Adaptive Solutions to Resource Provisioning and Task Allocation Problems for Cloud Computing

by

Ronald J. Desmarais
B.S.Eng., University of Victoria, 2006

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

Adaptive Solutions to Resource Provisioning and
Task Allocation Problems for Cloud Computing

by

Ronald J. Desmarais
B.S.Eng., University of Victoria, 2006

Supervisory Committee

Dr. Hausi A. Müller, Supervisor
(Department of Computer Science)

Dr. Venkatesh Srinivasan, Departmental Member
(Department of Computer Science)

Dr. Randall Sobie, Outside Member
(Department of Physics and Astronomy)

Dr. Kin F. Li, Outside Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Hausi A. Müller, Supervisor
(Department of Computer Science)

Dr. Venkatesh Srinivasan, Departmental Member
(Department of Computer Science)

Dr. Randall Sobie, Outside Member
(Department of Physics and Astronomy)

Dr. Kin F. Li, Outside Member
(Department of Electrical and Computer Engineering)

ABSTRACT

With the emergence of the cloud computing paradigm, we can now provide dynamic resource provisioning at unprecedented levels of scalability. This flexibility constitutes a rich environment for researchers to experiment with new designs. Such experimental novel designs can take advantage of adaptation, controllability, self-configuration, and scheduling techniques to provide improved resource utilization while achieving service level agreements. This dissertation uses control and scheduling theories to develop new designs to improve resource utilization and service level agreement satisfaction. We optimize resource provisioning using the Cutting Stock problem formulation and control theory within feedback frameworks. We introduce a model-based method of control to manipulate the scheduling problem's formulation model to achieve desired results. We also present a control based method using Kalman filters for admission control. Finally, we present two case studies — the Yakkit media social application and the Rigi Cloud testbed for deploying virtual machine experiments. The results of our investigations demonstrate that our approaches and techniques can optimize resource utilization, decrease service level agreement violations, and provide scheduling guarantees.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Hausi — I wish to thank my Advisor for mentoring, support, encouragement, and patience, but most of all for his vision and friendship. Well done Hausi!

Randy — I wish to thank my first boss, who opened the world of grid and cloud computing to me as both a coop student and an employee. Thank you Randy!

NSERC and IBM CAS — My first real experience with industry was through IBM's Center for Advanced Studies. I would like to thank Marin Litoiu and Kelly Lyons for their advice and mentor-ship. Finally I wish to thank the Canadian NSERC program for providing financial support.

Rigi Group — I wish to thank members of Rigi Group: Przemek Lach for our endless arguments, Andreas Bergen and Pratik Jain for the memories, Nina Taherimaksousi for our discussions on what is a novel contribution, Lorena Castaneda for her practicality, Norah Villegas for raising the standard, Sowmya Balasubramanian for proving that you can be a mother, wife and grad student all at the same time, Marcus Csaky for being cool and working at IBM, Ishita Jain for her questions of which I felt honored to answer, Priyanka Agrawal for being smart like Ishita, Atousa for her dedication, Dylan for teaching the kids it's OK to eat dirt, Scott Brousseau for asking the advice of a Killick, Quan Yang for her great vision like Haüsi, Sweta for the good old days, Piotr for teaching and being the best compute dude I know, Alexey for 'Spasiba', Jochen for being Jochen and that really cool 3-D modeling simulation thing, and lastly Qin Zhu for always liking the stuff I was building and giving me a hand for testing it out!

HEP NET Group — I would also like to mention members of Dr. Randall Sobie's grid research group, specifically, Ian Gable the network manager, Patrick Armstrong, Frank Berghaus and Dan Vanderster for accelerating my interest in using simulation.

CSC Staff — I would like to mention the hard work of our graduate secretary Wendy, my techno buddy Tomas Bednar and Brian Douglas.

do or do not there is no try
Master Jedi Yoda to Luke Skywalker - Star Wars - the Empire Strikes Back

DEDICATION

I wish to dedicate this dissertation to Valerie MacNeil, Bhreagh MacNeil-Desmarais, Linda Desmarais and Rene Desmarais for their support during my time as a Graduate Student! As well I would like to thank Bixby C. At and his two sisters Bijou and Paully.

...

Chapter 1

Introduction

1.1 Motivation

With the emergence of the cloud computing paradigm, we can now provide dynamic resource provisioning at unprecedented levels of scalability. This paradigm provides a rich environment for researchers to experiment with new designs. These experimental designs can take advantage of adaptation, controllability, self-configuration, and scheduling techniques to provide improved resource utilization for cloud providers, while achieving service level agreements (SLAs) for cloud users. This dissertation uses control and scheduling theories within a feedback framework to develop new designs for improving resource utilization, SLA satisfaction, and taking advantage of the cloud's dynamic nature. For example, we use a scheduling formulation within a feedback framework in a simulation environment to evaluate several provisioning scenarios. The results indicate that these techniques can provide improved resource utilization and decrease service level agreement violations.

The cloud provides users with access to large amounts of computational resources, bandwidth, and storage [RCL09]. It is replacing older computing paradigms such as Grid and Cluster computing. Its success is due to the bridging of virtualization software with dynamic on-demand provisioning systems (e.g., Amazon's EC2, Microsoft's Azure, Google's App Engine, IBM's Smart Cloud, or Apple's iCloud) which can take advantage of virtual infrastructure to deploy their services and applications.[1] The term 'Cloud' in this dissertation most often refers to infrastructure as a service

[1] http://www.informationweek.com/cloud-computing/infrastructure/amazon-cloud-revenue-mystery-persists/240153962

(IaaS) and occasionally as software as a service (SaaS). These two layers of the cloud paradigm are interesting from a research perspective because they require solutions for runtime adaptation.

Cloud services manage operating system environments, application platforms, and install and negotiate access to hosted applications. These pay-for-use services consume resources that are purchased like any other utility [BYV+09]. This provides opportunities to explore new designs which can take advantage of a pay-as-you-go utility. These designs use dynamic theories, including control theory and feedback models combined with scheduling theories, to improve resource utilization, maintain service level agreements, and minimize costs [BDG+13] [AAB+07] [HDPT04] [AAB+10] [HDPT05] [ADG+06] [AAC+10] [DDKM08] [DM07] [FAA+10] [LNM+12] [PSS+12] [DMK+13] [KAB+12] [GB12] [MK12] [NCS12] [TMMVL12] [GSLI12] [WKGB12] [Mur03].

The foundations of cloud computing emanate from distributed computing, virtualization, and economics. Distributed computing system topologies can be categorized as hierarchical (Grid Computing), centralized (Email), or decentralized (Peer to Peer) [DNB05]. The objective of distributed systems is to provide aggregation of distributed resources for access by users. However, they differ in how they achieve accessibility. Virtualization, in this dissertation, refers to the management of hardware access by multiple competing operating systems on the same machine. These operating environments run in an isolated sandbox from other operating environments on the same machine (e.g., VMware[2] or Xen.[3]) Lastly, economic considerations form the basis for computing as a utility. Rajkumar et al. refer to cloud computing as the fifth utility next to water, electricity, gas, and telephony [BYV+09].

1.2 Problem Statement

The combination of distributed computing, virtualization, and economics has dramatically changed the computing landscape. Cloud computing has the potential for software systems to change themselves dynamically to match user load while main-

[2]http://www.vmware.com
[3]http://xen.org

taining service level objectives and improving resource availability in a cost effective manner. These improvements relate to increased system utilization and user satisfaction which are two of the primary benefits of cloud computing.

The key research questions addressed in this dissertation are (1) how do we take advantage of distributed computing and virtualization technologies, and (2) how do we design them together to achieve desired user and system objectives. There are many different computing architectures in which distributed computing and virtualization may be constructed. There are many different scheduling, control, and management options available to control how users access their applications hosted on the cloud. This is where the wealth of research opportunities in cloud computing exists.

There are many interesting cloud computing challenges as enumerated below. These challenges drive the research in this dissertation.

1. What are good characterizations of the cloud computing paradigm?
2. Where does scheduling apply within a cloud computing paradigm? How do scheduling techniques at different levels of cloud computing affect system performance, service requirements, and user experience?
3. What are the relative merits of employing control system theory in cloud computing? How does control theory apply?
4. How can we support different types of users and applications in a fair manner? How do high performance computing applications and users differ from social web applications and users?
5. How can autonomic and feedback models be useful in cloud computing? How is this different from control theory when applied on the cloud?
6. Can we provide a taxonomy and characterization of what, when, where, and how to use scheduling with control and autonomic theory to manage cloud systems?

In this dissertation, we focus on characterizing cloud scheduling applications based on control and autonomic theories. We investigate the use of the cutting stock problem as applied to resource provisioning and task allocation, and investigate model-driven control to affect scheduling results. We investigate how control and autonomic

theories can be combined within a feedback framework. This is difficult since cloud architectures are layered. In practice, each layer does not allow other layers access to lower layer states. This presents significant challenges for developing accurate layered models to be used by schedulers and controllers.

1.3 The Approach

In this dissertation we propose several hybrid designs that combine control and scheduling theories within a feedback framework. Concepts from control theory (i.e., proportional-integral-derivative (PID) controllers and Kalman filters), are integrated with approaches from scheduling theory (i.e., scheduling models such as cutting stock). The feedback framework allows schedulers and controllers to work together. For these designs to work well, the systems need to be configured at runtime. This necessitates research into adaptive and self-adaptive control models. Adaptive controls have the ability to adapt to changes in the computing environment (i.e., virtual machine migration, changes in provisioning, or changes in user workload). Self adaptive controls can extend adaptive control solutions by changing the solutions that adaption controllers use. Several adaptive and self-adaptive models are explored including Model Identification and Adaptive Controllers (MIAC) and Model Reference Adaptive Controllers (MRAC) [Rav00].

The approach employs a scheduling system designed to segment available resources to match resource requests. This technique can be used to provision virtual machines with physical resources (i.e., processor, memory, and bandwidth). This approach uses the cloud's ability to reconfigure dynamically as user loads change. If a cloud provider is in danger of violating its SLAs by over provisioning, the cloud can offload selected user requests to a third party (overflow) at additional cost. The objective is to minimize third party usage in order to maximize profit. The cutting stock formulation models this problem and simulation results show improved profits when employed. Another scheduling based approach is to investigate model-driven scheduling. With this approach, the scheduling formulation model is modified so that the scheduling algorithms can provide minimum guarantees for scheduling solutions. This approach uses proven mathematical theories to ensure scheduling results, but it requires the system to have the ability to modify the system workload and system resources. This

is necessary to ensure that the problem formulation has a specific structure. For example, it may be necessary for the formulation to be structured as a matroid to ensure scheduling results are at least half of optimal, where optimal is the absolute best achievable schedule. For example, an optimal schedule for a set of jobs can execute in one hour. Using our method we gaurantee to find a solution that will not exceed two hours.

Another approach uses a control system design based on Kalman filters. The Kalman filter has a predictive property which can be useful when making load balancing decisions. In this approach, wherein we devise and implement a scenario using the OMNET [4] network simulation environment, the filter is used to predict application performance. If the filter predicts that the application will be overloaded, user requests are offloaded to a third party at additional costs. The objective is to minimize the frequency of application service violations, while minimizing costs. This approach proves useful if the application service is approaching violation (i.e., the user request rate is equal to the application service rate).

Another approach involves two case study designs. The first case study is a social tool called Yakkit, designed to take advantage of a distributed set of clouds using our own application overlay called iCon. The application data structure is distributed over a cloud or set of clouds and the overlay facilitates distributed searches and message passing between users [DLM11]. The second case study designed and implemented a cloud testing framework called RIGI Cloud. The design employs approaches described in this dissertation to manage cloud provisioning using a batch scheduling system called Torque.[5]

The approaches discussed are dependent on the cloud architecture used to host user applications. In this dissertation, we use an economic based cloud architectural model to deploy and evaluate our designs. These designs can be used at any layer of the cloud model (e.g., Infrastructure, Platform, and Software). They can be used within the cloud, or used to aggregate several cloud providers as a global pool of resources. However, cloud providers (i.e., Amazon's EC2) do not allow access to their state. This presents significant challenges and necessitates models at runtime.

[4] http://www.omnetpp.org/

[5] http://www.adaptivecomputing.com/products/open- source/torque/

1.4 Dissertation Overview

Chapter 2 describes current research on cloud computing and associated technologies including computing architectures [FK03], a federated cloud compute paradigm [BYV+09] [AFG+10], grid scheduling and economics [Buy02] [Ran07] [Van08], control systems [Oga87] [HDPT04] [Xu07], autonomic models [Mül06] [IBM06] [SILI10], and peer to peer computing. Research is summarized along several dimensions using a taxonomic approach.

Chapter 3 dimensions the cloud research domain and classifies relevant literature using several of the taxonomies developed in Chapter 2. We use the cloud paradigm developed by Buyya et al. [BYV+09] and Fox et al. [AFG+10] within a feedback framework developed by IBM [IBM06] along with a taxonomy on cloud computing to categorize the literature. Finally, this chapter details important aspects of our selected research areas.

Chapter 4 introduces the cutting stock problem and maps cloud resource provisioning problems to the cutting stock problem. Results are presented along with a taxonomy on scheduling within the cloud. The model has been adapted and published as a case study for scheduling to a distributed set of clouds [BDM+11].

Chapter 5 uses a model-driven approach to provide scheduling guarantees. The problem formulation is manipulated so it has specific structural properties. Problem formulations with specific structural properties provide guarantees with respect to solution quality. In this case, we use the greedy algorithm to solve the scheduling problems [BDM+14].

Chapter 6 presents experiments and works with feedback and utility as applied to computing systems. A taxonomy is presented which categorizes feedback with utility, along with a description of the PID controller, autonomic controllers, and Kalman filters. This work includes the simulation of a Kalman filter for service admission control. Service accessibility is a valid control mechanism which is useful in cloud computing and therefore ought to be studied. This work further describes how feed-

back and utility can be applied with scheduling and where it is applicable in the cloud paradigm.

Chapter 7 presents Yakkit, a case study that investigates the use of an inter-cloud overlay network (iCON). The overlay provides support for a distributed data structure and an interface for searching and message passing algorithms. This work investigates how an overlay could potentially make inter-cloud communication ubiquitous. This work was published in the 2011 CASCON Proceedings [DLM11].

Chapter 8 presents RIGI cloud, a test bed for cloud experiments and exploring scheduling/control scenarios. A scheduling and testing framework is presented to boot and execute virtual infrastructure for experiments. This work extended previous work on providing an autonomic grid management system which was published in Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2007) [DM07]. We demonstrated this work at CASCON (2010) as a cloud cluster queue in which we match virtual worker-node resources dynamically to the user workload queued on the head-node.

Chapter 9 summarizes the dissertation and our contribution. We also outline avenues for future research in this domain.

Chapter 2

Related Work

2.1 Introduction

This chapter characterizes and reviews related research using several cloud taxonomies based on closely related computing paradigms, as described by Buyya [BYV+09] and Fox [AFG+10] and IBM's Autonomic Computing Reference Architecture (ACRA) [IBM06]. This chapter describes our view of the cloud paradigm. We explore it using several proposed taxonomies to aid in the characterization of related research.

2.2 A View of the Cloud Computing Paradigm

The cloud computing paradigm, described by Buyya [BYV+09] and Fox [AFG+10], provides a description of cloud computing artifacts and architectural perspectives. This section presents our view of these cloud models within a control and feedback framework.

Our view of the cloud computing paradigm is derived from Buyya's and Fox's rendition as depicted in Figure 2.1 [CRB+11]. Buyya's cloud provider paradigm is based on a brokerage system. In this system, a brokerage is used to buy and sell cloud services for users. Users of the system specify to their brokers which services they want and how much they are willing to pay. It is consistent with the view of cloud computing as described by Fox [AFG+10]. Here, the cloud architecture is partitioned into three layers: the cloud provider layer; the software as a service (SaaS) layer;

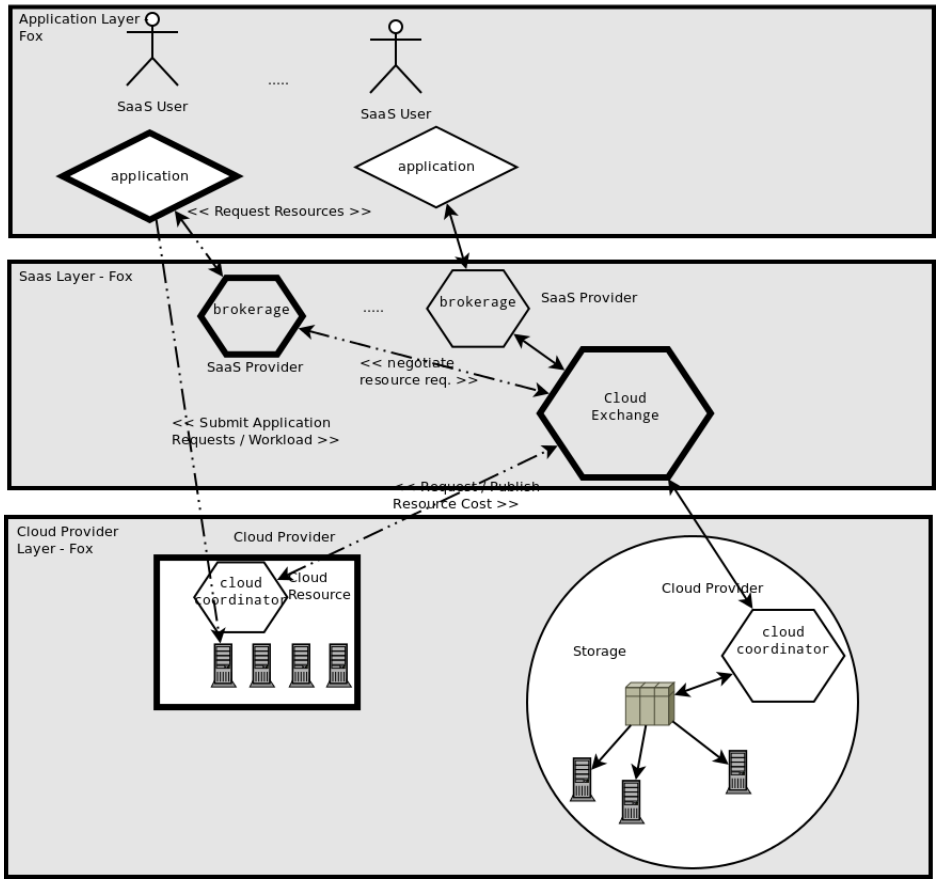

Figure 2.1: an economics based federated cloud compute model comprised of Buyya's [CRB+11] and Fox's [AFG+10] models.

and the application layer. These three layers provide an architectural perspective to Buyya's model. In addition to the three layers, Buyya describes the concept of a cloud exchange which is used by cloud providers and SaaS Providers to publish and bid on resources. This concept is supported by Fox [AFG+10] in its description of cloud computing economics. The cloud exchange facilitates the federation of many cloud providers. This allows SaaS Providers to select cloud provider resources based on cost. This dissertation utilizes the following list of cloud computing artifacts as used in Buyya's economics based cloud model.

- Cloud Provider — The categories of cloud provider described herein are compute and storage clouds. The following is a list of the cloud provider artifacts:

  - Cloud Coordinator — The cloud coordinator executes on each cloud provider. The coordinator is responsible for the following:
    * Export (Publish) — The cloud publishes the services it provides to the cloud exchange. These services include infrastructure services (to boot virtual environments and network infrastructure) and platform services (i.e., to start an Apache web server).
    * Monitoring Resources — The coordinator needs to monitor the physical load being executed to ensure that service level objectives (SLO) are being achieved (i.e., ensuring service utilization does not exceed 75%, or virtual machines are load balanced). In a cloud provisioning system, the required monitoring information may not be available. Having the ability to specify monitoring requirements in a distributed cloud provisioning system is an area where research could be beneficial.
    * Monitor Applications — The coordinator needs to ensure applications are properly load balanced and are achieving their users Quality of Service (QoS) requirements as specified in the SLAs. Scheduling for load balancing is difficult in a cloud provisioning system, due to a lack of both control of localized scheduling and good monitoring. Scheduling for schedulers, or Meta-scheduling, is an interesting research area.
  - Cloud Resource — There are two types of cloud resources: compute and storage. Compute resources refer to CPU utilization and how many Millions of Instructions Per Second (MIPS) the user is allocated for their tasks. Storage resources refer to either disk, tape, or memory in regards to storage of accessible content (i.e., video, audio, text, application objects).

- SaaS Provider — The objective of the SaaS Provider is to ensure its users have access to cloud hosted applications in accordance with the users Quality of Service requirements (which are agreed upon in the form of Service Level Agreements). The SaaS Providers are responsible for ensuring that their clients are getting good value (i.e., cost and QoS) when using their cloud hosted applications. They are also responsible for bidding on resources published by cloud providers. If their bid is successful, then the applications can be dynamically booted and hosted on the cloud provider. Once the applications are started, the brokers can negotiate their users work loads to the best cloud sites, which optimizes cost but still achieves their users QoS requirements.

- SaaS User (Application User) — The SaaS user utilizes their SaaS Provider to negotiate access to a desired application with a set of desired QoS requirements. For example, the user may desire a minimum response time for their requests to a hosted application, or desire access to several processor cores. The user's QoS requirements can be categorized into two types, intrinsic and external. Intrinsic QoS deal with the workload the user desires to be executed. For example, the user may desire a task or job to be executed by a virtual machine with two cores and two gigabytes of RAM. External QoS are those that the user or others can apply to the work load to improve their own objectives. For example, a user objective may be that their tasks be prioritized according to how many cores each task requires. Therefore the user can apply a cost value metric to the multi-core option tasks that is larger than the value they give to a single-core option task. This value is an example of an external QoS metric. Application layer scheduling on the cloud is an interesting research area since they would likely conflict with the resource schedulers.

- Cloud Exchange — The Cloud Exchange manages supply and demand and negotiates deals between SaaS Providers and acts on behalf of and in good faith to their users and cloud providers. The economics of buying and selling services is an interesting research area for cloud computing. This area could benefit from game theory or advanced scheduling algorithms.

Figure 2.1 is an economics based cloud model which, Buyya says, can provide computing power as the 5th utility. In this dissertation, we use Buyya's model within a control and feedback framework as depicted in Figure 2.2. The control and feedback framework is provided by IBM's architectural blueprint for autonomic computing [IBM06]. The blueprint is based on the Autonomic Manager's (AM) Monitor Analyze Plan Execute (MAPE) control flow model and the Autonomic Computing Reference Architecture's (ACRA) management model as depicted in Figure 2.2. The merging of Buyya's economics-based cloud utility within a control and feedback framework provides flexibility to monitor and adapt the artifacts (i.e., the SaaS Provider, cloud provider, or application).

