and show that our approach works well in hardware.

We experimented with applications whose performance is primarily determined by CPU and memory availability. We provided a basic evaluation of an application for which I/O performance is the bottleneck. Validating our approach for an application with networking performance impact is another possible next step. In this case one has to take into account concerns such

as the following: applications sensitive to networking performance would be affected significantly by live migration compared to other application types, the behavior of the reconfiguration algorithm has to be investigated as more resource dimensions are taken into account.

The proposed approach has been mostly analyzed from a user's perspective. Although we have discussed that the cloud provider benefits from the proposed approach, it would be interesting to investigate the provider's side of the story objectively. One may simulate a scenario where several applications are running on the cloud, each of them adopting the proposed dynamic reconfiguration scheme. We could analyze factors of concern to the cloud provider such as datacenter utilization and energy impact in this situation.

An important research direction that we found to have received less focus from the research community in the past is the design of a scalable, general-purpose workload emulator. Some of the issues we encountered in using existing emulators such as the RUBiS Client emulator, RAIN and httperf were discussed in Section 4.1.3. For thread based emulators memory proved to be a bottleneck. Although httperf is a high performance sequential emulator, it only exploits one processing core, and there is more room to exploit in a multi-core machine. Most prior work have considered workloads that are small which they are able to emulate using a few machines. The solution adopted to emulating more workload has been to use more machines for emulation. Therefore, designing a workload emulator that is highly scalable and is general purpose would be benificial to the research community.

## References


[1] Michael Armbrust, Michael Armbrust, A Fox, A Fox, R Griffith, R Griffith, AD Joseph, AD Joseph, RH, and RH. Above the clouds: A Berkeley view of cloud computing. *University of California, Berkeley, Tech. Rep. UCB*, pages 07–013, 2009.

[2] Amazon ec2 instance types. `http://aws.amazon.com/ec2/instance-types`. Accessed: 2015-04-01.

[3] Amazon web services - auto scaling. `http://aws.amazon.com/autoscaling/`. Accessed: 2014-10-20.

[4] Tania Lorido-Botran, Jose Miguel Alonso, and Jose A. Lozano. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing*, (1):1–34, 2014.

[5] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-Up. In *In Proc. of the Annual IEEE/ACM International Symposium on Microarchitecture*, page 248, 2011.

[6] Rui Han, Li Guo, Moustafa Ghanem, and Yike Guo. Lightweight resource scaling for cloud applications. In *CCGRID*, pages 644–651. IEEE, 2012.

[7] Masum Z. Hasan, Edgar Magana, Alexander Clemm, Lew Tucker, and Sree Lakshmi D. Gudreddi. Integrated and autonomic cloud resource scaling. In *NOMS*, pages 1327–1334. IEEE, 2012.

[8] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Enacting slas in clouds using rules. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, *Euro-Par (1)*, volume 6852 of *Lecture Notes in Computer Science*, pages 455–466. Springer, 2011.

[9] Auto scaling in the amazon cloud. `http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html`. Accessed: 2015-03-20.

[10] Peter Bodík, Rean Griffith, Charles A. Sutton, Armando Fox, Michael I. Jordan, and David A. Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *Workshop on Hot Topics in Cloud Computing, HotCloud'09, San Diego, CA, USA, June 15, 2009*, 2009.

[11] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[12] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: towards a fully automated workflow. In *Seventh International Conference on Autonomic and Autonomous Systems, ICAS 2011*, pages 67–74. IEEE, May 2011. MoVe INT LIP6.

[13] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Le Yi Wang, and Gang George Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In Simon A. Dobson, John Strassner, Manish Parashar, and Onn Shehory, editors, *ICAC*, pages 137–146. ACM, 2009.

[14] Xavier Dutreilh and Sergey Kirgizov. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. *International Conference on Autonomic and Autonomous Systems*, (c):67–74, 2011.

[15] Jia Rao, Xiangping Bu, Cheng zhong Xu, Leyi Wang, and George Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *In ICAC*, 2009.