The ACRA model defines five layers of control and management. The layers are defined from top to bottom as follows:

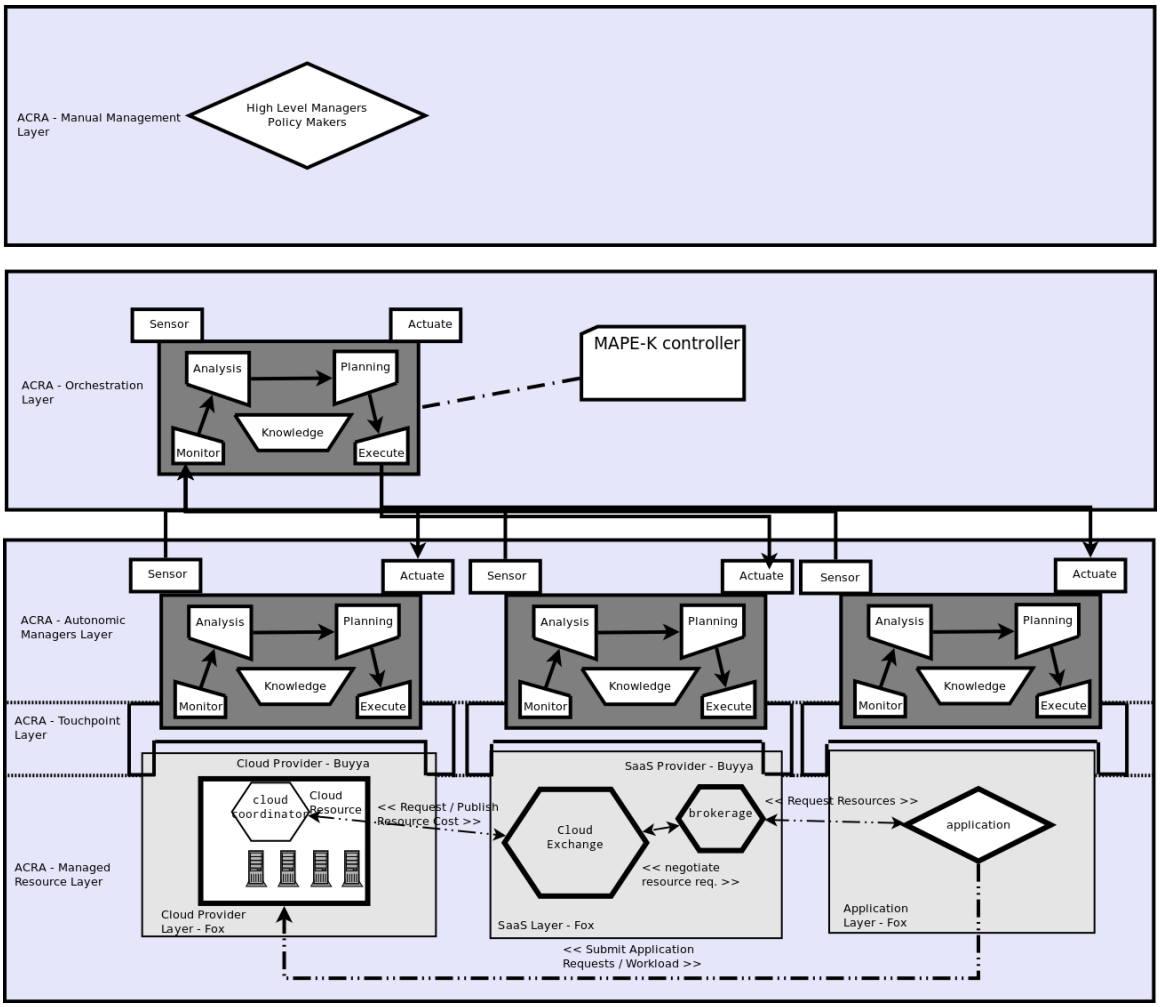

Figure 2.2: Economics Based Cloud Paradigm within a Control and Feedback Framework.

- Manual Manager — is the management interface, specifies which policies should be implemented by the system.
- Orchestration AM — monitors and manages groups of AMs to ensure they work together in an efficient manner (i.e., manage an application's deployment to find cheaper resources).
- Touch Point AM — monitors and manages a resource (i.e., monitor a SaaS Provider to ensure it is not violating its service level agreements).
- Touch Point — control interface to a resource.

- Managed Resource — a controlled resource (i.e., a User's Application or Workload).

Autonomic Managers (AMs) reside at the Orchestration AM and Touch Point AM layers of the ACRA model. Autonomic Managers implement the MAPE control flow loop. The MAPE elements are as follows:

- Monitor — is a collection of sensor inputs used by the AM's analyzer.
- Analyzer — analyzes sensor inputs with knowledge information.
- Planner — uses the analysis results to create an execution plan.
- Executer — actuates the plan created by the Planner.
- Knowledge — used by the Analyzer and Planner to keep state.

Buyya's and Fox's model (cf. Figure 2.1) present several areas where research opportunities exist. For example, the cloud exchange would benefit from research in scheduling to maximize its revenue, or the cloud provider would benefit from admission control theory to ensure its Service Level Agreements (SLAs) are not being violated. In addition to this, by managing Buyya's model within IBM's ACRA model, other research opportunities are presented. For example, scheduling at the orchestration layer could be used to manage an application deployment, or manage a virtual machine deployment for several applications to execute on.

The previous description of Buyya's and Fox's cloud architectures and artifacts within IBM's ACRA enables an assessment of related work of their impact on the cloud computing paradigm as depicted in Figure 2.2. For example, grid computing embraces the concept of meta-scheduling. Meta-scheduling techniques in the grid paradigm may be useful in the cloud paradigm's orchestration layer to schedule applications to cloud providers via Buyya's cloud exchange. The difficulty is where and how easily the grid solutions can be applied and adapted for the cloud paradigm. Section 2.3 presents several taxonomies to classify current literature into our cloud model (cf. Figure 2.2).

2.3 Taxonomies of Cloud Computing

This section creates several taxonomies for our model as depicted in Figure 2.2. The objective is to use the taxonomies to classify current literature for our model. There are seven aspects (concepts and components) depicted in Figure 2.2 of interest aiding in the creation of the taxonomies. We classify cloud computing research into seven aspects of interest.

- The Cloud Aspect — the three layered cloud itself could benefit from a taxonomy of taxonomies that categorizes different perspectives.
- Scheduling Aspect — scheduling refers to resource provisioning and task allocation. Resource provisioning specifies the adding or removing of resources on an entity (i.e., a service, application, or virtual machine) that it consumes. For example, an application may be provisioned to a virtual machine to execute on, or a virtual machine may be provisioned more memory so it can host more applications. Task allocation specifies how user generated work load is distributed to application instances. Scheduling in a federation of clouds is difficult due to the heterogeneity of cloud providers. For example, an issue may be which scheduling topology is required to integrate the cloud providers. Do the schedulers work independently at different levels or communicate to better optimize altruistic objectives? To achieve this, several scheduling levels need to work together. At the lowest layer, the cloud providers provision their virtual machines with physical resources (i.e., cpu, memory, storage, and bandwidth), and specifies which virtual machines will host user applications. The next layer determines which cloud providers are used to host user applications. Finally, the last layer allocates user workload to cloud hosted applications. Referring to Figure 2.2, there are three areas where scheduling can be applied as follows:
  - SaaS Provider — creates SLA contracts between users and cloud providers. The SaaS Provider provisions user applications using cloud resources (possibly from serveral cloud providers) which are negotiated using the cloud exchange. The SaaS Provider may continuously look for better deals in having its user's applications hosted.
  - Cloud Provider — ensures the virtual machines (VMs) have access to physical resources to do actual work.

  - ACRA Orchestration Layer — ensures the SLAs made through the SaaS Providers are being adhered to and that the user task requests are being handled efficiently to optimize the number of entities required to execute the user workload. In the case of peak workloads, it optimizes access to more expensive resources to handle the load.

- Cloud Coordination Aspect — facilitates communication between cloud entities. Questions include how do schedulers at different levels in the cloud paradigm interact with each other? Can they affect and monitor each other?

- Discovery Aspect — ensures resources from cloud providers are continuously updated. Chapter 3 of Ranjan's dissertation describes the resource discovery system in detail [Ran07]. This is not the focus of this dissertation but is of interest to researchers working on discovery mechanisms.

- Autonomic Management and Control Aspect — the application of autonomic theory to cloud computing is new and evolves from biological systems that autonomously adapt to environmental changes. For example if your body has a virus, then your body may increase its temperature to destroy the virus. Control theory is also an important aspect to the cloud computing paradigm. The dynamic nature of cloud computing makes the cloud an attractive platform for software providers, but requires control mechanisms. To provide control, feedback from the system is required to determine system performance. Research in this area has already been done with control system theory [Oga87] [HDPT04]. The difficulty is in how to combine control theory and autonomic management theory with task allocation and resource provisioning.

- Application and Workload Aspect — applications and their workloads affect cloud system performance; for example, they can be processor intensive and consume large amounts of bandwidth and memory.

- Security Aspect — Cloud and grid providers provide a security mechanism to access and use system resources. For example, the CERN grid uses X509 certificates in a single sign-on framework. Security is not addressed in this dissertation.

From the seven aspects of interesting research derived from our model (cf. Figure 2.2), we choose three on which to focus our research. The cloud aspect is selected

because it represents ACRA's framework, and is necessary to manage Buyya's economic based cloud artifacts. The Scheduling Aspect is selected because it provides models, strategies, and solutions to manage the cloud systems. The Application and Workload Aspect is selected because it is affected by cloud organization and scheduling. Therefore, it is interesting for us to investigate effects different workloads have on the applications.

Each of the three aspects described have classifications (taxonomies) which pertain to our model. We propose eleven taxonomies grouped into the three aspects. The taxonomies are used to identify where current literature (i.e., identified techniques from grid scheduling or control theory) may be useful in our model. The taxonomy groups are as follows.

- The Cloud Aspect (cf. Section 2.3.1)
    1. Cloud Taxonomy — is a Taxonomy of Taxonomies
    2. Organizational Taxonomy
    3. Resource Taxonomy
    4. Co-ordination and Discovery Taxonomy
    5. Management Taxonomy
- The Scheduling Aspect (cf. Section 2.3.2)
    1. Management Taxonomy — is a Taxonomy of Taxonomies
    2. Management Organization Taxonomy
    3. Cloud Management-Control Taxonomy
    4. Cloud Management-Scheduling Taxonomy
- The Application and Workload Aspect (cf. Section 2.3.3)
    1. Application Taxonomy
    2. Workload Taxonomy

Sections 2.3.1-3 describe the cloud aspect's taxonomies, the scheduling aspect's management taxonomies, as well as the application and workload aspect's taxonomies, respectively.

2.3.1 The Cloud Aspect's Taxonomies

Our cloud taxonomy is a taxonomy of taxonomies consisting of four sub taxonomies as depicted in Figure 2.3.

1. Cloud Organization Taxonomy (cf. Figure 2.4) — this taxonomy classifies clouds according to structure. For example, is the cloud a grid of clouds or a single cloud, or does it support federation over several data centers?

2. Cloud Resource Taxonomy (cf. Figure 2.5) — this taxonomy consists of several taxonomies. It classifies resources according to organization and resource attributes. For example, the resource may be a physical static resource (like a computer core) which is organized as a federation of similar resources (where the entire system is perceived as one collection of cores and uniformly accessed).

3. Cloud Co-ordination and Discovery Taxonomy (cf. Figure 2.6) — this taxonomy classifies resource discovery entities. How the resources are found and used are useful classifications for characterizing a tool or technique.

4. Cloud Management Taxonomy (cf. Figure 2.7) — classifies management tools, techniques, models, and algorithms in terms of their organization, scheduling, and control capabilities. Each of these have their own sub taxonomies.

The cloud organization taxonomy is depicted in Figure 2.4 as two main classes, the cloud federation class and the data center federation class. The cloud federation classification is concerned with multi-cloud systems. The term 'grid of clouds' can be used to describe this scenario in which different clouds compete for user applications and workloads which are negotiated through SaaS providers. The data federation class is not addressed in this dissertation, but there is an increasing need to support 'Big Data Applications'.

The cloud resource taxonomy consists of two sub-taxonomies that classify resources according to organization and attributes. The cloud resource attribute taxonomy classifies resource attributes using the virtualizable and dynamic classes. The virtualizable classification organizes attributes that are virtualized versus those that are not (i.e., a physical processor is not a virtualized resource). The dynamic class classifies attributes according to their dynamic nature. For example an operating system installed on a machine is not dynamic, as opposed to the amount of RAM

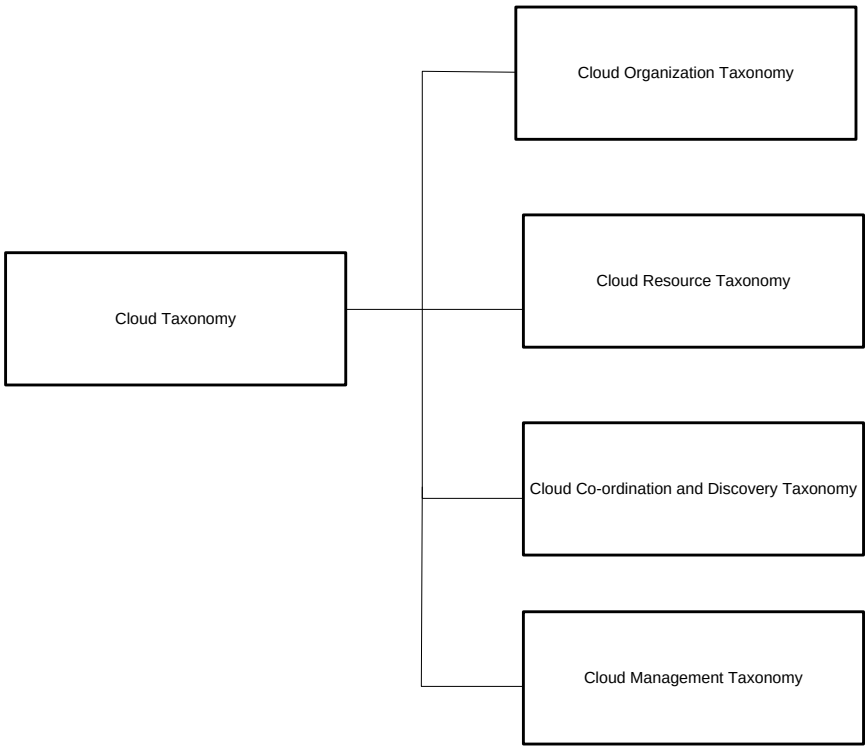

Figure 2.3: A Cloud Taxonomy

allocated to a virtual machine is.

The cloud co-ordination taxonomy classifies how cloud entities find and use each other. For example, cloud entities may register to a central server that has a lookup service, or they may be part of a distributed hash ring in which all participating nodes have a lookup service that works in a decentralized way, or the entities may register locally and be part of a hierarchy of registered entities.

The cloud management taxonomy is one of the foci of this dissertation. Hence we present the last sub-taxonomy of the cloud (a cloud management taxonomy) in its own sub-section(cf. Section 2.3.2).

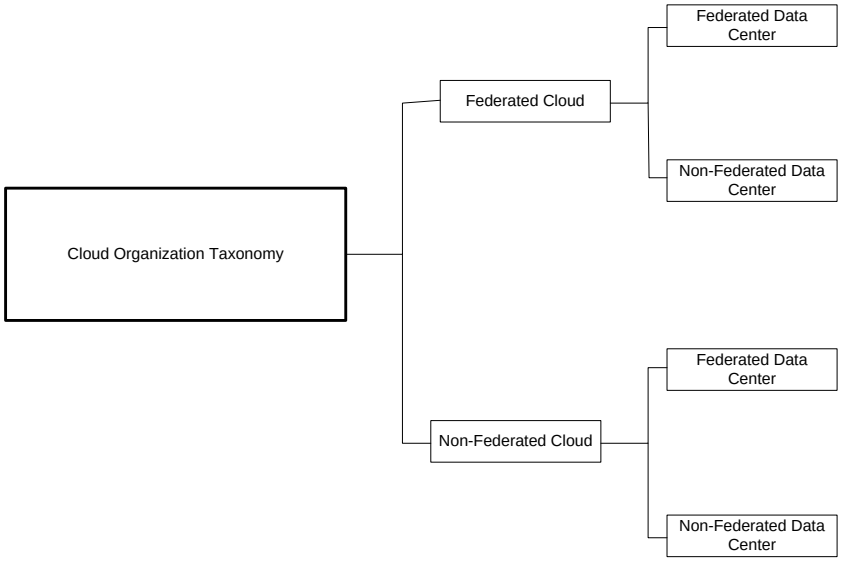

Figure 2.4: A Cloud Organization Taxonomy

2.3.2 A Cloud Management Taxonomy

The cloud management taxonomy, as depicted in Figure 2.7, consists of three sub-taxonomies. The cloud management organization taxonomy, the cloud management-scheduling taxonomy, and the cloud management-control taxonomy as described below.

The cloud management-organization taxonomy, as depicted in Figure 2.8, classifies the management entities according to their organization. In this case they may be organized as centralized, decentralized, or hierarchical categories. For example, the scheduling entity may be organized in a hierarchical fashion in which local resources have a local scheduler and the federation of many local schedulers have a meta-scheduler. In this case the schedulers are organized in a hierarchical fashion. Another possibility is a single centralized scheduling system which manages scheduling at all levels within the cloud system. A decentralized scheduling approach is another pos-

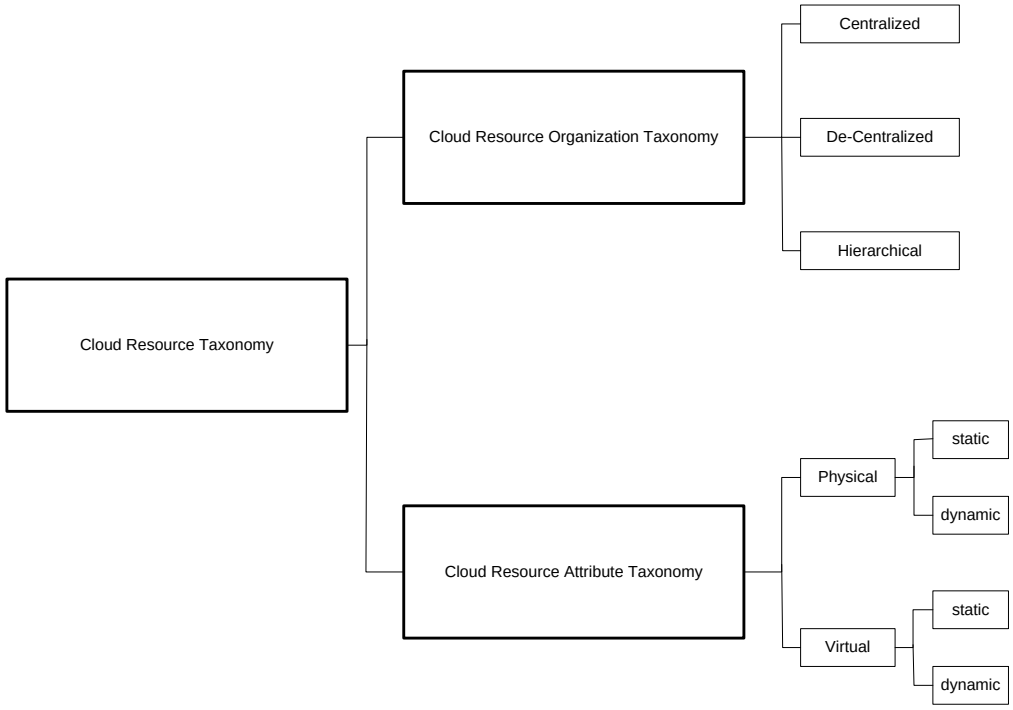

Figure 2.5: A Cloud Resource Taxonomy

sibility in which a unified scheduling algorithm can schedule independently of other schedulers yet they implicitly work together.

The cloud management-control taxonomy, as depicted in Figure 2.9, classifies cloud entities according to their control capabilities and controllability. There are three primary classes: control type, stimulus, and model type. The control type is classified as autonomic, control, and mix methods. Autonomic classification refers to controls and/or management tools which follow the Monitor, Analyze, Plan, and Execute (MAPE-K) model. The control classification categorizes according to traditional control methods such as PID controllers. Mix methods classifies entities which combine traditional and autonomic control methods.

Autonomic controls are classified by stimuli, information analysis, and actions. Stimuli describe how the controller receives information (i.e., from the surrounding environment or internal events). Information analysis may be sensor dependent, or

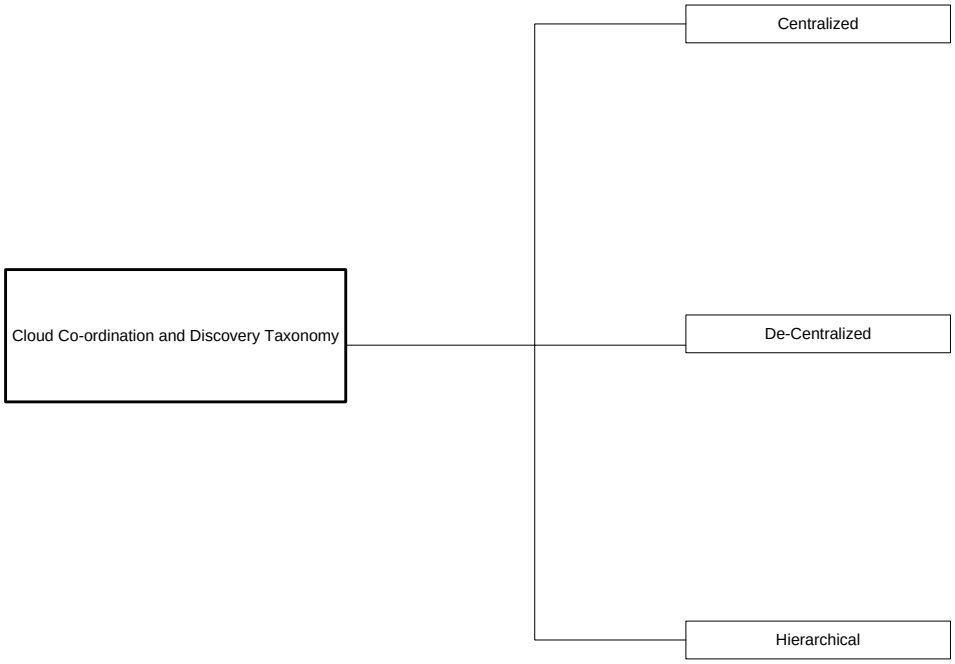

Figure 2.6: A Cloud Coordination Taxonomy

internal logic, or both. Finding a good balance is difficult. Lastly, how does the autonomic system affect itself and its environment. This classifies entities as actively being able to change the environment versus changing itself.

Traditional controls can be classified using first principle models versus more complex models. Another classification is how many inputs and outputs does the controller have. More complex models involve adaptive control and can be classified by following a MIAC (Model Identification) or MRAC (Model Reference) model which identifies how dynamic the control system is. MIAC controllers update the controller's model of the system, and as the model changes, the controllers are tuned to adapt. MRAC models have a predefined model of the system under control and try to fine-tune the control model to adapt to minor disturbances to the system [ÅW11].

Mixed method is a catch-all classification for control entities that do not fit nicely into autonomic or control classifications explicitly. Generally combination models

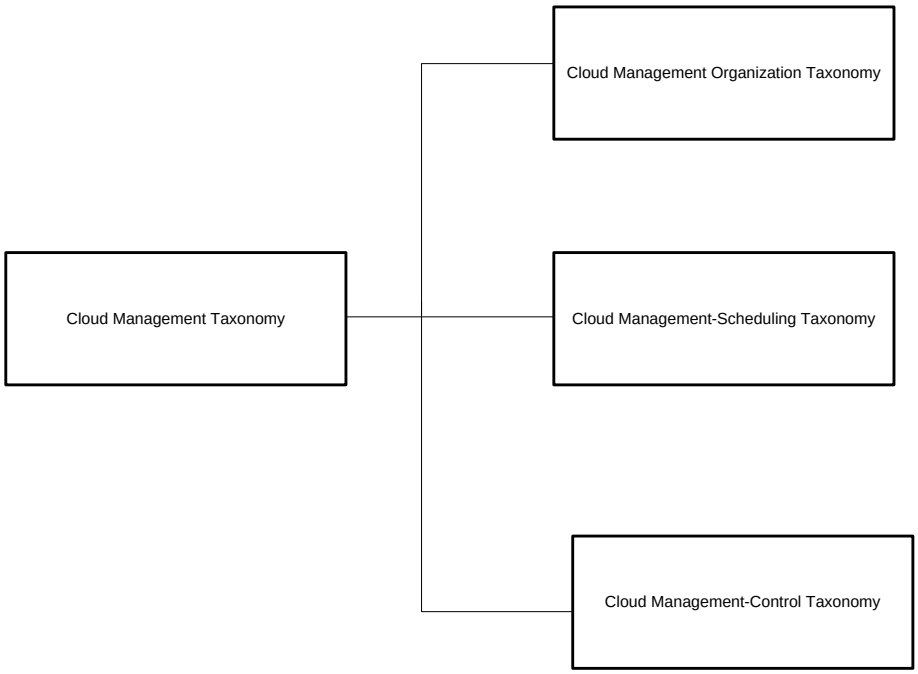

Figure 2.7: A Cloud Management Taxonomy

that use both control and autonomic mechanisms are classified here.

The cloud management scheduling taxonomy, as depicted in Figure 2.10, classifies entities by their organization and objectives. The objective may be classified as a SaaS Provider objective (i.e., find best QoS for lowest cost), or the objective may be classified as a cloud provider objective which manages user work load, provisioning applications, achieving all SLAs, and satisfying internal SLOs.

Cloud management organization is a sub-taxonomy that classifies management entities according to their interoperability. For example, is there a central management system or are there several that all work together in some way?

SaaS provider scheduling can be classified as on-line or off-line scheduling along with classifications for systems that employ feedback and those systems that allow themselves to be configurable.

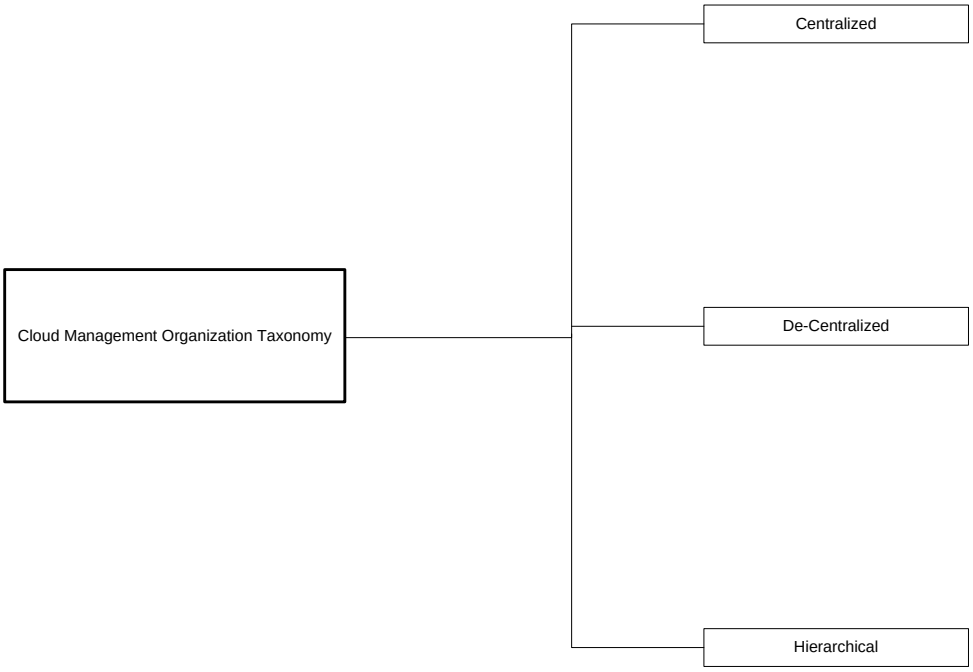

Figure 2.8: A Cloud Management Organization Taxonomy

The cloud provider has a sub-taxonomy for organization which classify according to how entities interact (i.e., coordinated or uncoordinated interaction). The more interesting classifications arise when considering how the cloud provider provisions resources and allocates workload. These two classes can be further divided into on-line or off-line scheduling. On-line scheduling can be further classified according to coordination and use of feedback.

2.3.3 A Cloud Application and Workload Taxonomy

The application taxonomy, as depicted in Figure 2.11, classifies applications as being able to run in parallel or in-sequence. This depends on the dependencies of the workload and/or the application. For example, the application may have a single database that limits the amount of parallel access to the data. This is further classified into

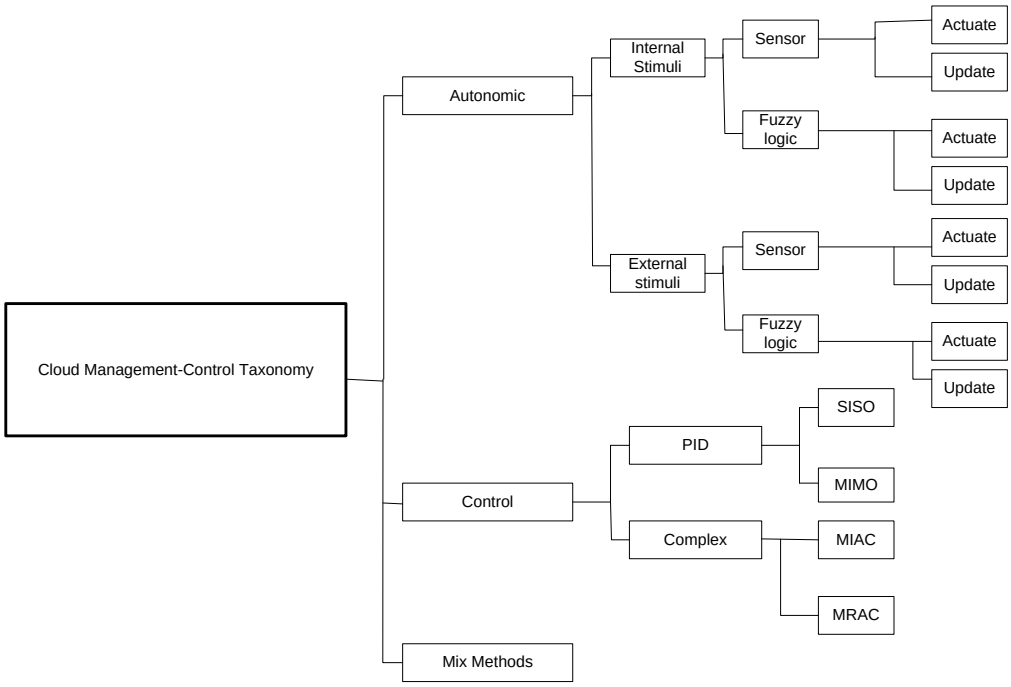

Figure 2.9: A Cloud Management-Control Taxonomy

single or multi-tiered applications. The application is then classified in terms of resource dependencies. Does it require significant bandwidth or is it processor intensive (i.e., a ray-tracer is processor intensive versus a video streaming application which is bandwidth intensive)?

The cloud workload taxonomy, as depicted in Figure 2.12, classifies the workload based on task dependencies of the user's workload (i.e., if task A has to wait for task B to finish). This is further classified into traffic patterns (i.e., burst versus non-burst traffic). Burst traffic is non-uniform traffic in which high volumes of tasks and jobs are submitted followed by idle periods. Non-burst traffic refers to uniform task flow (i.e., user workload arrives at a steady rate). Workload dependencies have a sub-classification of being centralized or decentralized. This means dependent tasks may have to execute on the same machine due to shared memory requirements for processing data. This is the case for OpenMP (multi-platform shared-memory parallel programming) parallel tasks, as opposed to MPI (message passing interface) which

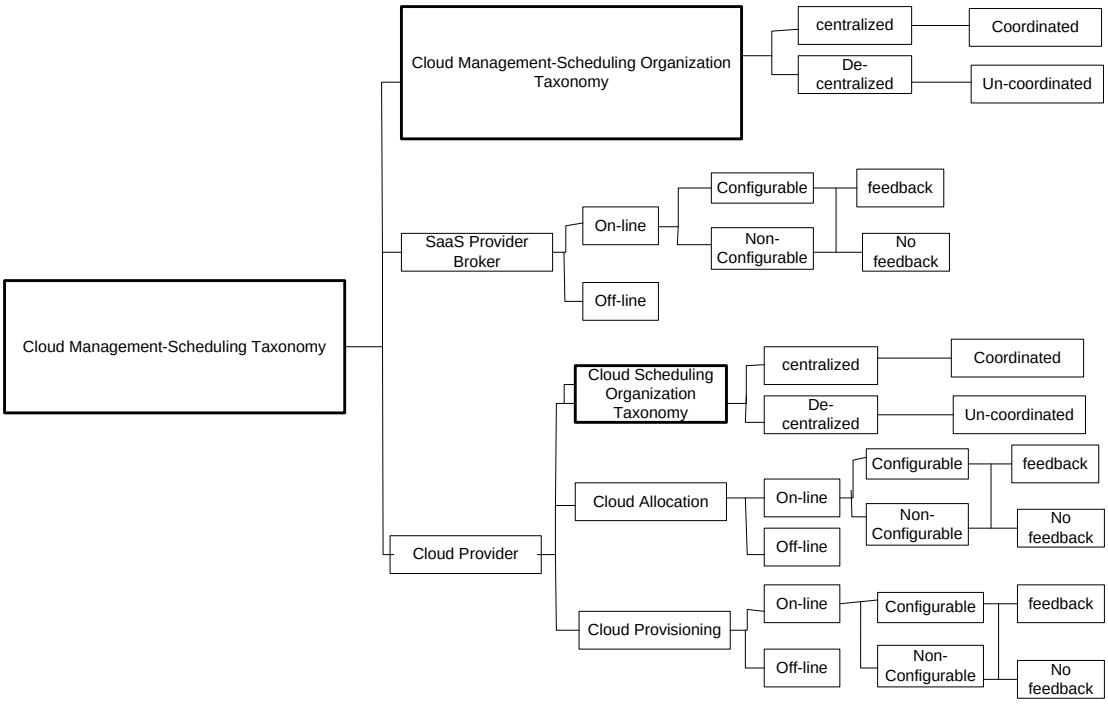

Figure 2.10: A Cloud Management-Scheduling Taxonomy

can be a distributed system where tasks can run on any machine and use message passing to send results to waiting tasks.

2.4 Summary

This chapter provided a broad classification based on eleven proposed taxonomies. In Section 3, literature from cloud computing, grid and cluster computing, peer-to-peer computing, autonomic systems, and management (including control and scheduling) are systematically characterized using selected taxonomies to classify their strengths in potential areas of the cloud computing paradigm as depicted in Figure 2.2.

Dong et al. contributed a comprehensive survey report on grid computing algorithms [DA06]. They use the same approach in providing taxonomies for classification of the grid scheduling domain. Using this methodical approach to research simplifies

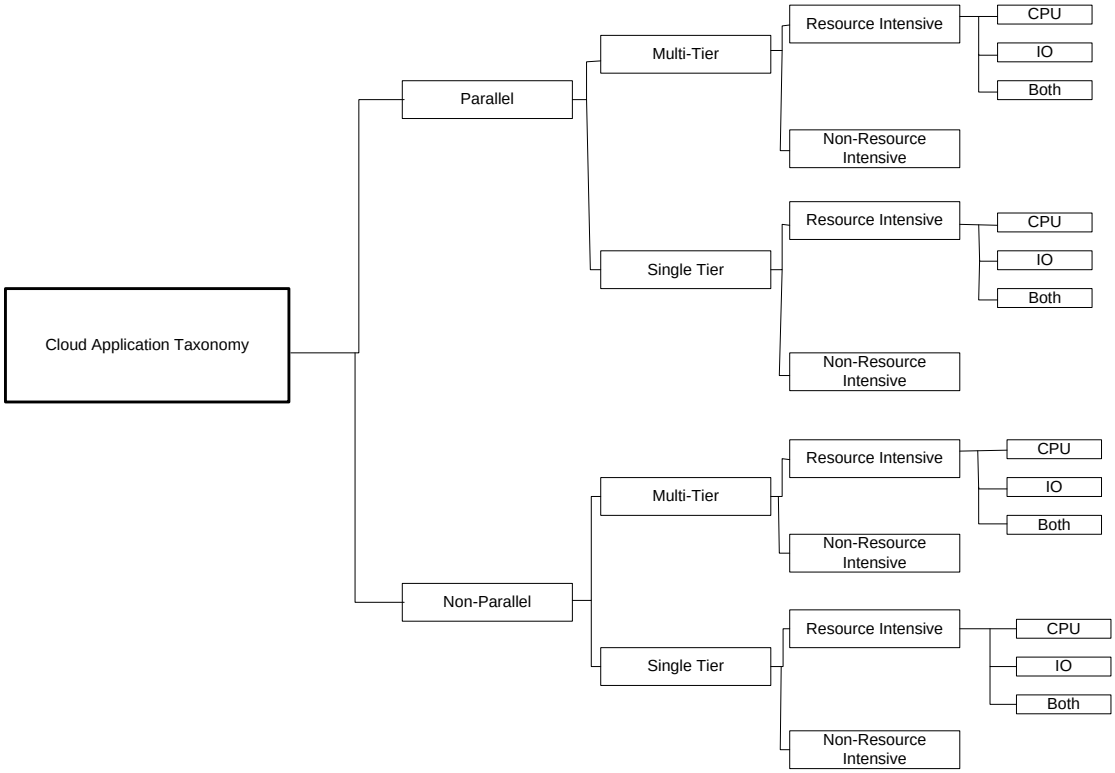

Figure 2.11: A Cloud Application Taxonomy

the literature review and provides insights on how to proceed within a research area by determining how current literature is applicable.

Grozev and Buyya [GB12] provide an architectural taxonomy of the current state of the art. They classify academic and industry projects with respect to inter-cloud research to determine the directions current research is progressing in. One of the areas identified for further research is Service Level Agreement techniques which is addressed in this dissertation.

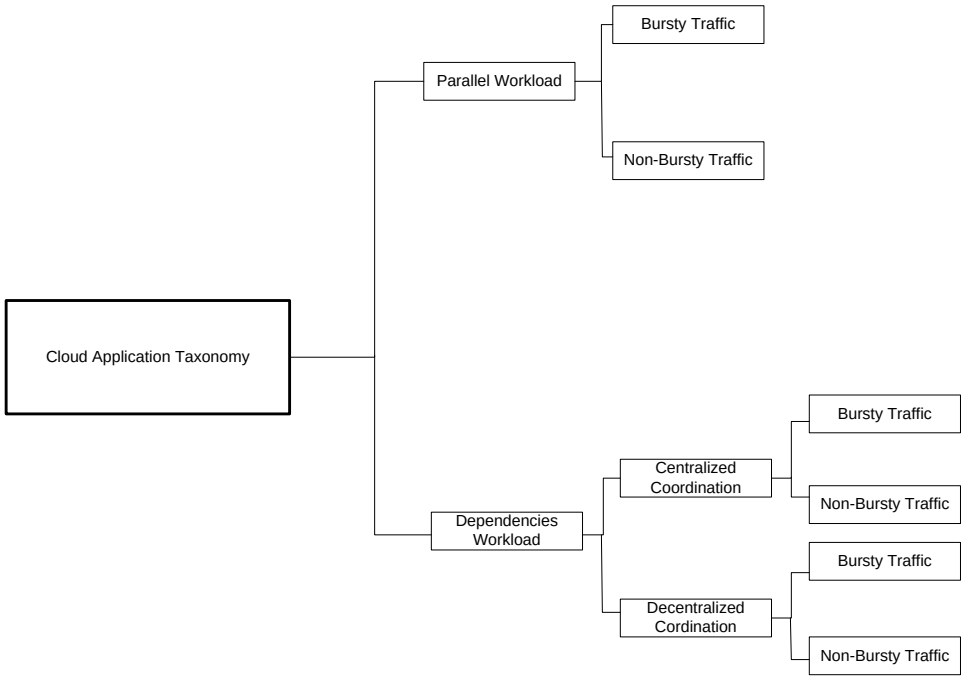

Figure 2.12: A Cloud Workload Taxonomy

Chapter 3

An Adaptive Perspective for Classifying the Literature

This chapter applies two of the taxonomies proposed in Chapter 2 to classify related literature (i.e., cluster computing, grid computing, control system theory and computational clouds) given our view of the cloud computing paradigm (cf. Figure 2.2).

3.1 A Literature Classification

To discuss the related literature, it is prudent to categorize and discuss related works according to the taxonomies developed in Section 2. In particular, we concentrate on the literature related to the cloud research paradigm as depicted in Figure 2.2. We categorize and analyze the literature within this paradigm. Several research categories within the cloud paradigm can take advantage of techniques used in other domains (i.e., the grid domain). Two of these categories, specifically the application of scheduling and control within a feedback framework, will become the focus of this dissertation.

We reviewed many papers, journals, books, and theses for this dissertation and a selected subset is discussed in this chapter. To simplify the discussion and categorization (i.e., using taxonomies from Chapter 2), the literature is grouped into five compute paradigms. Grid and cloud systems, peer to peer systems, autonomic and adaptive and self-adaptive systems, control and feedback systems, and scheduling

systems. This section classifies the literature aspects from each of the five compute paradigms using selected taxonomies. This provides a mapping of aspects from related literature into our proposed cloud paradigm as depicted in Figure 2.2.

The focus of this dissertation is applying scheduling and control strategies in the cloud paradigm, therefore, literature is classified using two relevant taxonomies. The cloud management-scheduling taxonomy and the cloud management-control taxonomy. The literature aspects are recorded in Tables 3.1 and 3.2 and their classifications are depicted in Figures 3.1 and 3.2. For example, Proportional Integral (PI) Admission Control Scheduling is an aspect discussed in Yang [YTX04] (cf. Table 3.1). Using the cloud management taxonomy (cf. Figure 3.1), Yang's aspect is relevant to the cloud broker and can be used for dynamic allocation and provisioning (on-line) when resources are statically defined (non-configurable) in a feedback framework (i.e., PI admission can take advantage of feedback). For some cloud applications this is reasonable and the PI controller could be used for scheduling management.

3.2 The Cloud Classification

Scheduling and control techniques are some of the most pervasive technologies. Since the focus of this dissertation is on scheduling strategies for clouds, it is important that the entire computing field be reviewed for their potential application in our cloud model. Paradigms examined include grid, peer-to-peer, cluster, and autonomic computing. From the literature it is clear some scheduling oriented aspects are common such as:

- Admission Control Scheduling (PI or Kalman)
- Scheduling Algorithms (EDF or FIFO)
- Scheduling Models (Cutting Stock)
- Economics of Scheduling

These generalized aspects are classified into our model using the cloud management-scheduling taxonomy and the cloud management-control taxonomy.

Admission Control and Scheduling appear in literature to be a popular way to avoid over or under utilization of a system. The control policy can be as simple as measuring a server's utilization and assigning tasks to the server if it is below a specified threshold. Buyya [YB06] and Yang [YTX04] are two examples of using admission control for cluster and grid computing, respectively. Buyya focuses on using admission control as a way to manage inaccurate deadline estimates for deadline critical jobs in clusters. They accomplish this by weighting the effect of allowing a job to be scheduled by the cluster. They use a risk metric, a deadline delay metric, and an enhanced decision making process. Yang uses resource-based admission control for grid computing. In this case the resourced-based metric is server utilization. The admission controller is based on a PI controller. The system is modeled (the model's characteristic equation is developed) as a PI and different gain values for K_p and K_I are experimented with to determine their effects under certain load conditions.

In Yang's paper [YTX04], rather than attempting to determine the system model parameters (via traditional PI methods, i.e., a step response), they are assigned using data from previous studies. Through manual testing the controller parameters K_p, K_I and sample rate are set. Figures 3.1 and 3.2 classify these works according to the two taxonomies. The scheduling classification Figure 3.1 supports Yang's [YTX04] use of control theory as a potential solution in the cloud computing paradigm to support the Cloud Broker as an on-line non-configurable feedback controller. Buyya's [YB06] work has the potential to support the cloud provider for on-line allocation and provisioning of workloads and resources. The control classification Figure 3.2 indicates that Yang's [YTX04] technique is classified as a first principles model PID control system with single input and single output, but has potential support for multi-input and multi-output, where Buyya's [YB06] technique is surprisingly categorized as autonomic. This is because the controller does estimation and prediction, which are very difficult to model as a PI control system.