[16] Harold C. Lim, Shivnath Babu, Jeffrey S. Chase, and Sujay S. Parekh. Automated control in cloud computing: Challenges and opportunities. In *In First Workshop on Automated Control for Datacenters and Clouds*, 2009.

[17] Harold Lim, Shivnath Babu, and Jeffrey S. Chase. Automated control for elastic storage. In Manish Parashar, Renato J. Figueiredo, and Emre Kiciman, editors, *ICAC*, pages 1–10. ACM, 2010.

[18] Bhuvan Urgaonkar, Prashant J. Shenoy, Abhishek Chandra, Pawan Goyal, and Timothy Wood. Agile dynamic provisioning of multi-tier internet applications. *TAAS*, 3(1), 2008.

[19] Daniel A. Villela, Prashant Pradhan, and Dan Rubenstein. Provisioning servers in the application tier for e-commerce systems. *ACM Trans. Internet Techn.*, 7(1), 2007.

[20] Hien Nguyen Van, Frédéric Dang Tran, and Jean-Marc Menaud. Sla-aware virtual resource management for cloud infrastructures. In *International Conference on Computer and Information Technology, Xiamen, China*, pages 357–362, 2009.

[21] Makhlouf Hadji and Djamal Zeghlache. Minimum cost maximum flow algorithm for dynamic resource allocation in clouds. In *International Conference on Cloud Computing, Honolulu, HI, USA*, pages 876–882, 2012.

[22] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management*, pages 119–128. IEEE, 2007.

[23] Nicolo Maria Calcavecchia, Ofer Biran, Erez Hadad, and Yosef Moatti. VM Placement Strategies for Cloud Scenarios. *International Conference on Cloud Computing*, pages 852–859, June 2012.

[24] Emiliano Casalicchio, Daniel a. Menascé, and Arwa Aldhalaan. Autonomic resource provisioning in cloud systems with availability goals. *Proceedings of*

*the 2013 ACM Cloud and Autonomic Computing Conference on - CAC '13*, page 1, 2013.

[25] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. Efficient resource provisioning in compute clouds via VM multiplexing. *Proceeding of the International conference on Autonomic computing*, page 11, 2010.

[26] Ajay Gulati, Ganesha Shanmuganathan, Anne M. Holler, and Irfan Ahmad. Cloud scale resource management: Challenges and techniques. In *USENIX Workshop on Hot Topics in Cloud Computing Portland, OR, USA*, 2011.

[27] David Erickson, Brandon Heller, Nick McKeown, and Mendel Rosenblum. Using network knowledge to improve workload performance in virtualized data centers. In *International Conference on Cloud Engineering, Boston, MA, USA*, pages 185–194, 2014.

[28] Chunqiang Tang, Malgorzata Steinder, Michael Spreitzer, and Giovanni Pacifici. A Scalable Application Placement Controller for Enterprise Data Centers. *Proceedings of the 16th international conference on World Wide Web - WWW '07*, page 331, 2007.

[29] Ron Chiang, Jinho Hwang, Howie Huang, and Timothy Wood. Matrix: Achieving predictable virtual machine performance in the clouds. In *ICAC*. IEEE Computer Society, 2014.

[30] Brian J. Watson, Manish Marwah, Daniel Gmach, Yuan Chen, Martin Arlitt, and Zhikui Wang. Probabilistic performance modeling of virtualized resource allocation. *Proceeding of the International conference on Autonomic computing*, page 99, 2010.

[31] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L. Wiener, Armando Fox, Michael Jordan, and David Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In

*Proceedings - International Conference on Data Engineering*, pages 592–603, 2009.

[32] Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta. Modeling virtualized applications using machine learning techniques. *Proc. of the ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments*, page 3, 2012.

[33] Rubis: Rice university bidding system. `http://rubis.ow2.org/`. Accessed: 2014-08-05.

[34] Filebench: a framework for simulating applications on file systems. `http://www.solarisinternals.com/wiki/index.php/FileBench`. Accessed: 2015-05-20.

[35] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Proc. of the International Conference on Cloud Computing*, pages 500–507, 2011.

[36] Performing vm migration under xen. `http://wiki.xenproject.org/wiki/Migration`. Accessed: 2015-03-16.