Scheduling algorithms in the literature have been around for many decades and there are many models and solutions. Generalized problems are posed and algorithms are developed to solve them. Concrete problems which can be formed from generalized problems can use prescribed solutions. The difficult part is mapping a real world problem to a generalized problem form. For example, the cutting stock problem is a standard problem which tries to cut finite resources into many different predefined

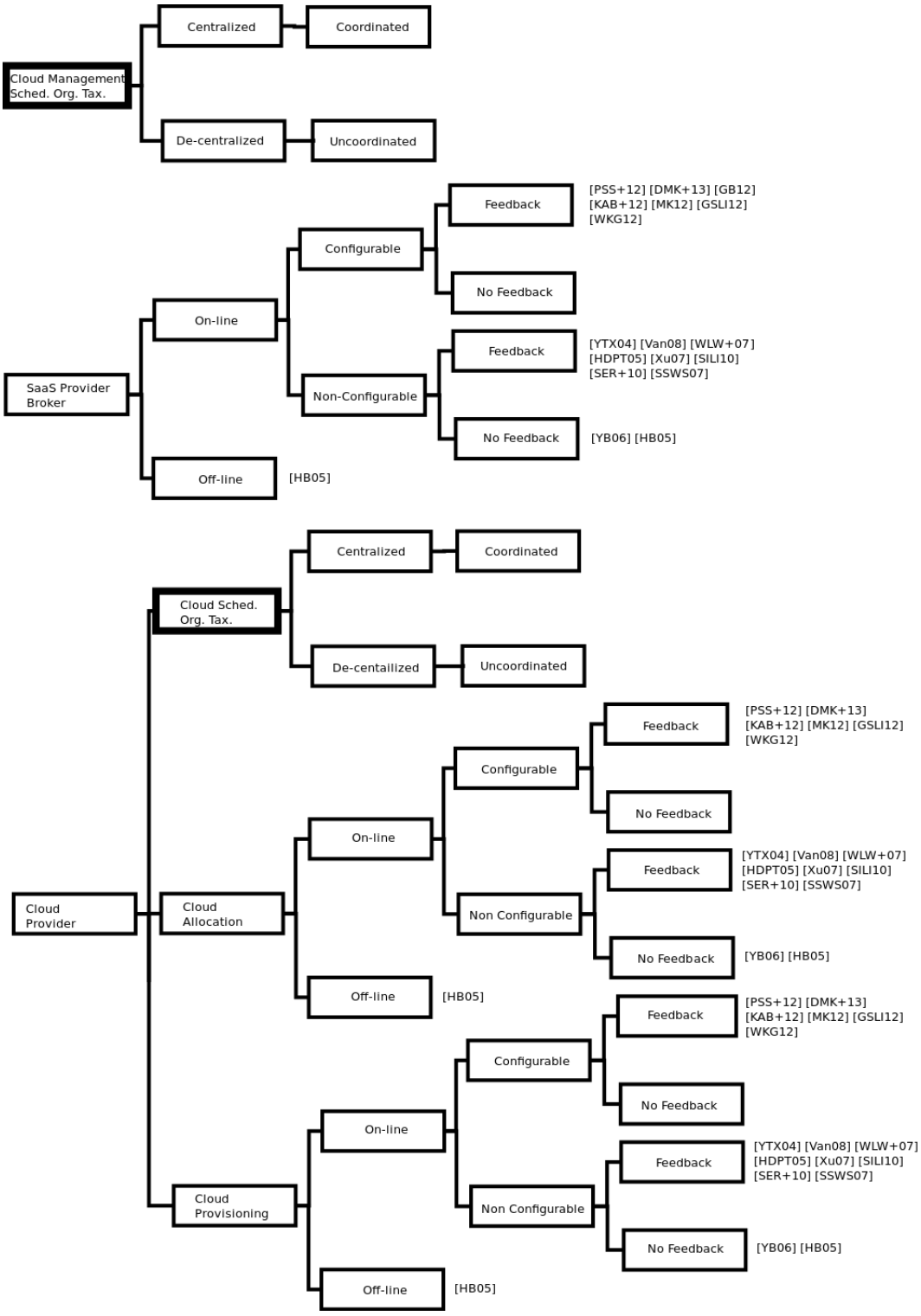

Figure 3.1: Cloud Management-Scheduling Taxonomy — Literature Classification

dimensions. This problem was originally posed for cutting rolls of paper, but has been mapped to solutions for cutting two dimensional sheet metal. In this dissertation, it is mapped as a resource provisioning problem. Other popular scheduling algorithms include Dominant Sequence Clustering (DSC) which is used in minimizing make-spans of Directed Acyclic Graphs (DAGs). Various researchers have embraced these types of techniques [HB05] [ZC05] [CJ01] [ZZ00] [YG94] [PDC08] and [LP98]. Tables 3.1 and 3.2 identify interesting aspects of these papers with potential impact on our model; Figures 3.1 and 3.2 classify the aspects into our model (cf. Figure 2.2). Figure 3.1 indicate most of these models and algorithms are offline scheduling techniques. The scheduling system performs as a state machine with no adaptation, fault tolerance, or assessment capability. These techniques generally have a list of all the tasks to be scheduled, a static model of the resources, and the current state of the resources. This is generally of no use in a dynamic system in which the tasks (workload) are unknown. However, if the scheduling system uses scheduling cycles (i.e., creates a new schedule periodically), then tasks can be queued and an off-line approach can be used to determine a schedule. Figure 3.2 indicates that no DSC algorithms could be classified as control management. This should further motivate research into how cluster methods could be used as controllers, or there are simply no good classification of the clustering methods to be used as controllers.

In Moschakis's paper [MK12], they use Gang scheduling to manage Virtual Machines. They employ an adaptive first come first fit algorithm and a largest job first algorithm to schedule workload to the virtual infrastructure. Wu [WKGB12] provide a comprehensive simulation for admission control based on service level agreements (SLAs). They acknowledge the benefit of research into knowledge based admission control which is an area this dissertation addresses. Tordsson [TMMVL12] provide their own cloud broker design. In this design users provide their own VM template that specify a specific image and target cloud. Nathani [NCS12] uses policy based scheduling to manage virtual machines. They use a lease-based scheduling approach along with other scheduling techniques such as backfilling to fill in scheduling gaps. Litoiu [GSLI12] use a feedback model to update an application's performance model. The scheduler uses the performance model to manage the infrastructure and application deployment. The key contribution of the work is in tracking changes in the models so the scheduler can cause adaptation.

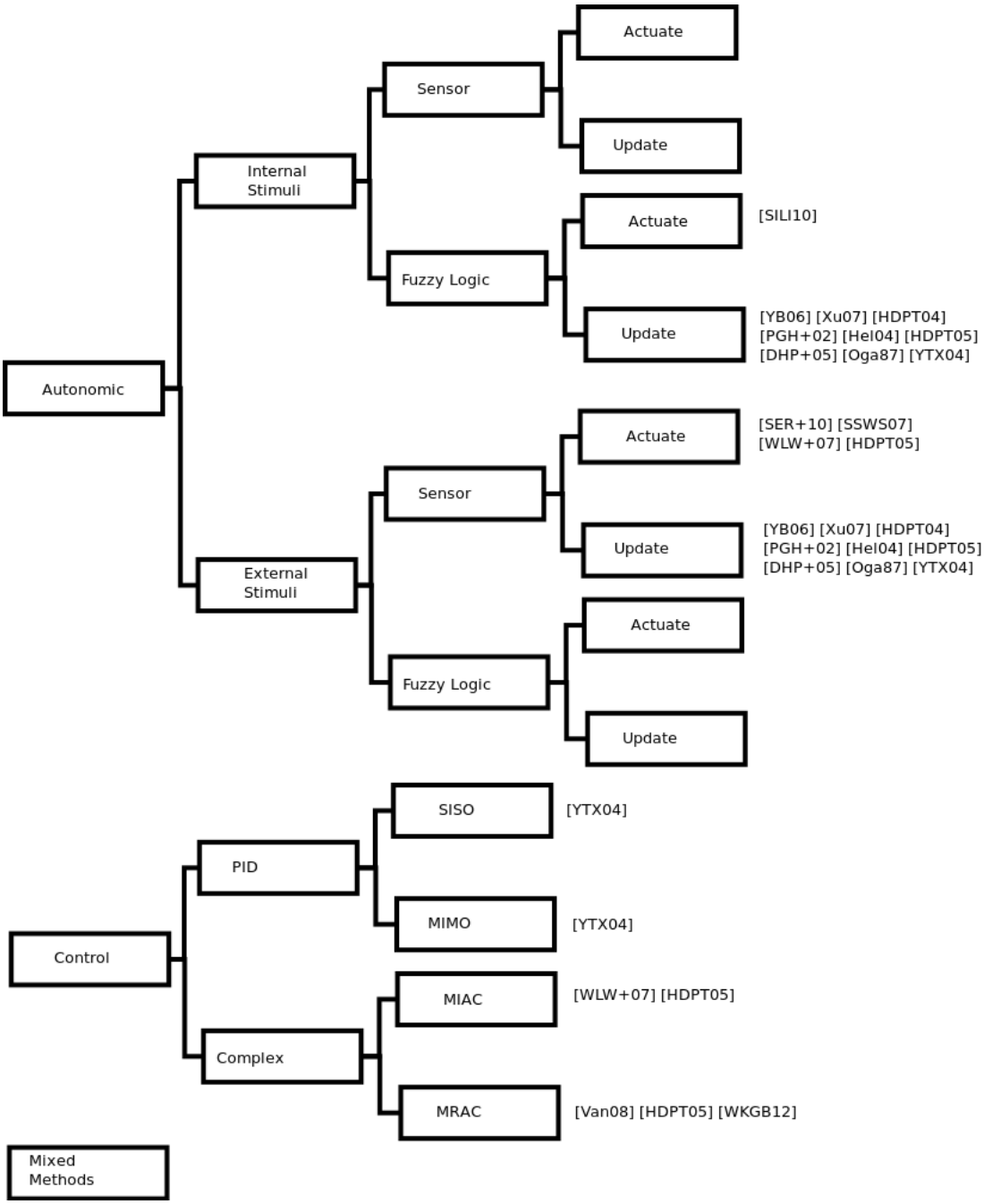

Figure 3.2: Cloud Management-Control Taxonomy — Literature Classification

Another method of scheduling is to use optimization models (e.g., Knapsack or Cutting Stock) which can model scheduling problems using an objective function and a set of resource constraints. These types of models often use heuristics to solve the problem. A solution is a set of tasks assigned to a set of resources. An example of

this is presented in Vanderster's dissertation [Van08]. In this work the problem is presented as a 0-1 multi choice Knapsack problem (0-1 MMKP) and solved using a non-optimal algorithm that employs heuristics. Interesting aspects from these works are in Tables 3.1 and 3.2, in this case the optimization model aspect and optimization solutions aspect. Figures 3.1 and 3.2 classify these aspects into our model (cf. Figure 2.2). This type of scheduling is applicable to both SaaS brokers and cloud providers according to its classification. Figure 3.2 indicates there is potential to apply this theory as an MRAC (Model Reference Adaptive Control), task schedules sent to the cluster have an expected result due to knowledge of the compute performance model (i.e., how many MIPS the cores are rated at). Differences between actual and expected execution times can be used to update the reference model.

Table 3.1: Cloud Management-Scheduling Literature Aspects

Cloud Management-Scheduling Literature Aspects	
Literature's Aspect	Literature Reference
PI Admission Control	[YTX04]
Risk Admission Control	[YB06]
DSC	[HB05]
Optimization Models and Solutions	[Van08] [PSS+12] [DMK+13] [KAB+12] [GB12] [MK12] [NCS12]
Predictive Modeling using Kalman Filter	[SILI10]
External Adaptivity	[SER+10] [SSWS07] [GSLI12]
Application Based Provisioning	[WLW+07] [GSLI12]
Autonomic Feedback Control	[HDPT05] [Xu07] [GSLI12] [WKGB12]

Scheduling management mechanisms can be divided into two categories, autonomic control approaches and traditional control approaches. Traditional control refers to control theory (i.e., engineering control design) and autonomic refers to adaptive and self-adaptive approaches using feedback loops. Several approaches to implement adaptation include predictive modeling, such as a Kalman filters, as presented by Litoiu [SILI10] and others focus on contextual models and adaptation rules based on predicate mathematics as in Sama [SER+10]. Several adaptive schemes have been presented by Yingsong [SSWS07] which provision virtual machines with

physical resources in an adaptive way such that the resources are not over provisioned.

A more difficult but powerful approach to using adaptive mechanisms is the self-adaptive mechanism. This type of mechanism needs the ability to change the actual adaptation process. Some examples of these systems can be found in [AdAM09] [GL09] [MPS08]. Müller [MPS08] focuses on the visibility aspects of adaptive systems which is the first step in creating self-adaptive systems.

Another approach to management is control theory. An excellent masters thesis from UCBerkley by Xu [Xu07] nicely explains the use of traditional control theory to parse system logs to predict and ultimately work through system failures. Moreover, a book by Hellerstein [HDPT04] provides a review of building and constructing controllers for computing systems. Hellerstein is one of the leaders in this area who employs traditional control theory for self-management to satisfy service level agreement requirements [Hel04] [HDPT05] [DHP+05] [PGH+02]. Aspects of these works, such as the Kalman filter, are listed in Tables 3.1 and 3.2, and classified in Figures 3.1 and 3.2. Litoiu's [SILI10] work is classified as predictive control. From a scheduling perspective, this can be classified as any class of feedback. From a control perspective, this can be classified as autonomic. The Kalman filter is a predictive controller, it uses gathered knowledge to make decisions. It uses external stimuli to gain knowledge of the system under control. Therefore, depending on the Kalman filter designed, it can be classified as more internal or more external in terms of its sensitivity (i.e., is the filter sensitive or non-sensitive to quick changes in the environment). Yingsong [SSWS07] is classified as scheduling by supporting SaaS or cloud provider software, and as a controller it can be classified as autonomic, which means it would make the perfect autonomic manager for cloud computing.

Autonomic management can be characterized as systems that aim for a balance within their operating environment, as, for example, our autonomic nervous system. In reality, many systems follow this model whether or not that was their designer's intention. Systems designed with the autonomic approach can be classified as autonomic. The scheduling characteristics of an autonomic system are not entirely clear and depend on the designer. However, there are several works related

Table 3.2: Cloud Management-Control Literature Aspects

Cloud Management-Control Literature Aspects	
Literature's Aspect	Literature Reference
PI Admission Control	[YTX04]
Risk Admission Control	[YB06] [Xu07] [HDPT04] [PGH+02] [Hel04] [HDPT05] [DHP+05] [Oga87] [YTX04]
Optimization Models and Solutions	[Van08] [PSS+12] [DMK+13] [KAB+12]
Predictive Modeling using Kalman Filter	[SILI10]
External Adaptivity	[SER+10] [SSWS07] [GSLI12]
Autonomic Provisioning	[WLW+07]
Autonomic Feedback Control	[HDPT05] [GSLI12]

to this that describe and use autonomic systems. Filino [FS07] describes a tool as autonomic that constructs grid applications according to the current grid environment. Quiroz [QKP+09] uses decentralized on-line clustering to classify grid and cloud work flows. Using this, provisioning based on classification of workload is provided. Wang [WLW+07] proposes autonomic provisioning to support outsourcing the hosting of virtual machines based on an appliance-based provisioning framework. These three works are good examples of using autonomic systems to describe their solutions since they adapt to environmental changes.

IBM provides a blueprint for autonomic computing in the form of white papers [IBM06] describing models (such as the MAPE-K loop), architectures (such as ACRA), frameworks, and components (such as the autonomic manager) to construct autonomic systems. To provide motivation as to why autonomic systems are important, Müller [Mül06] provides a case for its necessity for software evolution and software intensive systems due to their ultra-large scale design and implementation. Aspects of these works, such as Autonomic Control and Risk Admission Control, are listed in Tables 3.1 and 3.2, and classified in Figures 3.1 and 3.2. Wang's [WLW+07] work is classified using cloud scheduling and cloud control taxonomies. It is a good example of the mix of complex control systems with autonomic systems. They use autonomic systems to self-adapt the control mechanisms which are modeled using queuing theory. There is also an optimization problem to solve to reduce costs. Wang's paper [WLW+07] is classified in many categories of Figures 3.1 and 3.2, therefore his techniques are potentially useful in many areas of cloud computing.

Considering economics is essential to scheduling and thus is a rich research field. Economics applied to grid computing has great potential in clouds due to the distributed nature of both paradigms. Moreover the idea of providing computing as a utility is one of the driving forces in this field. Buyya [Buy02] and Vanderster [Van08] both use economic models in their scheduling designs as applied to grids. Buyya [BAGS02] describes the grid economic market place in detail using several economic models. Aspects of these works, such as external adaptivity, are listed in Tables 3.1 and 3.2, and classified in Figures 3.1 and 3.2. Hellerstein [HDPT05] provides comprehensive approaches to integrating autonomic and control theory. Hellerstein has by far the most classifications as being relevant in our Cloud Computing Domain. Pawluk [PSS+12] and Keahey [DMK+13] [KAB+12] both offer solutions to multi-cloud federation using a cloud broker. These solutions use application models and cloud costs to determine deployment of virtual machines and applications. For example, the STRATOS [PSS+12] broker can load balance an application across several clouds with varying costs. The problem is formulated as a multi-criteria optimization problem and solved using several optimization techniques.

3.3 Summary

Literature aspects listed in Tables 3.1 and 3.2 are classified using two taxonomies as depicted in Figures 3.1 and 3.2. Using this classification, we get a sense of where current related literature aspects can be applied to our cloud computing domain as depicted in Figure 2.2. Most literature aspects can be applied to the cloud brokerage and cloud provider. One technique that sticks out from Table 3.1 is the off-line clustering DSC algorithm that has potential for the cloud broker. This could be useful in profiling user traffic flow to group the tasks. This could aid in simplifying scheduling from the brokers perspective. However, it is not very dynamic, and large changes in task workload characteristics could be damaging. A solution to this could be an autonomic control system (cf. Table 3.2) which could be used in addition to the off-line scheduling method to help adapt the scheduling classification when the classification model differs from the actual system model by a significant amount. For example a combination of aspects from Hakem [HB05] and Wang [WLW+07] or Hakem [HB05] and Hellerstein [HDPT05] could be used together to implement a solution for a SaaS

Broker.

Many other combinations from Tables 3.1 and 3.2 can be used to implement the artifacts of the cloud paradigm (cf. Figure 2.2). The remainder of this dissertation investigates and evaluates some of these solutions and their combinations as applied to cloud computing.

Chapter 4

Scheduling Strategies

A key research question derived from questions in Chapter 1 is 'Where does scheduling apply from the perspective of the schedules cost?'. A key area to apply scheduling strategies in terms of cost is within the cloud provider domain. Buyya [BYV+09] states that current cloud technologies need to be extended into cloud provider's infrastructure in order to avoid SLA violations and manage risks. Two years later Buyya [BRC10] re-stated the need for modeling behavior and performance to provide scheduling reasoning for service provisioners. These are some of the motivating factors for research in this domain. This chapter explores scheduling models for virtual machine placement within a cloud provider. This work provides controllability in virtual machine placement, and identifies provisioning metrics.

To find a good approach to virtual machine placement, we refer to the classification works in Chapters 2 and 3. Tables 3.1 and 3.2, Hernandez and Vanderster's work is a good start. Vanderster and others use a 0-1 Multi Choice Knapsack Problem to model grid resource task allocation [PHVD04] [VDPHS06] [VDS07] [Van08] with a utility model to provide quality of service influence.

However not all literature supports the use of optimization models for virtual machine allocation. Flrin [HMGW07] states that the difficulty is due to the unkown state of system resources. In terms of cloud virtual machine provisioning, the difficulty is that the state continuously changes. Vanderster shows that this is true and used a backfilling strategy to compensate the scheduling cycle allocation [Van08]. Vanderster also notices a slight improvement in scheduling performance when compared to a pure backfill strategy.

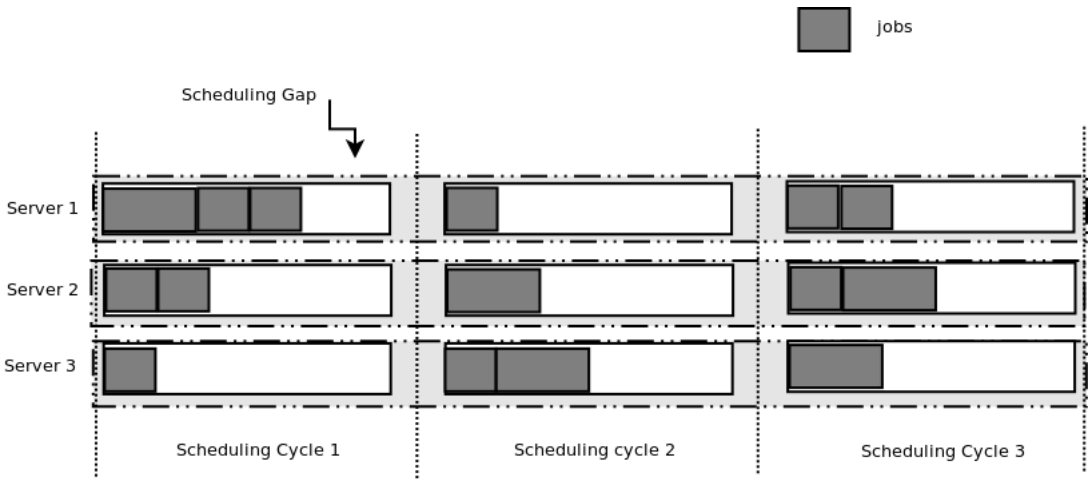

Figure 4.1: Scheduling Gaps

Rather than using a backfill strategy, this chapter proposes a different model to manage the scheduling cycle gaps as depicted in Figure 4.1. The idea is to treat the resource being provisioned to the virtual machine (e.g., physical memory) as a cut of stock and the amount of time that the virtual machine keeps the cut of stock as the amount of stock you wish to order. This problem can be modeled as a *Cutting Stock Problem* (CS), in which the stock replenishes every scheduling cycle.[1] For example, Figure 4.2 depicts a partitioning of two servers. For each scheduling cycle, the servers partition their memory to virtual machine instances. The process of partitioning can be modeled as a CS problem.

The cutting stock problem is closely related to the knapsack problem. The cutting stock problem uses knapsack to solve subproblems. In specifics, the knapsack problem specifies how many possible configurations there are for a given set of tasks and finite resources. This process is repeated for all sets of finite resources from each cloud. The final solution selects a configuration for each cloud such that all tasks are scheduled in one or more scheduling cycles [GG61].

The remainder of this chapter describes the Cutting Stock Problem in detail and explains how it maps virtual machine placements within a cloud. Utility models are

[1] http://en.wikipedia.org/wiki/Cutting stock problem

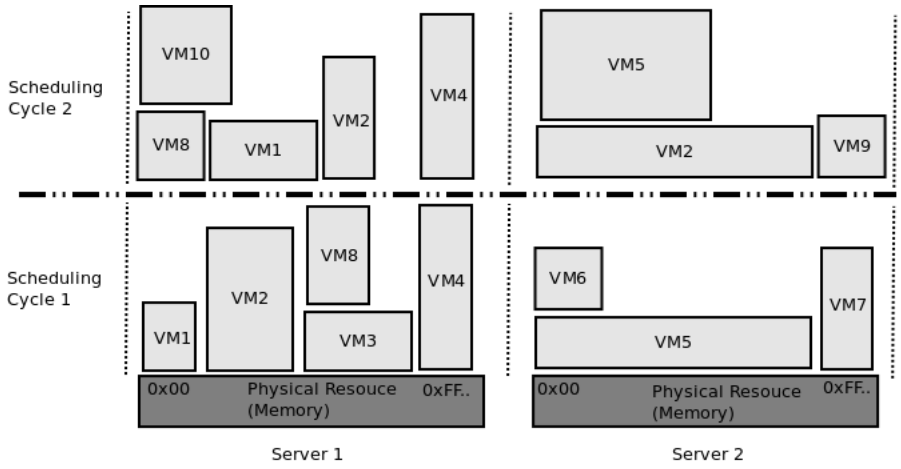

Figure 4.2: Scheduling Gaps

discussed for achieving QoS requirements. We also discuss a scenario using the cutting stock formulation and present experimental results. The chapter ends with some discussion and thoughts on using this type of modeling and its potential for being used as a controller.

4.1 The Cutting Stock Problem

The cutting stock problem can be solved using an integer linear programming solution. It models the problem with an objective function and a list of constraints. These problems suffer from the scalability issue in which the solution space becomes too large to solve by brute force. Therefore much research goes into identifying heuristics (i.e., resulting in potentially non-optimal solutions) which can solve the problem in a reasonable amount of time. The generalized form is presented in Equation 4.1.

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{i=1}^{n} c_i x_i \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{n} a_{ij} x_i \geq q_j, \forall j = 1, ..., m \\ & x_i \geq 0 \end{aligned} \tag{4.1}$$

To illustrate cutting stock formulation we devise a simple scenario. A cheese vendor has several packaged blocks of cheese of 50 cm in length, and three customers requiring different quantities of cheese as itemized in Table 4.1. The vendor wants to

Table 4.1: Customer Order Problem

Customer	Size of Cheese	Quantity of Cheese
1. Bob	20 *cm*	4 pieces
2. John	33 *cm*	3 pieces
3. Kate	6 *cm*	26 pieces

fulfill the orders while opening the fewest blocks.

The vendor sells the cheese according to a policy. In this scenario, the policies are as follows. (1) the vendor fills a customer order from packaged cheese and then recycles any remaining cheese from the order because cheese must be sold fresh from its packaging. (2) the vendor keeps the remaining cheese and tries to use it immediately in the following order. (3) the vendor looks through all three orders at once and tries to determine the best cuts to minimize waste (i.e., the cutting stock policy). The outcomes of this simple scenario are as follows.

It is clear from this scenario that the vendor could benefit from aggregating the orders and a computer program to implement and solve the cutting stock formulation. The cutting stock Equation 4.1 variables in terms of the cheese scenario are as follows:

1. m: is the number of orders. In this case with three customers there are three orders.
2. q_j: is the number of pieces (cuts) for order j. In this case the first order q_1 had 4 cuts, the second order q_2 had 3 cuts, and the third order q_3 had 16 cuts.
3. n: is all possible patterns for the cuts. For example, one possible pattern for a cut is two 20 cm pieces and one 6 cm piece. Another valid pattern is one 6 cm piece (although it is not a great pattern due to the waste, it is still a valid pattern). The number of patterns grows exponentially with the number of orders m. Therefore this type of problem is considered NP-Hard. In this small problem there are more than likely several hundred if not thousands of different patterns.
4. c_i: is the excess waste of pattern i.
5. x_i: is the number of times a pattern is used. In this example, the optimal solution uses two patterns (i.e., pattern number 322 and pattern number 118). The first pattern (20 cm + 5x6 cm) is used four times $x_{322} = 4$ and the second

Table 4.2: Cutting Stock Solution for Customer Orders

Policy	Cuts	Waste (cm)	Blocks
1	1. Order 1 — uses 2 block of 50 cm, waste is 2 blocks of 10 cm 2. Order 2 — uses 3 blocks of 50 cm, waste is 3 blocks of 17 cm 3. order 3 — uses 4 blocks of 50 cm, waste is 3 blocks of 2 cm and 1 block of 38 cm	**115 cm**	9
2	1. Order 1 — uses 2 block of 50 cm, waste is 2 blocks of 10 cm forwarded to next order 2. Order 2 — uses 3 blocks of 50 cm with 2 blocks of 10 cm from previous order, waste is 3 blocks of 17 cm and the 2 blocks of 10 cm which are forwarded to next order 3. order 3 — uses 3 blocks of 50 cm with 3 blocks of 17 cm and 2 blocks of 10 cm which were forwarded from previous two orders, waste is 2 blocks of 4 cm (10 cm-6 cm) and 3 blocks of 5 cm (17 cm-(2 x 6 cm) and a 2 blocks of 2 cm (50 cm - 8*6 cm) and 1 block of 38 cm	**65 cm**	8
3	All Orders at once: 4 blocks of 50 cm, with one 20 cm piece and five 6 cm pieces, no waste. And 3 blocks of 50 cm, with one 33 cm piece and two 6 cm pieces, waste is 5 cm per block.	**15 cm**	7

pattern (33 cm + 2x6 cm) $x_{118} = 3$ is used three times. It is necessary to fill all the orders.

6. a_{ij}: is the number of times that order j appears in a pattern. For example, the first order of four 20 cm blocks of cheese needs to show up in the selected patterns at least four times in order to fulfil the order. Notice that the 20 cm order shows up once in pattern 322 and that the pattern was used four times.

The idea is to iterate through all possible patterns and determine how many times each individual pattern is selected. There are two major issues for the vendor. The first is that the number of patterns grows exponentially. The second is to determine

how many of each pattern to cut. As the problem grows larger, even the fastest computers will not be able to solve it in a reasonable amount of time (e.g., using dynamic programming). Fortunately for the vendor, there are several sub-optimal solutions to the problem. These types of solutions are usually good solutions (although not the best) and are determined in a reasonable amount of time. Advanced algorithms such as cutting-plane, branch and bound, and delayed column generation can all be used to solve these problems.

In Section 4.2 we explore the cutting stock formulation and identify its application for virtual machine placement and memory provisioning.

4.2 Resource Provisioning Modeling using the Cutting Stock Formulation

The virtual machine provisioning problem is quite similar to the cutting stock problem. In this section we model the provisioning problem for a single resource using the cutting stock formulation. To do this, we first describe the virtual machine provisioning problem, and then map the virtual machine provisioning problem to the cutting stock problem as modeled in Equation 4.1. Lastly, we discuss problems, issues, and potential solutions.

4.2.1 Virtual Machine Provisioning Problem

The provisioning problem refers to provisioning of physical resources to a virtual machine (VM). A key issue is that the physical machine's memory is finite and must be divided among competing VMs. Once a VM no longer needs its resource allocation, the resource is released and re-allocated to another VM. Therefore there is a temporal aspect to the problem. Questions the provisioning system must answer are where, when, and for how long should physical resources be provisioned for a VM. In cloud terms, this act of provisioning is referred to as Infrastructure as a Service (IaaS). Figure 4.3 is an example of a user using IaaS to create a new virtual machine and have it provisioned with two gigabytes of RAM. One of the problems this dissertation addresses is how resource re-allocation can be done efficiently from the cloud provider

perspective.

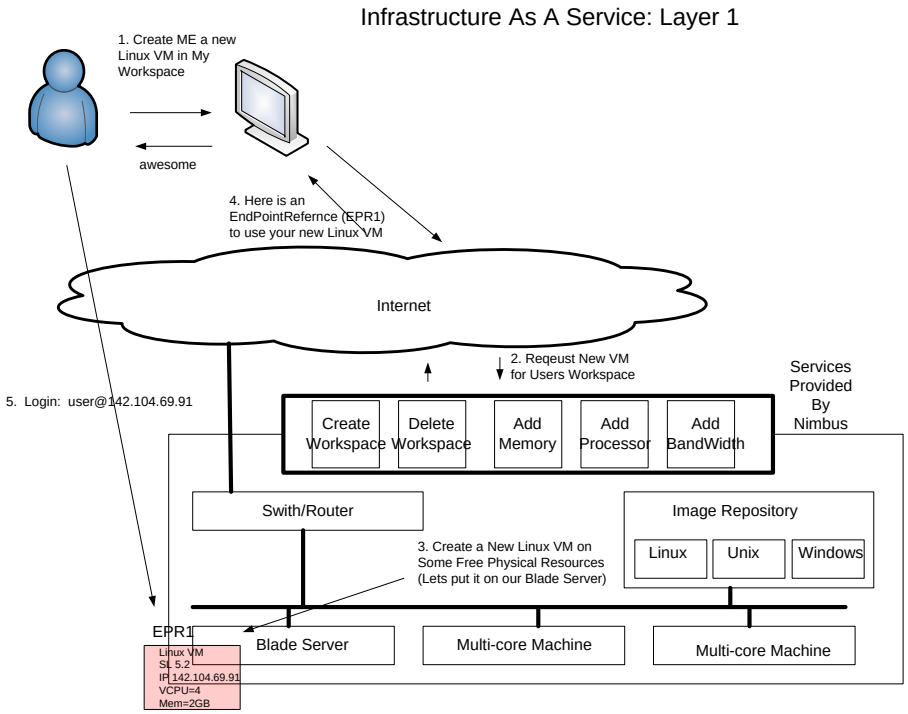

Figure 4.3: Infrastructure as a Service

4.2.2 Provisioning Problem as a Cutting Stock Problem

To formulate the resource provisioning problem as a cutting stock problem we need to map the objective function and the constraints.

- In the cutting stock example, the resource is composed of blocks of cheese (which are 50 cm in length). Their equivalent in this problem is memory (i.e., 16 gigabyte RAM per machine).
- In the cutting stock example, the number of orders are customers. In this case they are requests for a specific type of virtual machine with a given resource requirement. For example, an order may be for a virtual machine with one gigabyte of RAM, while another order is for a virtual machine with two gigabytes of RAM.

- In the cutting stock example, each customer order has a quantity of cheese. For example, the first order requires 4 pieces of 20 cm cheese blocks. The pieces of cheese correspond to time slots. For example, a virtual machine needs one gigabyte of RAM for at least four time slots in order to finish its workload.
- In the cutting stock example, waste is unusable cheese ends. In this case, waste is unused memory.
- In the cutting stock example, a good solution requires fewer packages of cheese. In this case, a good solution should lead to fewer scheduling cycles.

4.2.3 Discussion

There are several issues with this mapping. The first is with over provisioning of resources. If there are more VMs than physical cores, then the virtual machines will have to time or space share the processors. This will increase the amount of time required by the VM to complete its workload. The second issue is when a VM needs to be duplicated or migrated to execute on another physical resource. Replication and migration have overhead, and therefore affect scheduling. The third issue is workload dependencies, for example the workload tasks need to be executed in a specific ordering. The fourth issue is one of utility. Today's clouds have federated data centers running all over the world. Some data centers are cheaper to use than others, and pricing may change during the day dependent on fluctuating power costs (e.g., power is cheaper at night). It is advantageous for a provisioning system to provide some utility to take this into account. This is achieved by allocating a budget (i.e., add monetary significance to the system). A good scheduling system uses its budget effectively.

4.3 Case Study

In this section we introduce a modified cutting stock formulation of the VM provisioning problem. The modifications address issues as discussed in Chapter 4.2.3. Issues of VM migration and supporting a budget for using differently priced data centers are included into the objective function and resource constraint models. The remainder of this section describes the variably priced data centers scenario. Following this we present an adaption to the cutting stock model with a mapping to the scenario's objectives. We present an experiment to evaluate the cutting stock model through

simulation. Finally, we analyze and discuss the results of the simulation.

4.3.1 Data Center Selection Scenario

The data center selection scenario as depicted in Figure 4.4 consists of three different data centers and three different user workloads. In this scenario, the federated data center manager must access the user workload (Orders) and decide how to configure the VM environment to execute the workload. A caveat in this scenario is the addition of a cost function of the data center as a utility. In this case the cost is in power usage over time and in this scenario it is correlated to the resources (i.e., memory) being used at a selected site. The cost of the Vancouver, San Diego, and New York sites are 0 cents per hour, 25 cents per hour and 10 cents per hour, respectively. So it is cheapest to use the Vancouver site. However, as in most systems, there are often bursts of network traffic. During this heavy workload, the users can suffer from resource deprivation. To address this issue, the data center manager is offered a budget to use. With this budget, other sites such as San Diego or New York can be used to access more resources.

Section 4.3.2 describes an adapted cutting stock formulation that includes the ability to apply a budget to the scenario, and addresses the issue of multiple cloud sites. For example, each site has different resources and therefore different patterns (cuts) which it can apply to the workload VM memory requirements.

4.3.2 The Adapted Cutting Stock Formulation

In this section we address the utility of scheduling. Utility in this case is how cost is applied to resource consumption over time. This is similar to how users pay for electricity or natural gas today. The generalized form of the extended cutting stock formulation for the federated data center scenario is described by Equation 4.2 and its cost objective function by Equation 4.3.

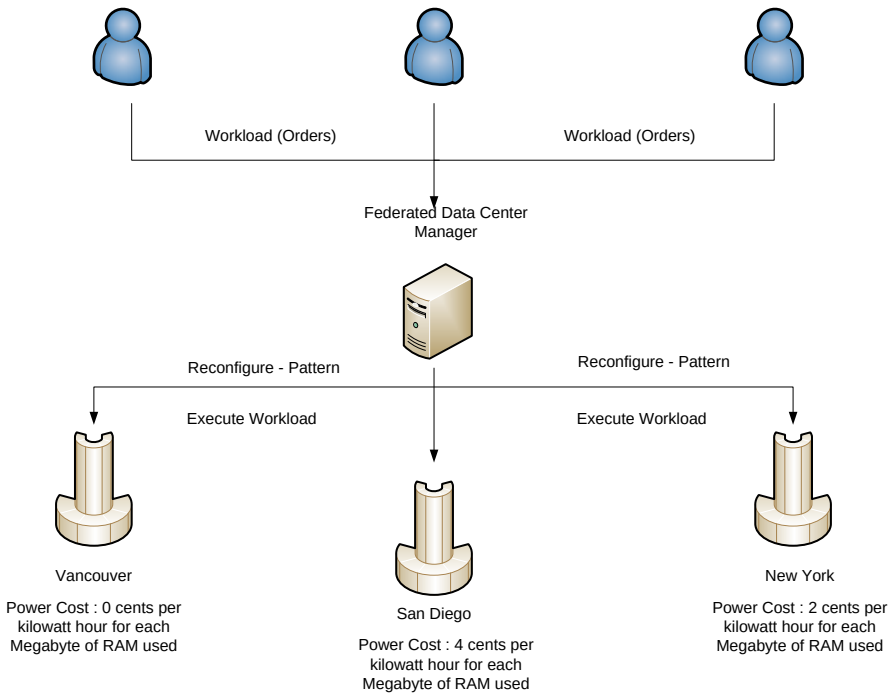

Figure 4.4: Data Center Scenario

$$
\begin{array}{lll}
\text{minimize} & \sum_{k=1}^{m}\sum_{i=1}^{n_k} c_{ki}x_{ki} & \text{minimize cost by choosing the cheapest deployment.} \\
\text{subject to} & \sum_{k=1}^{m}\sum_{i=1}^{n_k} a_{kij}x_{kj} = 1, \forall j = 1, ..., p & \text{for any selection of strategies job j appears once.} \\
 & \sum_{i=1}^{n_k} x_{ki} \leq 1 \forall_k 1, ..., m & \text{for each cloud k, only one deployment is selected.} \\
 & \sum_{k=1}^{m}\sum_{i=1}^{n_k} c_{ki}x_{ki} \leq Budget & \text{cost must be within allotted budget.} \\
 & x_{ki} \in 0, 1 & \text{the selection variable is either 1 or 0.} \\
 & a_{kij} \in 0, 1 & \text{the job selection variable indicates if cloud is scheduled the job.} \\
 & min(Budget) \geq min_k(\sum_{i=1}^{n_k} c_{ki}) & \text{the minimum budget must greater than the cost of executing all jobs on the least expensive cloud.}
\end{array} \tag{4.2}
$$

$$\sum_{j=1}^{p_{ki}} B_k ERT_{jk} \quad \text{cost of cloud k using configuration i is equal to the sum of } p \text{ resource requests in configuration i. Each resource request costs } B_k \text{ and the amount of resource is } ERT_{jk} \tag{4.3}$$

The meaning of the variables of Equations 4.2 and 4.3 is as follows:

- m: is the number of data centers available for configuration.
- k: is the currently selected data center index $k \in 1, ..., m$.
- n_k: is the number of possible deployment patterns (i.e., virtual machine configurations) available on data center k.
- i: is the currently selected pattern index $i \in 1, ..., n_k$.
- c_{ki}: is the cost of using cloud k with virtual machine memory deployment pattern i.
- x_{ki}: is a selection variable (either 0 or 1) and indicates if pattern i is selected for cloud k.
- *Budget*: is a cost provided to the Federated Data Center Manager to use to improve workload performance by buying resources from expensive data centers.
- j: is a resource cut (i.e., 0.5 gigabytes of RAM for a virtual machine) which is effectively the user order.
- a_{kij}: is a selection variable to indicate if resource cut j is in cloud k's i^{th} pattern, a_{kij} is either 0 or 1.
- B_k: is the cost of using cloud k per hour.
- ERT_{jk}: is the amount of resources of order j on cloud k, in this scenario the order is the same regardless of which cloud it is provisioned from.

There are two points to note. First how do we determine a good pattern? This is where the cutting stock formulation from Equation 4.1 can be applied. Feasible patterns created using Equations 4.1, 4.3, and 4.2 can be used to determine a global partitioning of data center resources. It is noteworthy here to state that for small problems, an optimal partitioning of the resource space over time can be achieved. However, the scaling of the problem input causes an exponential rise in solution time and renders it unsolvable in a reasonable amount of time. This is why sub-optimal algorithms to solve these problems are an interesting research area.

In Section 4.3.3, we describe a simulation of the data center provisioning scenario and present and discuss the results obtained.

4.3.3 Data Center Simulation

This section first describes the objective of the simulation. We then describe and discuss the results of the experiment.

The objective of the simulation is to investigate how the cutting stock formulation 4.1 with its extensions in Equations 4.3 and 4.2 compare to a simple scheduling strategy such as the first fit strategy or a mixed strategy algorithm called 'cutting stock mix' that combines first fit and cutting stock. The mixed algorithm randomly selects a subset of jobs that the cutting stock can solve in a reasonable amount of time. The process is repeated until all jobs have been scheduled. We use the three strategies with a single policy objective to achieve a good scheduling allocation of memory (RAM) to virtual machines (i.e., as required by user workload orders) and use the allocated budget wisely in order to minimize the number of scheduling cycles to execute the entire workload. For example, if it costs the data center federation manager an extra hundred dollars to go from five scheduling cycles to three, then it should do it if the budget is sufficient.

We conducted the experiment on the Mercury Cluster at the University of Victoria. We implemented the simulation in Java and graphed the results using Matlab. The simulation consisted of modeling the state of an underlying cloud system and submitting a stochastic workload to measure scheduling allocation results. The experiment consists of three heterogeneous data centers as follows:

- Data Center Setup:
  - Vancouver : 2 machines of 4 cores and 4GB of Ram cost 0 cents per Megabyte
  - San Diego : 1 machine of 8 cores and 16GB of Ram cost 4 cents per Megabyte
  - New York : 2 machines of 4 cores and 4GB of Ram cost 2 cents per Megabyte

We use three different sets of resource cuts (i.e., user orders of required VM memory) as input to the simulator. The sets involve 5 cuts, 10 cuts, and 100 cuts of randomly generated sizes from 128MB to 4GB of RAM. We then applied three algorithms (i.e., cutting stock, first fit, and cutting stock mixed) to calculate the cost and cycle count (averaged over several hundred samples of varying cut sizes). We also assigned varying budgets to the federated data center manager to determine the effect budget had on the allocations.

Figures 4.5, 4.7 and 4.9 depict the results of the 5, 10 and 100 cuts of costs, respectively. Figures 4.6, 4.8 and 4.10 depict results for cycle counts (i.e., how many scheduling cycles are required to complete the orders). Each of these graphs represents insight into how the algorithms perform and compare to each other.

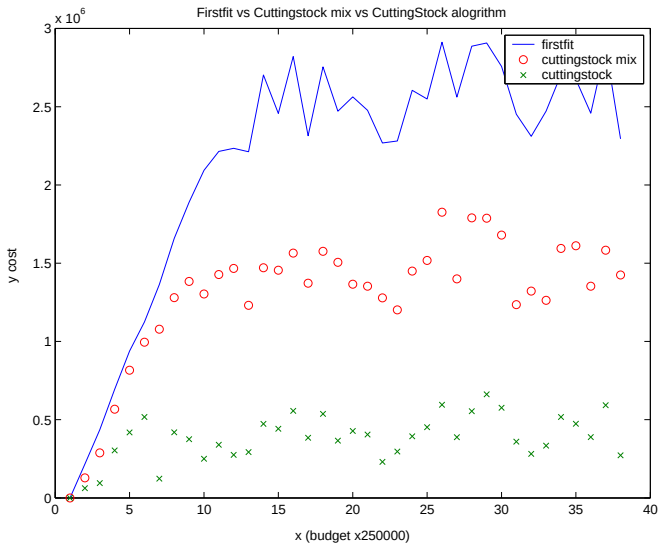

Figure 4.5: Five Resource Cuts Cost

Figure 4.5 represents the average financial cost of scheduling to the data centers using budget sizes from [0-40 x 2500000] credits (i.e., 40 different amounts of budget). Since the number of resource requests is low, the cutting stock algorithm, which is optimally solved (by iterating over all possible solutions), is compared to the mix and first fit algorithm. The use of cutting stock (using dynamic programming) to determine which clouds to use is much cheaper than using a first fit or mix algorithm. For this simulation, the mix algorithm was approximately 40% cheaper and the cut-

ting stock algorithm was approximately 80% cheaper. The mix algorithm provides cheaper solutions than the first fit algorithm. Notice that once the budget exceeds (10 x 250000) credits, the cost settles and does not increase with an increase in budget. The reason for this is that all algorithms have achieved a scheduling cycle count of 1 which is best. So an increase in budget does not result in less scheduled cycles, therefore the cost becomes constant.

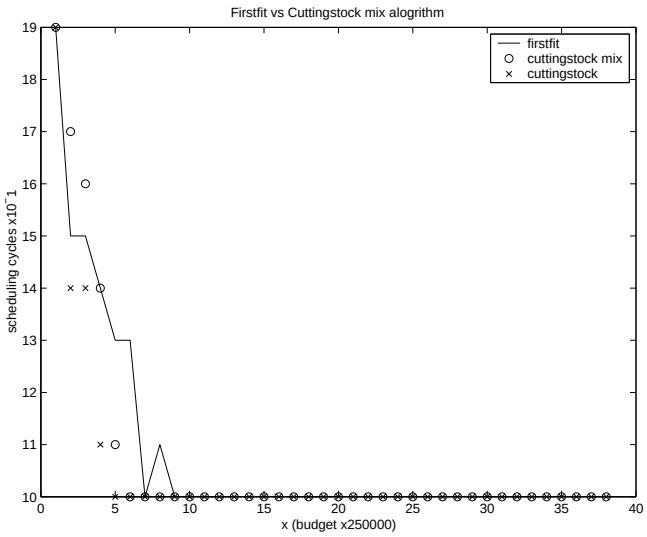

Figure 4.6: Five Resource Cuts Cycle Count

Figure 4.6 compares how many scheduled cycles are required to complete all resource requests. For example, with no budget using cloud system type A, all resource requests must execute on the third cloud which has no cost. Through simulation and a randomized selection of 5 resource request memory requirements, the first fit, cutting stock, and mix algorithms averaged 1.9 cycles to complete. Once a budget is provided, other clouds could be used to host resource requests. The cutting stock algorithm provides the cheapest solution with respect to cost and achieves a single scheduled cycle (which is fastest).

Figure 4.7 compares the first fit and mix algorithm's schedule costs. The mix algorithm does provide an improved cost by approximately 10% on average. The cutting stock algorithm can not be used due to time constraints in solving the problem of size 10. As well, after a budget of size (10 x 250000) both algorithms level off providing

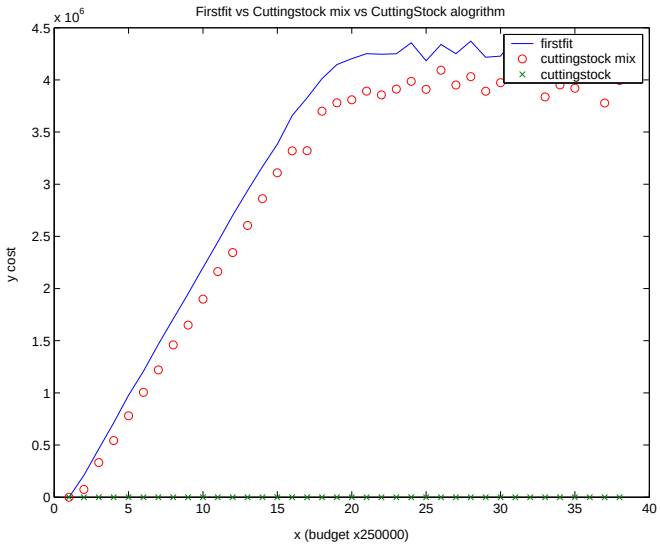

Figure 4.7: Ten Resource Cuts Cost

no more improvement in cost with increase in budget.