[37] Vmware vsphere. `https://www.vmware.com/products/vsphere/features/vmotion`. Accessed: 2015-03-16.

[38] Akshat Verma, Gautam Kumar, and Ricardo Koller. The cost of reconfiguration in a cloud. In *Proceedings of the 11th International Middleware Conference Industrial track*, pages 11–16. ACM, 2010.

[39] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. *Lecture Notes in Computer Science*, 5931 LNCS:254–265, 2009.

[40] David Breitgand, Gilad Kutiel, and Danny Raz. Cost-aware live migration of services in the cloud. *Proc. of the ACM International Systems & Storage Conference*, page 1, 2010.

[41] Guofu Feng, Saurabh Garg, Rajkumar Buyya, and Wenzhong Li. Revenue Maximization Using Adaptive Resource Provisioning in Cloud Computing Environments. *2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 192–200, September 2012.

[42] Li Zhang and Danilo Ardagna. Sla based profit optimization in web systems. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 462–463. ACM, 2004.

[43] Rodrigo N. Calheiros, Rajiv Ranjan, CÃľsar A. F. De Rose, and Rajkumar Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *CoRR*, abs/0903.2525, 2009.

[44] Marc Bux and Ulf Leser. Dynamiccloudsim: Simulating heterogeneity in computational clouds. In *Proceedings of the 2Nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, SWEET '13, pages 1:1–1:12, New York, NY, USA, 2013. ACM.

[45] Nikolay Grozev and Rajkumar Buyya. Performance Modelling and Simulation of Three-Tier Applications in Cloud and Multi-Cloud Environments. *The Computer Journal*, pages bxt107–, 2013.

[46] Kullback-liebler divergence. `https://en.wikipedia.org/wiki/Kullback-Leibler_divergence`. Accessed: 2015-06-01.

[47] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. In *HPDC*, pages 171–182. ACM, 2011.

[48] Felix Salfner, Peter Tr, and Andreas Polze. Downtime Analysis of Virtual Machine Live Migration. *International Conference on Dependability*, (c):100–105, 2011.

[49] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In Rong Chang, editor, *IEEE CLOUD*, pages 423–430, 2012.

[50] libvirt virtualization api. http://libvirt.org/. Accessed: 2014-07-25.

[51] Rubis workload: Simple installation guide. http://sanifool.com/2012/09/03/rubis-workload-simple-installation-guide. Accessed: 2014-11-10.

[52] Aaron Beitch, Brandon Liu, Timothy Yung, Rean Griffith, Armando Fox, and David Patterson. Rain: A workload generation toolkit for cloud computing applications. *Technical Report No. UCB/EECS-2010-14*, 2010.

[53] Marco Guazzone. The rubis workload implementation for rain. https://github.com/sguazt/rain-workload-toolkit, 2013. Accessed: 2015-01-22.

[54] David Mosberger and Tai Jin. Httperf—A Tool for Measuring Web Server Performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37, December 1998.

[55] Raoufehsadat Hashemian, Diwakar Krishnamurthy, and Martin Arlitt. Web workload generation challenges–an empirical investigation. *Software: Practice and Experience*, 42(5):629–647, 2012.

[56] httperf patch. http://people.ucalgary.ca/~dkrishna/SPE. Accessed: 2014-11-21.

[57] httperf patch. https://github.com/klueska/httperf. Accessed: 2015-02-30.

[58] Linux: Increasing the number of open file descriptors. https://cs.uwaterloo.ca/~brecht/servers/openfiles.html. Accessed: 2015-02-22.

[59] sysstat. http://sebastien.godard.pagesperso-orange.fr. Accessed: 2014-07-11.

[60] Worldcup 98. http://ita.ee.lbl.gov/html/contrib/WorldCup.html. Accessed: 2014-12-10.

[61] fio: Flexible i/o tester. http://freshmeat.net/projects/fio. Accessed: 2015-05-15.

[62] How linkedin gets twenty times more money per user than facebook. http://www.businessinsider.com/linked-revenue-facebook-2012-7. Accessed: 2015-07-12.