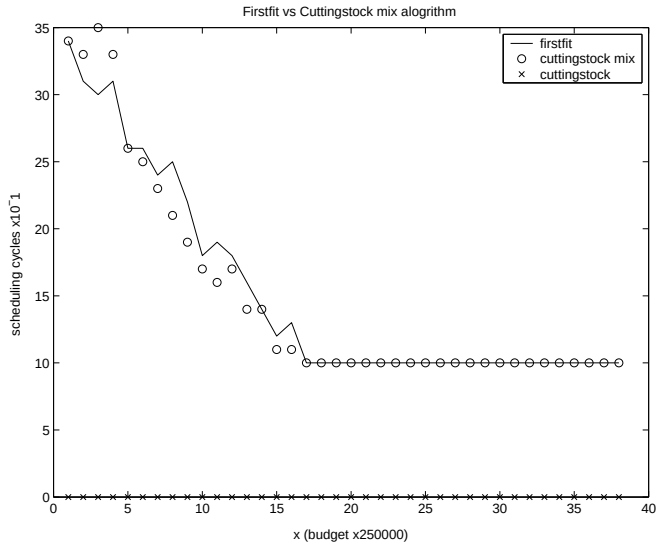

Figure 4.8: Ten Resource Cuts Cycle Count

Figure 4.8 presents how many a scheduling cycles each algorithm produce. Generally there is no benefit in the number of scheduled cycles regardless of which algorithm was used.

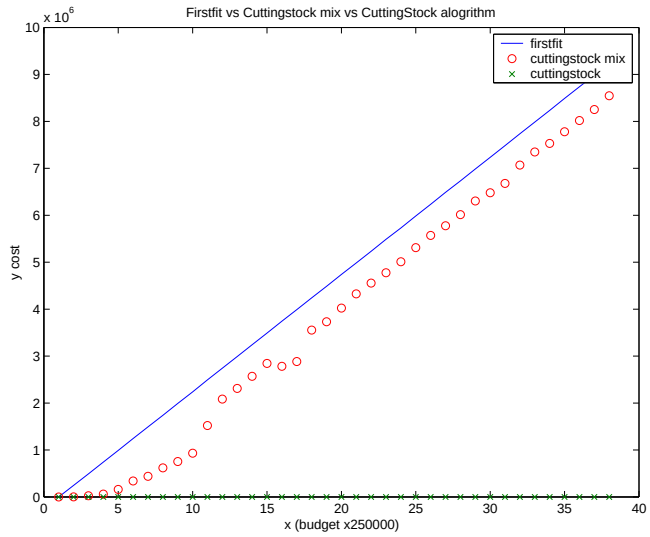

Figure 4.9: One Hundred Resource Cuts Cost

Figure 4.9 represents the cost comparison between using a first fit strategy versus the mix algorithm. In this case the mix algorithm is on average about 15% more efficient. However, for smaller budgets [0-5 x 250000] the mix algorithm did much better than this. Both algorithms are linear after a budget of (13 x 250000).

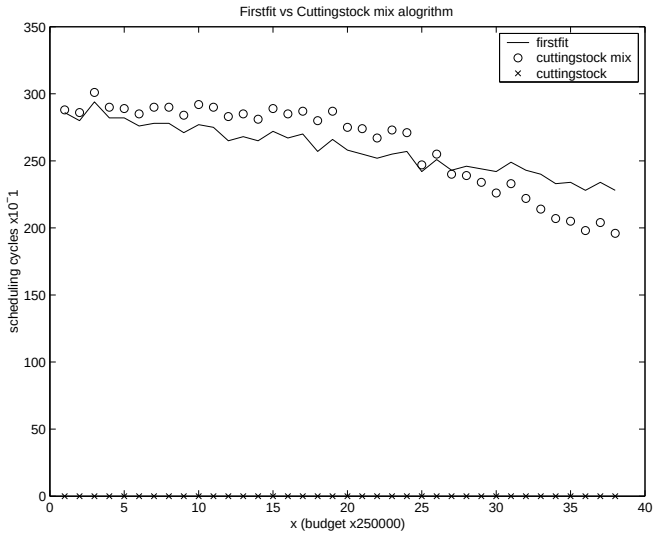

Figure 4.10: One Hundred Resource Cuts Cycle Count

Figure 4.10 represents the number of scheduling cycles both algorithms produce. Generally the first fit and mix algorithms are close in cycle count. However, after a budget of (25 x 250000), the mix algorithm appears to perform better.

It is clear that the cutting stock formulation uses less budget to acheive the same number of scheduling cycles compared to the first fit algorithm. These results are useful when there are finite localized resources that could benefit from a third party cloud provider during peak loads. With these results, developers can prepare for peak loads and minimize third part costs.

4.4 Summary

This chapter described the cutting stock formulation and mapped it to the virtual machine provisioning problem. The problem was extended to model a scenario of several distributed data centers that charge for resource usage. A simulation of the scenario using Equations 4.2 and 4.3 for the cutting stock formulation was compared to a first fit strategy and a mixed methods strategy which used both cutting stock and first fit. It was found for small problem sizes that the extended cutting stock formulation 4.2 could provide an improvement in both cost and number of scheduling cycles required to complete the resource cuts.

Chapter 5

Scheduling Policy: Guarantees and Selection Strategies

A key question derived from Chapter 1 is 'Where does scheduling apply within a cloud computing paradigm from the perspective of maintaining service requirements?'. To answer this question, this chapter explores an alternative view of the scheduling problem. A scheduler provides a temporal mapping of one type to another type. For example, the scheduling of user jobs to compute resources, or the scheduling of physical resources to virtual machines (VMs). The mapping function is the heart of the scheduler. It is described with a set of constraints and an objective function (problem formulation), along with a scheduling algorithm (a solution process). Researchers have found that scheduling algorithms have interesting effects on the quality of the solutions they produce when the problem has certain structural properties [NWF78] [Edm71] [Mes06] [BDM+11] . If the problem formulation and choice of algorithm have certain properties, then theoretical guarantees can be made about the quality of the solution. The quality of the solution is subjective, we defined it as how close a solution is to the optimal solution. Structural properties are applied to the objective function and model constraints of the problem. The two dimensions of the problem formulation form a matrix of combinations. Certain combinations of structural properties are explored in terms of scheduling guarantees when solved by a greedy algorithm.

The scheduling problem defined in this chapter explores solutions which do not always result in an optimal schedule. In many scheduling formulations, the optimal

solution cannot be solved in a reasonable amount of time. The relationship between model constraints and the objective function can be exploited to explore alternative scheduling techniques. Specifically, these techniques are adaptive in nature. Rather than focus on determining the best schedule using a complicated algorithm, they focus on adapting the constraints and objective function to best match the desired policy. For example, the desired policy of a service is to achieve a scheduling revenue of at least $\frac{1}{2}$ of the optimal scheduling revenue, where optimal is the best schedule in terms of its revenue. However, due to current workload and service model, the scheduling algorithm will only produce $\frac{1}{3}$ of the optimal scheduling. The adaptive mechanism will manipulate the workload by removing selected jobs so the objective function and constraints are now changed and will produce a schedule of $\frac{1}{2}$ of the optimal schedule.

Developing intuition on how workload and system models affect the objective and constraints is difficult due to the non-deterministic nature of workloads. In this chapter, we use simulation to understand and gain knowledge on how workload affects the objective function and constraint formulation of the problem. A natural choice is to investigate the effect of the greedy algorithm on four different model formulations that represent the combination of objective function and model constraint structural properties. A greedy algorithm is used because it is easy to implement and provides solutions quickly. Table 5.1 itemizes the scheduling formulations solved by the greedy algorithm that has proven performance guarantees compared to optimal. The vertical axis of the table represents the constraints and the horizontal axis represents the objective function. The constraints of the scheduling model satisfy matroid, k-extendable, or unrestricted properties. The objective function satisfies linear, sub-modular, or unrestricted properties. We implement the following four model formulations as greedy algorithms:

- matroid-linear (ML) — the constraints form a matroid (cf. Definition 1) and the objective function is linear (cf. Definition 4), therefore the greedy algorithm will produce an optimal schedule.
- k-extendable-linear (kEL) — the constraints form a k-extendable system (cf. Definition 7) and objective function is linear, therefore the greedy algorithm will produce at least a $\frac{1}{k}$ of an optimal schedule.

- matroid-sub-modular (MSM) — the constraints from a matroid and the objective function is sub-modular (cf. Definition 6), therefore the greedy algorithm will produce at least a $\frac{1}{k}$ of an optimal schedule.
- k-extendable-sub-modular (kESM) — the constraints from a k-extendable system and the objective function is sub-modular, therefore the greedy algorithm will produce at least a $\frac{1}{k+1}$ of an optimal schedule.

Table 5.1: Scheduling Guarantees based on Problem Formulation using the Greedy Algorithm - Utility Policy: dark gray; Goal Policy: light gray; Action Policy: white [BDM+11]

const. \ obj.	linear	sub-modular	un-restricted
matroid	optimal	$\frac{1}{k}$	no guarantee
k-extendable	$\frac{1}{k}$	$\frac{1}{k+1}$	no guarantee
un-restricted	no guarantee	no guarantee	no guarantee

The linear-matroid formulation is described by Equation 5.1. Problems formulated as such guarantee that the greedy algorithm will produce an optimal schedule. To prove the formulation is linear-matroid, the following must be true. Firstly, the linear objective function (cf. Definition 4) sums the individual sub element weights/revenue of the schedule (i.e., the maximize function $\sum_{i=1}^{m} x_i r_i$). Secondly, the constraints (i.e., the subject to $y_l \in \{0, 1\}$) must satisfy the downward-closure property (cf. Definition 2) and the augmentation property (cf. Definition 3). The downward-closure property states that any subset of a member of a collection is also a member of the collection. The augmentation property states there exists an element $x \in A - B$ such that $A \cup \{x\} \in F$. For example, let $U = \{\{1\}, \{2\}\}$ two jobs, and let $F = \{\{\emptyset\}, \{1\}, \{2\}, \{1, 2\}\}$ all possible schedules of jobs (feasible or infeasible). The example satisfies downward closure since F is all possible combinations and therefore $A, B \in F$. Then if $A \subseteq B$ is true, then downward closure is satisfied by Definition 2. The augmentation property is satisfied (cf. Definition 3) because F represents all possible combinations of A and B, therefore there cannot exist an element x that is only in B such that the union of A and x is not in F. Therefore the augmentation property is satisfied.

$$
\begin{array}{lll}
\text{maximize} & \sum_{i=1}^{m} x_i r_i & \text{maximize revenue} \\
\text{subject to} & y_l \in \{0,1\} & \text{each job interval } (0 \,..\, l) \text{ has at most} \\
 & & \text{1 job assigned} \\
 & x_i \in \{0,1\} & \text{indicates if job } i \text{ in the schedule} \\
 & z_i = 1 & \text{jobs are of size 1}
\end{array} \tag{5.1}
$$

In terms of inputs and outputs, Equation 5.1 is provided a list of jobs to be scheduled (i.e., the input). The constraints determine if the list forms a matroid (i.e, True or False) and the objective function determines the revenue of the schedule (i.e., the output).

Definition 1. Matroid System — Let U be a finite set and $F, F \subseteq 2^U$, be a collection of subsets of U. A set system (U, F) is called a matroid if it satisfies the downward-closure property and the augmentation property.

Definition 2. *F satisfies the downward-closure property: if $A \subseteq B$ and $B \in F$, then $A \in F$*

Definition 3. *F satisfies the augmentation property: if all $A, B \in F$ with $|B| > |A|$, there exists an element $x \in B - A$ such that $A \cup \{x\} \in F$.*

Equation 5.2 describes the linear-k-extendable formulation. Problems formulated as such guarantee that the greedy algorithm will produce a $\frac{1}{k}$ schedule. To prove the formulation is linear-k-extendable, the following must be true. The objective function must be linear as previously described, and the constraints must form a k-extendable system. To form a k-extendable system, it must provide the downward-closure property and the exchange property. The exchange property is supported in Equation 5.2 by the $max(z_i) - min(z_j) = k$ constraint. This limits the number of jobs that can be replaced by a single job. Definition 5 describes the property. It ensures any subset of jobs being replaced has a limit to the number of replaceable elements.

Definition 4. Linear function — For a given set U, a function $W : 2^U \rightarrow \mathbb{R}^+$ is linear if, for any $F \subseteq U, W(F) = \sum_{s \in F} w(s)$ for some fixed underlying weight function $w : U \rightarrow \mathbb{R}^+$.

Definition 5. *Let $A, B \in F$ with $A \subseteq B$, and let $x \in U - B$ be such that $A \cup \{x\} \in F$. Then there exists $Y \subseteq B - A$, $|Y| \leq k$, such that $(B - Y) \cup \{x\} \in F$*

$$
\begin{array}{lll}
\text{maximize} & \sum_{i=1}^{m} x_i r_i & \text{maximize revenue} \\
\text{subject to} & y_l \in \{0,1\} & \text{each job interval } (0 \,..\, l) \text{ has at most} \\
 & & \text{1 job assigned} \\
 & x_i \in \{0,1\} & \text{indicates if job } i \text{ in the schedule} \\
 & max(z_i) - min(z_j) \leq k & \forall i,j \in M; i \neq j \text{ — indicates max} \\
 & & \text{difference between job sizes}
\end{array} \tag{5.2}
$$

In terms of inputs and outputs, Equation 5.2 is provided a list of jobs to be scheduled (i.e., the input). The constraints determine if the list conforms to a k-extendable system (i.e, True or False) and the objective function determines the revenue of the schedule (i.e., the output).

Equation 5.3 describes the submodular-matroid formulation. Problems formulated as such guarantee that the greedy algorithm will produce a $\frac{1}{k}$ schedule. To prove the formulation is submodular-matroid, the following must be true. The objective function must be submodular (cf. Definition 6), and the constraints must form a matroid system as previously described. The problem formulation is submodular if it has the property of diminishing returns. This is supported in Equation 5.3's maximize function (i.e., $[max(\sum_{i=1}^{m}(x_i r_i - x_i r_{dim_i}), 0)]$). This objective function diminishes the returns of a schedule due to other conflicting jobs. The r_{dim_i} variable is the reason this formulation has the property of diminishing returns. It represents the total subtracted from its own revenue (i.e., the diminishing part). From Definition 6, the property $g(A \cup B) + g(A \cap B) \leq g(A) + g(B)$ for all $A, B \subseteq U$ is always true. This is because $g(A)$, $g(B)$, $g(A \cup B)$, and $g(A \cap B)$ all have positive revenue $x_i r_i$ and a negative revenue $x_i r_{dim_i}$. It turns out the LHS always equals the RHS in terms of positive revenue. However, the LHS always has more negative revenue than the RHS. Therefore, the property of diminishing returns is always satisfied. The reason the LHS always has more negative terms is because for every element $x \in Schedule(S)$, it has a diminishing term for all other elements. Therefore the more elements in g equals more negative terms. Since $|g(A \cup B) + g(A \cap B)|$ always has more elements than $g(A) + g(B)$, then it will always have more negative terms and therefore be smaller than the RHS.

Definition 6. Submodular function — For a given set U, function $g : 2^U \rightarrow \mathbb{R}^+$ is called submodular if g satisfies the following property: $g(A \cup B) + g(A \cap B) \leq g(A) + g(B)$ for all $A, B \subseteq U$.

maximize	$[max(\sum_{i=1}^{m}(x_i r_i - x_i r_{dim_i}), 0)]$	maximize revenue
subject to	$y_l \in \{0, 1\}$	each job interval $(0 \,..\, l)$ has at most 1 job assigned
	$x_i \in \{0, 1\}$	indicates if job i in the schedule
	$z_i = 1$	jobs are of size 1
	$r_{dim_i} = \sum_{j=1}^{m} x_j r_{dim_{ij}}$	a diminishing variable depend on other jobs in the schedule

(5.3)

In terms of inputs and outputs, Equation 5.3 is provided a list of jobs to be scheduled (i.e., the input). The constraints determine if the list forms a matroid (i.e, True or False) and the objective function determines the diminished revenue of the schedule (i.e., the output).

Equation 5.4 describes the submodular-k-extendable formulation. Problems formulated as such guarantee that the greedy algorithm will produce a $\frac{1}{k+1}$ schedule. To prove the formulation is submodular-k-extendable, the following must be true. The objective function must be submodular as previously described, and the constraints must form a k-extendable system as previously described.

Definition 7. k-Extendable System — Let U be a finite set and F, $F \subseteq 2^U$, be a collection of subsets of U. Set system (U, F) is called k-extendable system if it satisfies the property of downward closure and the property of exchange.

maximize	$[max(\sum_{i=1}^{m}(x_i r_i - x_i r_{dim_i}), 0)]$	maximize revenue
subject to	$y_l \in \{0, 1\}$	each job interval (0 .. l) has at most 1 job assigned
	$x_i \in \{0, 1\}$	indicates if job i in the schedule
	$max(z_i) - min(z_j) \leq k$	$\forall i, j \in M; i \neq j$ — indicates max difference between job sizes
	$r_{dim_i} = \sum_{j=1}^{m} x_j r_{dim_{ij}}$	a diminishing variable depending on other jobs in the schedule

(5.4)

In terms of inputs and outputs, Equation 5.4 is provided a list of jobs to be scheduled (i.e., the input). The constraints determine if the list conforms to a k-extendable (i.e, True or False) and the objective function determines the diminished revenue of the schedule (i.e., the output).

The remainder of this chapter is divided into two sub-sections. Section 5.1 describes a scheduling scenario for the scheduling formulations described above. Simulation is used to show the worst case schedules in terms of scheduling revenue (cf. Section 5.1.1) and how often worst case schedules actually occur by generating random workloads for a service queue (cf. Section 5.1.2). The simulation is also used to articulate the relationship between the random workload and the service model to provide knowledge for decision making. Section 5.2 of this chapter designs two control mechanisms to take advantage of the knowledge gained from simulation.

5.1 Scheduling Simulation for Model based Control

The objective of this section is to gain knowledge for model-driven control (i.e., the manipulation of work load in terms of job overlap, job length and job dependencies). To ascertain this, we devise two scheduling scenarios to simulate how the greedy algorithm performs in terms of revenue compared to the optimal solution (i.e., the schedule that produces the best revenue). Table 5.1 depicts the greedy algorithm's guaranteed result dependent on the problem formulation. We design the schedul-

ing scenario and provide four formulations: i) linear-matroid; ii) linear-k-extendable; iii) sub-modular-matroid; iv) sub-modular-k-extendable. Each formulation requires a slightly different work-flow and/or service model.

The first scheduling scenario is designed to show how changes in the work flow cause the greedy algorithm to produce schedules that approach the guaranteed minimum revenue. To show this requires a scenario with purposefully placed job conflicts and job ordering. The second scheduling scenario is designed to show a more realistic scheduling system using probabilistic arrival patterns and using a batch scheduling service simulation. A realistic scheduling simulation provides insight into the actual occurrence of the worst case and provides averages of how close to the optimal schedule the greedy algorithm performs.

The next two sub-sections discuss the two scheduling scenarios. To re-iterate, the first scenario is contrived to show how changes in the work flow cause greedy to approach but never violate their guarantees (i.e., as proven mathematically). The second scenario simulates a simple queuing service with a randomly generated workload to gain knowledge on what averages are actually achievable by the greedy algorithm. It also determines relationships between the workload and server model (problem formulation), the distributed arrival rate, and the servers scheduling cycle.

5.1.1 Scheduling Guarantees of the Greedy Algorithm

This section examines scheduling guarantees when using the greedy algorithm. This work is based on results from Nemhauser et al. [NWF78], Edmonds et al. [Edm71], Mestre et al. [Mes06] and Balasubramanian et al. [BDM$^+$11], which prove a problem formulation with certain properties can produce optimal or near optimal schedules using the greedy algorithm. To highlight these properties, we contrive a scheduling scenario and work-flow model. Figure 5.1 depicts four job work-flow models for a simple single queue service (cf. Figure 5.5). The problem formulations are as follows:

- ML — jobs have single unit processing time, and the scheduling returns in terms of revenue is linear.
- MSM — jobs have single unit processing time, but the scheduling return in terms of revenue may be diminishing (i.e., job conflict).

- 2EL — jobs do not have equal processing time, k-extendability in this case is 2 since the longest processing job is twice as large as its smallest. The scheduling returns in terms of revenue is linear.
- 2ESM — jobs do not have equal processing time, k-extendability in this case is 2, but the scheduling return in terms of revenue may be diminishing.

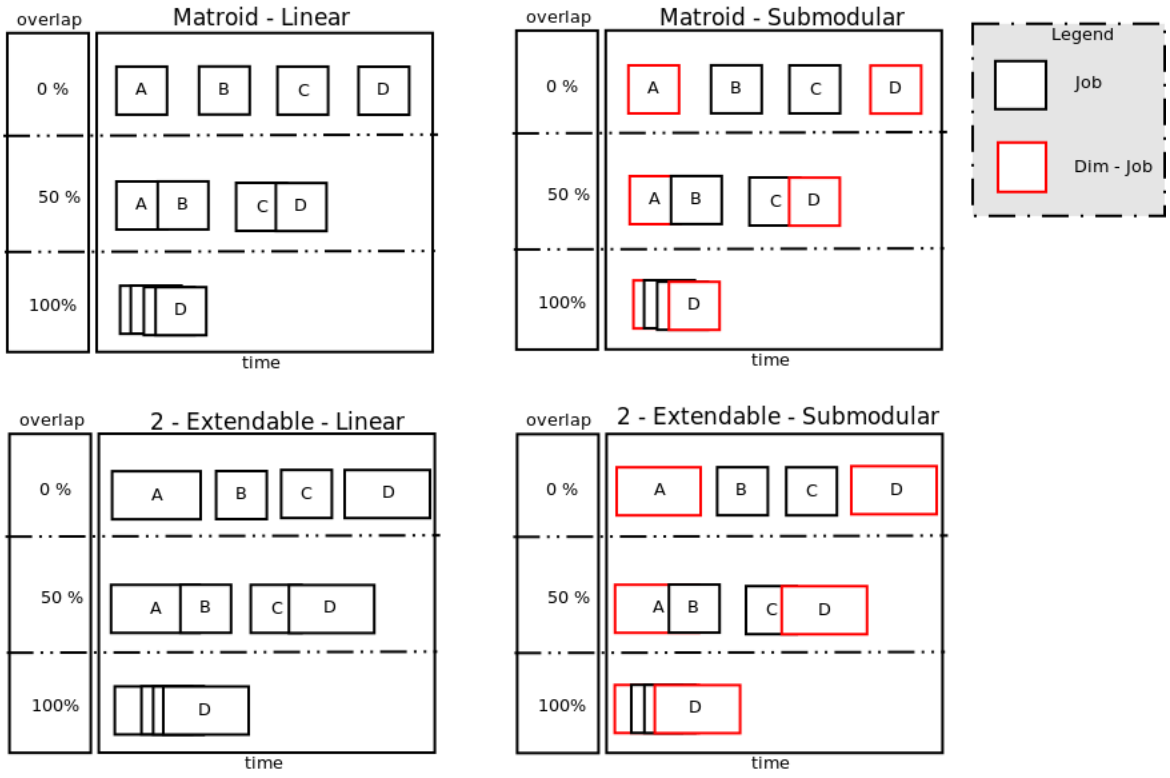

Figure 5.1: Contrived Schedule of Jobs - Jobs are scheduled and a scheduling return (revenue) is assigned; a Job is a task in the schedule with no conflicts; a Dim-Job is a Job that has a conflict with another Job in terms of scheduling returns

Figures 5.2, 5.3 and 5.4 depict the scheduling results for job schedules as presented in Figure 5.1.

Figure 5.3 depicts a 2-extendable constraint set with a linear objective function. The graph plots how well greedy performed compared to the optimal solution (i.e., vertical axis) versus job overlap. When the jobs have no overlap or they completely overlap, greedy performs optimally. However, if only some jobs overlap, then greedy

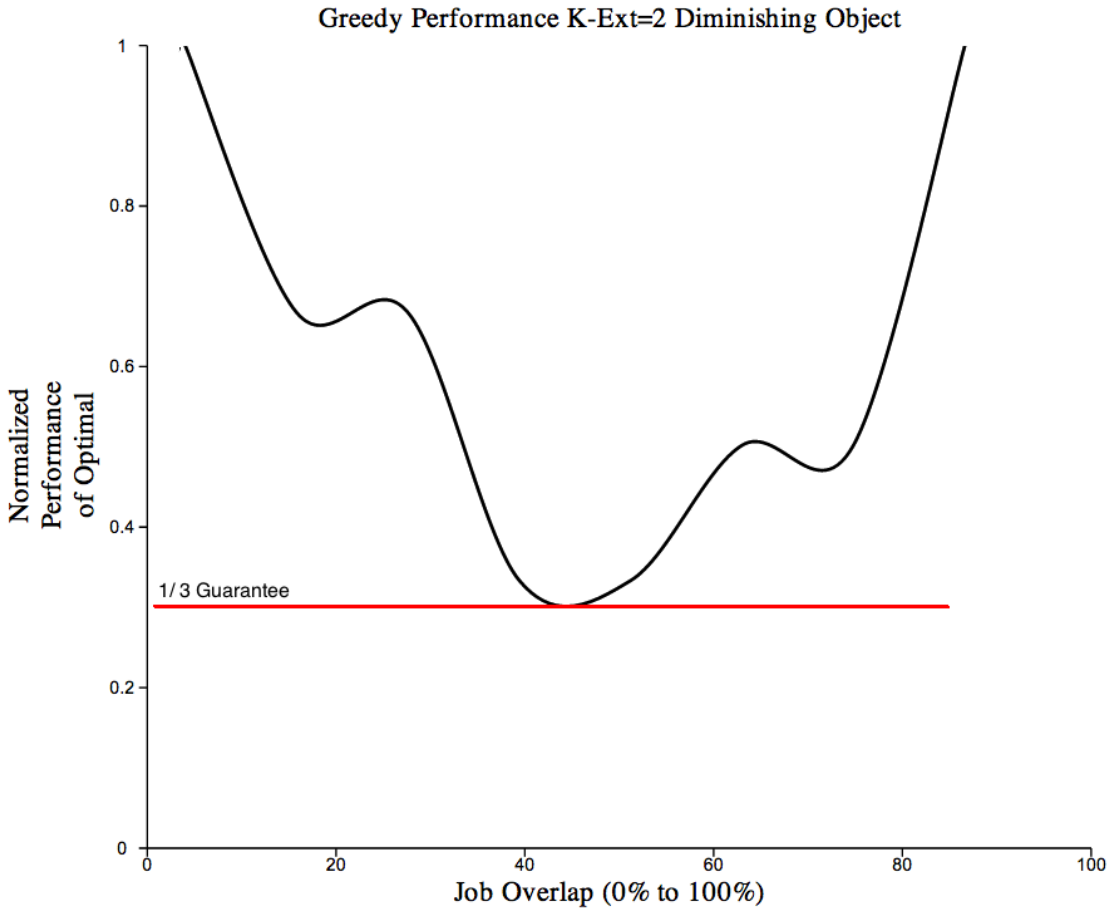

Figure 5.2: k-Extendable-Submodular: Greedy algorithm performance

starts to make poor choices and its schedules degrade compared to the optimal schedule. However, there are guarantees that the greedy algorithm will not degrade to less than half the optimal solution. From the figure it is clear that this is true. When the jobs overlap approximately 50% of the time, the greedy algorithm approaches $\frac{1}{2}$ of the optimal solution.

Figure 5.2 depicts a 2-extendable constraint set with a sub-modular objective function. In this scenario jobs A and D (cf. Figure 5.1) are in conflict and returns are diminished by the objective function if both jobs are scheduled during the same scheduling cycle. The graph plots greedy performance compared to optimal versus job overlap. The problem formulation for this scenario guarantees that greedy will produce a schedule of at least $\frac{1}{k+1}$ which is $\frac{1}{3}$ for this example. Notice greedy never

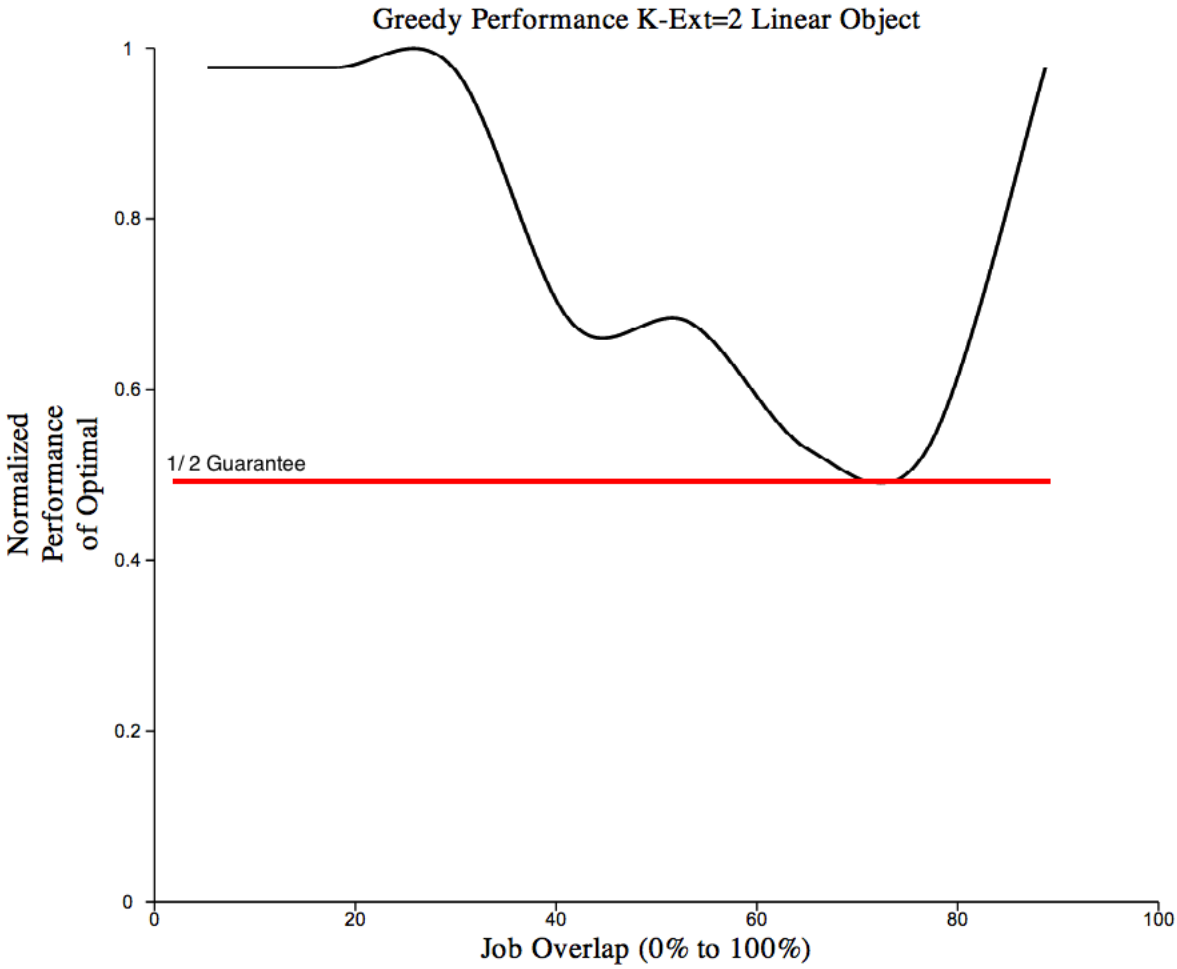

Figure 5.3: k-Extendable-Linear: Greedy algorithm performance

schedules worse that $\frac{1}{3}$ of the optimal schedule.

Figure 5.4 depicts a matroid constraint set with a sub-modular objective function. In this scenario jobs A and D are in conflict. However, all the jobs are unit jobs. This additional constraint causes the model to be a matroid. The graph plots greedy performance compared to optimal versus job overlap. The problem formulation for this scenario guarantees that the greedy algorithm will produce a schedule of at least $\frac{1}{k}$, which is $\frac{1}{2}$ for this example.

In this section, the jobs as described in Figure 5.1 are schedules, and their start times are decreased to force increased job overlap. Forcing job overlap causes greedy

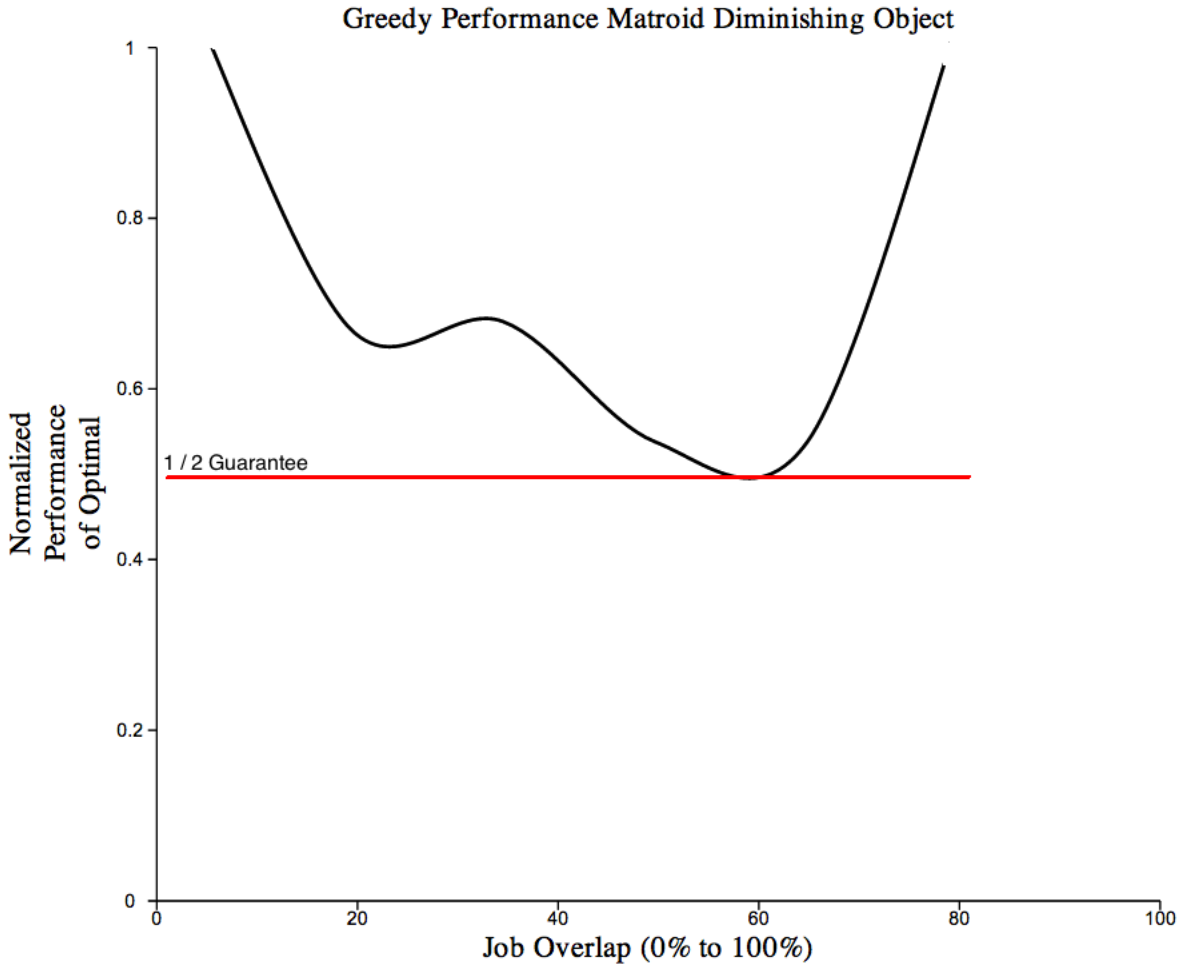

Figure 5.4: Matroid-Submodular: Greedy Aglorithm Performance

to occasionally make a bad scheduling decision. If all or none of the jobs overlapped, then greedy generates a optimal schedule. However, if selected jobs overlap, then greedy solutions may degrade in terms of revenue as depicted in Figures 5.2, 5.3 and 5.4.

Section 5.1.2 designs a scenario of using randomly generated jobs. The objective is to determine how often greedy produces optimal schedules versus lower limit guaranteed schedules. The jobs are generated for linear-k-extendable, matroid-submodular, and matroid-k-extendable variants of the scheduling scenario.

5.1.2 Service Queue Simulation using the Greedy Algorithm

In this section, we devise a simple queuing service that schedules user jobs (i.e., work load) to a server for processing. The work load is randomly generated for linear-k-extendable, matroid-submodular, and matroid-k-extendable variants of the scheduling scenario. The objective of the scheduler is to select jobs from the work load to maximize revenue. In this section we determine on average how well greedy solution's revenue compare to the optimal schedules revenue when the work flow is randomly generated. For example, we want to determine if a scenario will produce greedy schedules closer to optimal revenue versus closer to the guaranteed revenue. To realize this scenario, the following parameters are defined as referenced in the simulation model (cf. Figure 5.5):

- RNG: Random Number Generator that generates jobs using an exponential distribution as described by Equation 5.5. Where the input Y is a uniform distribution random variable, γ is the mean arrival rate of jobs, and the output X is an exponential random variable.
- JobSource: Create a job for each RNG event and submit it to the service queue.
- Job: For this scenario, a job has a submission time, start time, processing time, deadline and an associated revenue for use by the scheduler.
- Queue: A batch queue for randomly generated jobs. Jobs fill the queue until the scheduler is ready to start the next scheduling cycle.
- Schedule: The scheduler retrieves jobs from the batch queue and creates a feasible schedule. The scheduler tries to maximize the revenue of the job.
- Alg: For this scenario, the greedy algorithm and a known optimal algorithm are used to solve for the schedule, allowing to measure the quality of greedy schedules versus optimal schedules in terms of their revenue.
- lambda: mean arrival rate of user jobs, in this scenario it is set to 1.0 seconds.
- sigma: scheduling cycle time, in this scenario it is set to 10 seconds.

We selected a poisson distribution (i.e., exponential arrivals) to model our traffic pattern. This model is ideal for early research into SLA guarantees because it is easier to identify relationships between the workload model and the greedy algorithms

scheduling results. Further research using more complicated traffic patterns will benefit this research once a base line is developed using well known distributions.

$$X = -\frac{1}{\gamma} ln(1 - Y) \tag{5.5}$$

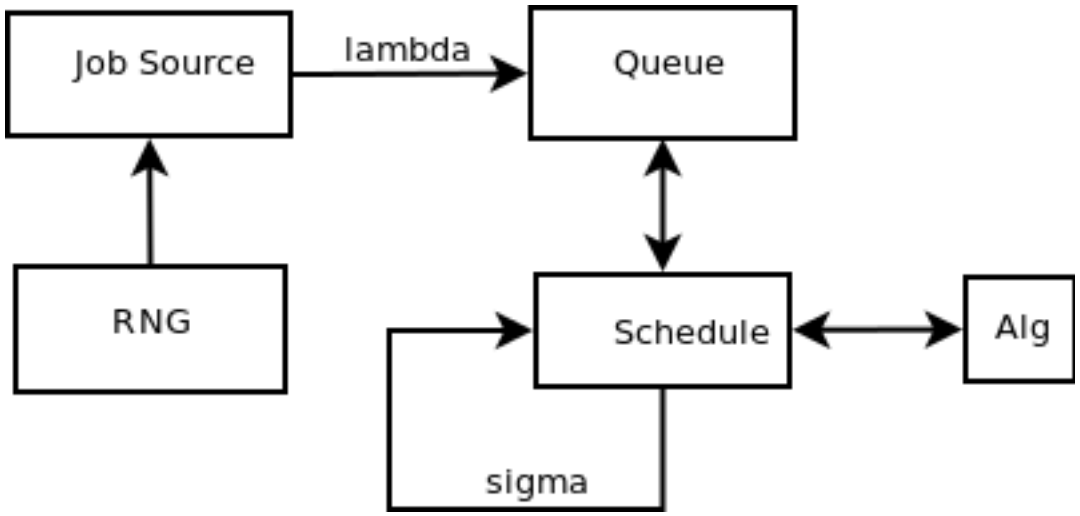

Figure 5.5: Scheduling Model of Single Queue Service

Figures 5.6, 5.7 and 5.8 graph the results of the three problem formulations of interest where greedy has guarantees of less than optimal. For this scenario the jobs were generated using an exponential distribution with a one second mean. The scheduler empties the batch queue every ten seconds and schedules the jobs using greedy. The schedule is also computed using our optimal algorithm. The greedy algorithm is then plotted for that scheduling cycle compared to the optimal algorithm. We now discuss results for 2-extendable with linear objective, 2-extendable with sub-modular objective, and matroid with sub-modular. Notice matroid with linear objective is not graphed because the theoretical result of using greedy is already optimal and our simulation confirmed this.

Figure 5.6 depicts a 2-extendable constraint set with a sub-modular objective function for scheduling a randomly generated workload. This compared to Figure 5.2 for the workload as depicted in Figure 5.1 shows that in reality, greedy on average performs closer to 70% versus the guaranteed lower limit of 33% or $\frac{1}{3}$. This more

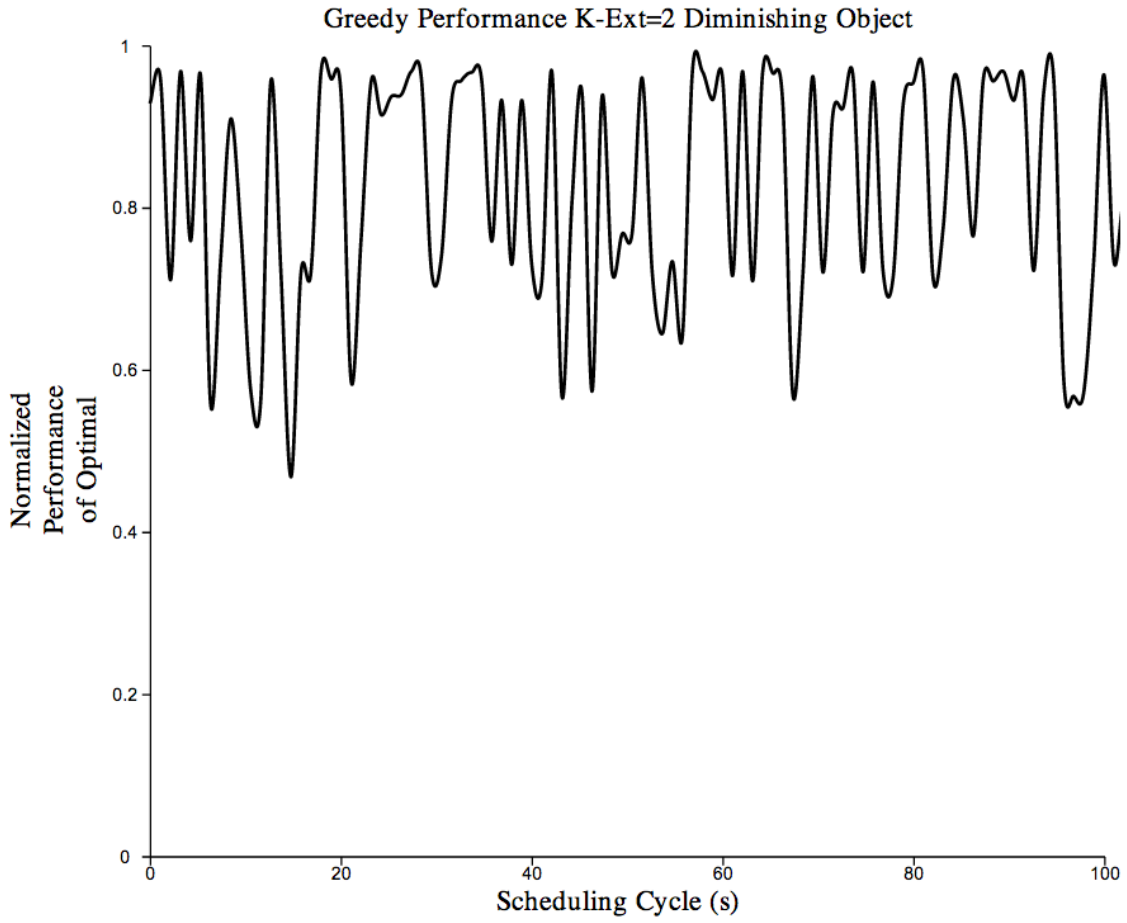

Figure 5.6: k-Extendable-Submodular: Greedy algorithm performance for 100 scheduling cycles

closely represents workload that would be generated in a real system.

Figure 5.7 depicts a 2-extendable constraint set with a linear objective function when scheduling random generated workloads. This compared to Figure 5.4 for the workload as depicted in Figure 5.1 shows that in reality, greedy on average performs closer to 88% versus the guaranteed lower limit of 50% or $\frac{1}{2}$.

Figure 5.8 depicts a matroid constraint set with a sub-modular objective function when scheduling random generated workload. This compared to Figure 5.3 for the workload as depicted in Figure 5.1 shows that in reality, greedy on average performs closer to 80% versus the guaranteed lower limit of 50% or $\frac{1}{2}$.

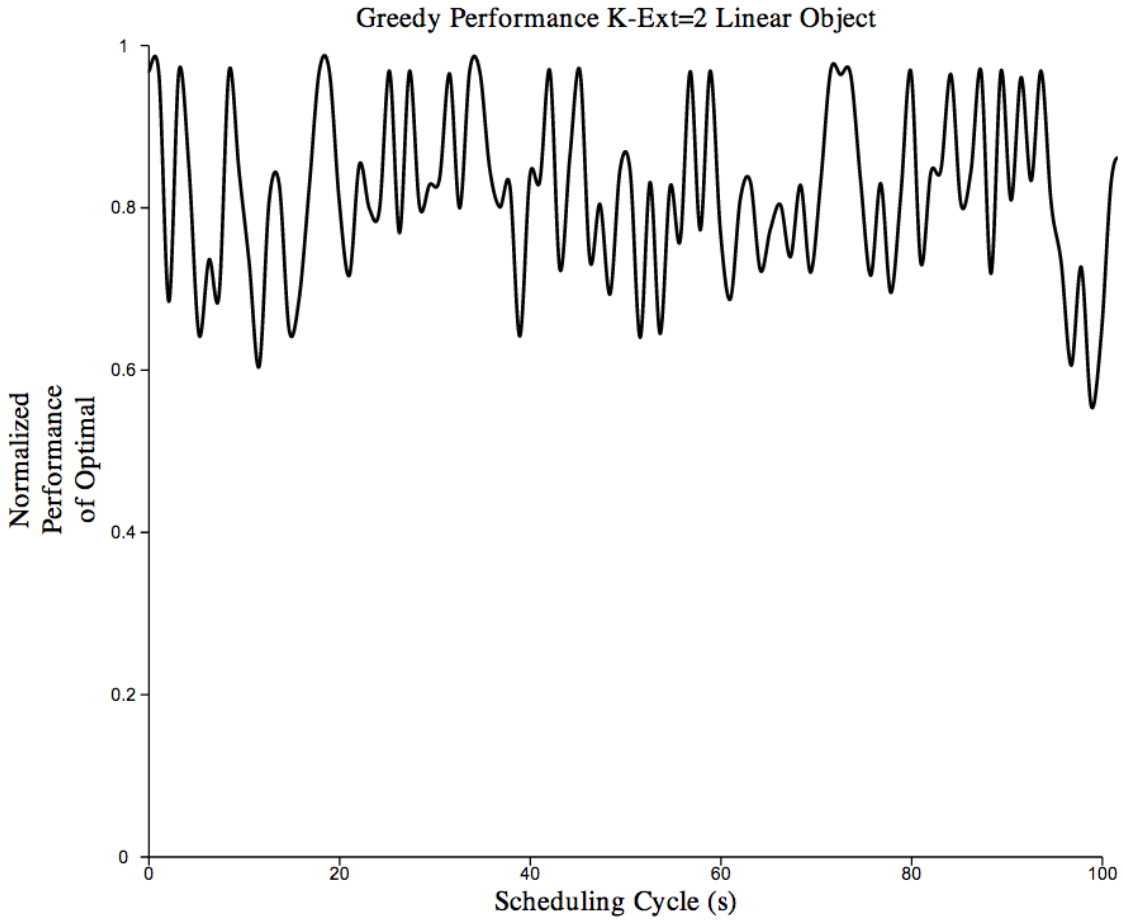

Figure 5.7: k-Extendable-Linear: Greedy algorithm performance for 100 scheduling cycles

In this section, we used an exponential workload to determine how the greedy schedules compare to the optimal schedule versus greedy's theoretical lower limit guarantees as determined by the problem formulation. The results clearly indicate that greedy will result in a schedule that is closer to its guaranteed value versus its optimal value. So we can confidently state using the greedy algorithm results in near optimal schedules and rarely produces schedules closer to its guaranteed lower limit for these scenarios. The relationship between these scenarios and the scheduling results is a promising area and will benefit from further research.

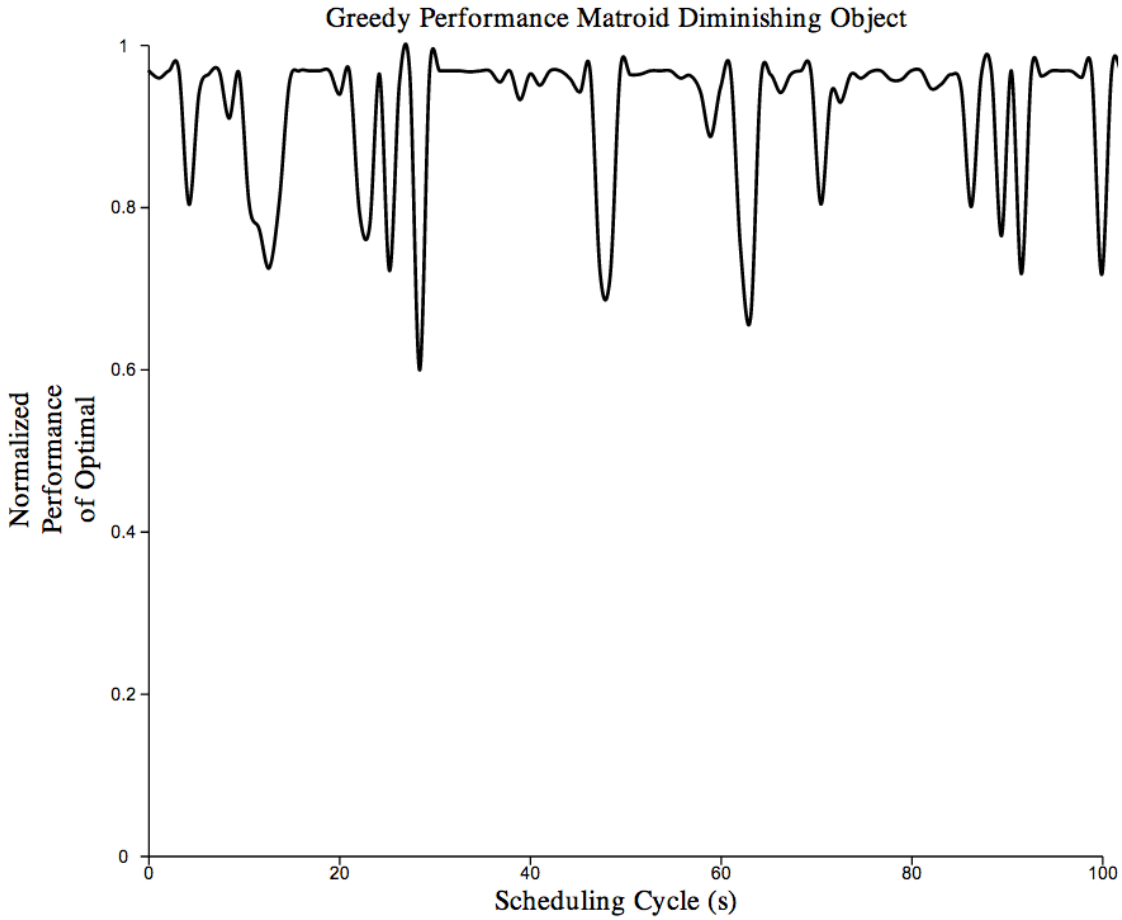

Figure 5.8: Matroid-Submodular: Greedy Algorithm Performance for 100 scheduling cycles

5.2 An Adaptive Mechanism for Model based Control of a Scheduling System

Using the results from the previous sections, an adaptive mechanism is proposed to adapt the problem formulation to best meet the needs of a user policy. In this scenario, three policies are proposed by the user, i) Utility-function; ii) Goal; iii) Action. These policies directly relate to the lower limit guarantees determined by the problem formulation. A utility-function policy requires that the problem formulation have a matroid constraint set with a linear objective function. If the problem formulation is as such, then the greedy algorithm will produce an optimal schedule. A goal policy requires the user also specify a lower limit guarantee and requires the problem

formulation to be either k-extendable and linear, k-extendable and sub-modular, or matroid and sub-modular. The goal policies are the most interesting from a scheduling perspective and for adaptive systems research. The previous sections focused on the goal policies for simulation. The third policy is unrestricted in that the problem formulation and/or objective function do not adhere to the mathematical structure requirements for greedy to produce a guaranteed lower limit schedule.

In the remainder of this section, we propose two adaptive mechanism for adaptive control. The first system is a feed forward control system (cf. Figure 5.9). The second system is a feed forward and feedback adaptive control system (cf. Figure 5.10). These systems are examples of model based control systems. These type systems are controlled by changing the model of the scheduling system and understanding what effect that will have on the results. For example, the problem formulation can be changed from a 3-extendable linear objective to a 2-extendable sub-modular objective in order to adapt to the guaranteed lower limit requirement. The remainder of this section describes the feed forward and the feed forward / feedback adaptive system models.

Figure 5.9 depicts the simple feed forward control model. The objective of this adaptive system is to profile the current workload and service model to determine what the current problem formulation is in order to ascertain the greedy algorithms lower guaranteed limit. This value is compared to the desired goal policies minimum limit. Comparing the desired lower limit and the expected lower limit, the adaptive system can alter the workload in order to change the problem formulation. Changing the problem formulation causes the greedy algorithm to achieve different guarantees. For example, the user may desire a lower limit scheduling of $\frac{1}{2}$ optimal value. However, the current work-flow jobs is such that the problem formulation is 3-extendable with linear objective. A 3-extendable linear objective only guarantees a $\frac{1}{3}$ lower limit. The feed forward controller produces a positive error and the job admission control will remove all jobs from the work flow so as to create a 2-extendable linear objective problem formulation which now has a $\frac{1}{2}$ lower limit guarantee. The feed forward adaptive controller components are as follows:

- Work flow Profiler — determines from the work load and system model the type of problem formulation.

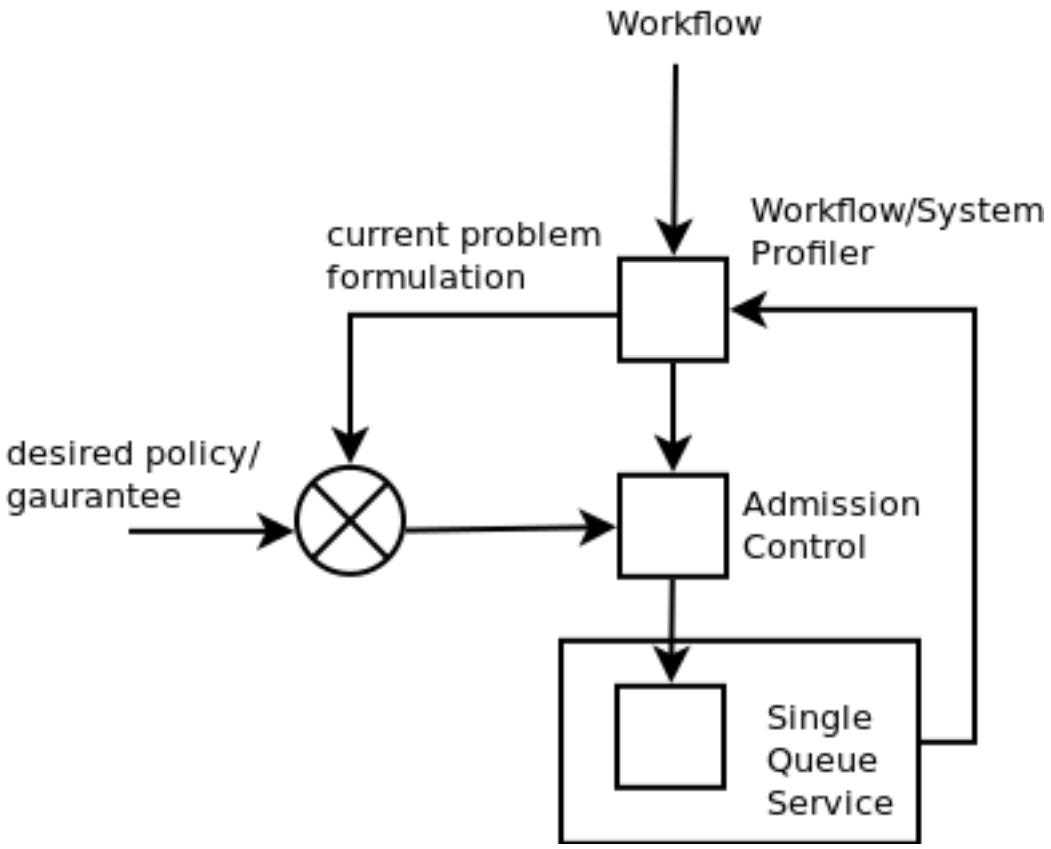

Figure 5.9: Feed Forward Controller

- Comparator — Compares problem formulation's lower limit (when using greedy) to the goal policies desired lower limit.
- Job Admission Control — Changes the problem formulation by moving or deleting jobs from the work flow which changes the problem formulation and causes greedy to schedule differently.
- Single Queue Service — (cf. Figure 5.5).

The feed forward adaptive controller evaluates the current problem formulation, compares it to the user requirement, and makes a change if necessary. For example, a change from a 2-extendable linear objective formulation to a 3-extendable linear objective formulation. However, it would be useful to investigate other formulations which have that same guarantee, but may result in a better schedule. For example, comparing a k-extendable-linear system to a matroid-sub-modular system. This comparison requires feedback to determine which is best. The reason this is required is

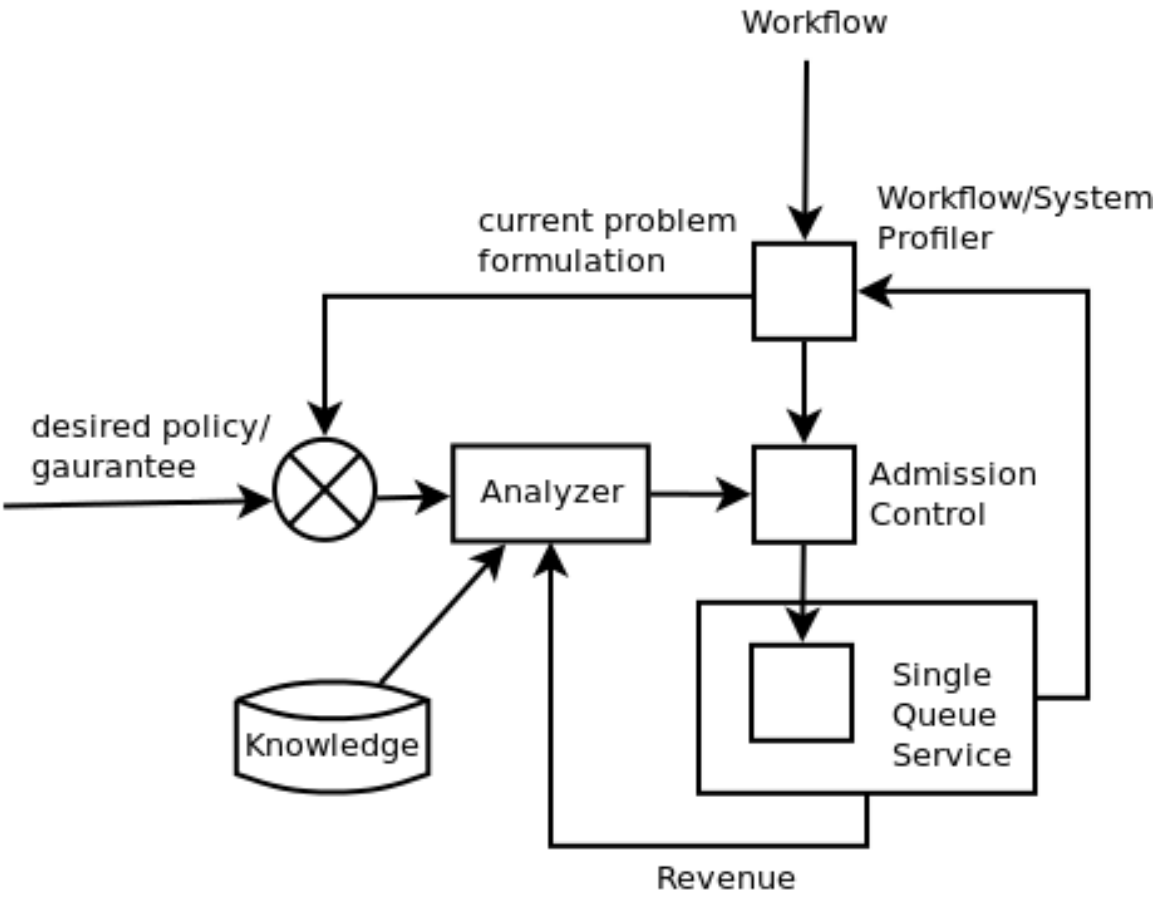

Figure 5.10: Feed Forwared and Feedback Controller

because both of these formulations actually have the same guaranteed lower limit, and feedback is required to determine which formulation produces greater revenue. The feedback adaptive controller (cf. Figure 5.10) requires additional components to take advantage of the feedback. These components are as follows:

- Analyzer: Uses the knowledge database of scenario averages along with current revenue of selected problem formulation to make changes until a good problem formulation is found.
- Knowledge Base: Used to keep information on how scenarios generally perform. For example information from Figures 5.6, 5.7, and 5.8.

5.3 Summary

This chapter explored the use of model based control. We evaluated three problem formulations for their theoretical lower limits when being solved by the greedy algorithm (cf. Figures 5.2, 5.3, and 5.4). We also implemented a simulation using these formulations for the simple queuing service to evaluate the effects when scheduling an exponentially distributed workload (cf. Figures 5.6, 5.7, and 5.8). Using these simulation results, we propose two adaptive control models to implement model based control to change the problem formulation (i.e., by changing workload) so as to achieve the user's desired policy and minimum guaranteed lower limits.

Chapter 6

Control Strategies

In Chapter 4 we used scheduling as a strategy to manage resource provisioning. In Chapter 5 we employed a model-driven approach to provide scheduling guarantees. In this chapter we use control theory to manage compute systems. Control theory and scheduling approaches are different. Control theory uses mathematical models to describe the behavior of systems. For example, if the model is linear and time invariant (LTI), its performance can be controlled by designing an LTI controller and applying feedback from the system [Oga87] [HDPT04]. Proponents of the use of control theory for computing systems include Murray [Mur03], who early on recommended substantial research into control integration for computing systems; Hellerstein [HDPT04], who showed that even non-linear computing systems can be orchestrated with simple linear controllers; Xu [Xu07], who effectively used control theory for log processing; and Litoiu [SILI10], who successfully formulated model predictive control techniques based on Kalman filters. The application of control theory to computing systems has proven useful in providing more resilient and fault tolerant systems.

In this chapter, we present control strategies to address cloud computing research challenges such as those described by Buyya [YB06] and Yang [YTX04]. In particular, we focus on admission control of a resource provider (i.e., how to handle stochastic workloads or disturbances in the resources). Other challenges, described by Litoiu [SILI10], include how to handle poor feedback signals through a predictive controller.

The scenario depicted in Figure 6.1 addresses the issue of admission control whose objective is to manage access to a server, service, or resource. In this scenario, the

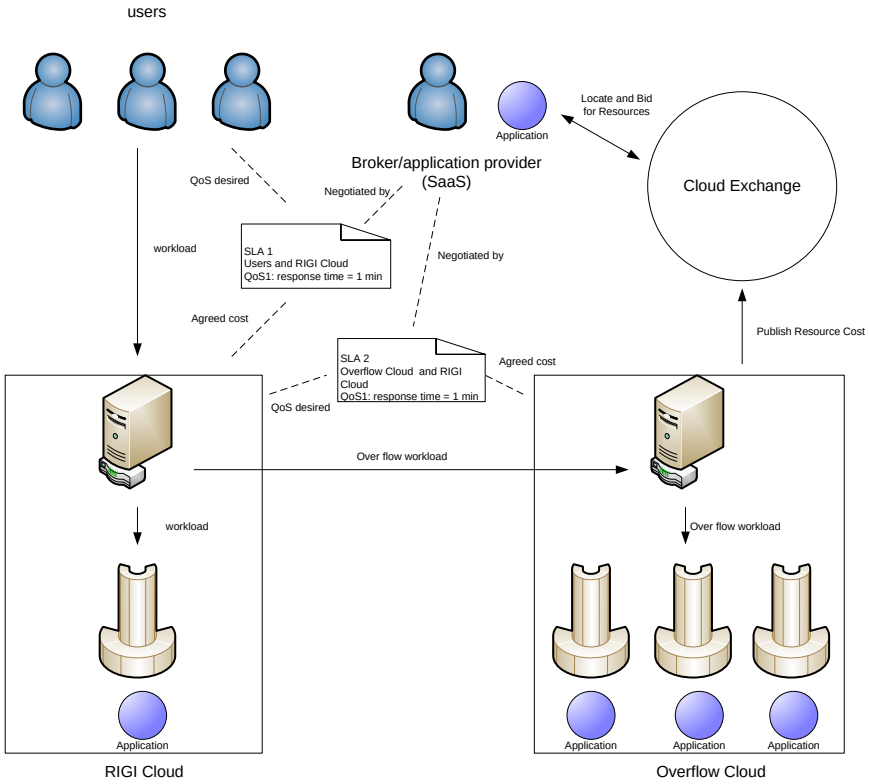

Figure 6.1: Admission Control Scenario

resource is a cloud provider called ECS-X1 Cloud. ECS-X1 Cloud uses several data centers for its resources. ECS-X1 Cloud serves its clients (users) for which service level agreements (SLAs) have been negotiated and established through an SaaS broker. The service level agreements are a set of quality of service (QoS) metrics. For example, a QoS metric may be a time limit on task execute. ECS-X1 Cloud evaluates its resources and determines how many SLAs it can manage. For example, the cloud provider may be able to support ten clients and guarantee that tasks will take no longer than one minute to execute. ECS-X1 Cloud uses the task's estimated execution time to determine if the cloud has enough resources to achieve the client's desired performance requirements. This works well except when clients submit their workloads at the same time. If many clients use the cloud resources at the same time, then response times increase. The result is an SLA violation and the cloud may have to pay a penalty. The issue is that admission control decisions are based on the average workload of each user. This policy fails if all users submit their workloads at the same time. To address this issue, ECS-X1 Cloud can use an overflow cloud. The

overflow cloud is another cloud provider that has an agreement with ECS-X1 Cloud (i.e., ECS-X1 Cloud has an SLA agreement with the overflow cloud). The overflow cloud costs ECS-X1 Cloud money, and thus it is in the ECS-X1 Cloud's best interest to use the overflow sparingly. In this scenario, ECS-X1 Cloud has two admission control decisions to make. The first decision is accepting new clients, and the second is using an overflow cloud to avoid an SLA violation.

Since this scenario is control related, we refer to the taxonomy classification from Table 3.2 and Figure 3.2. Several works provide insight for potential solutions to this scenario including Buyya [YB06], Yang [YTX04], Hellerstein [PGH+02] [Hel04] [HDPT05], Diao [DHP+05], Lu [LSTS99], Xu's Ph.D. thesis [Xu07], and books by Hellerstein [HDPT04] and Ogata [Oga87].

The remainder of this chapter is structured as follows. We describe a control system approach to manage computing systems. We identify control issues and propose solutions based on the literature mentioned above. The solution is implemented and evaluated using the OMNET simulation environment. For each solution we detail the design and discuss the simulation results. We also present a novel approach to open loop control using stochastic models. Lastly, we present chapter contributions and avenues for future research.

6.1 A Control Systems Approach

This section describes how to use control theory to manage a cloud computing system. Kalyvianaki [Kal09] uses a Kalman filter approach to predict resource usage, Hu [HWIL09] uses statistics to manage resource provisioning, and Dutreilh [DRM+10] uses latency, power, and input oscillations as control inputs for resource allocation. These methods are all forms of control, albeit by means of different models and solutions. We identify several issues when applying these control systems to other environments. In particular, we propose solutions based on the literature itemized in Table 3.2 and Figure 3.2.

6.1.1 Control Theory Basics

The objective of control theory is to achieve a desired behavior from a system (e.g., a power plant or web service). It can be defined as a set of elements, and the relationships between the elements.[1] General system characteristics include structure, topology, and behavior. Systems can be linear or non-linear. In this section we review the basics of control theory for linear systems as they are modeled, analyzed, and controlled.

Linear systems are easier to understand and analyze than non-linear systems and are characterized by two linear system properties. The first property is superposition—the response to multiple inputs equals the response to individual inputs summed. The second property is homogeneity—the superposition principle holds true for scaling of the inputs. With such properties, the response from the system to complex inputs can be described by summing individual responses of simpler inputs.[2] [Oga87] These properties simplify the analysis of the system to determine how it will respond to any random input.

Linear systems can be divided into two system types, linear time varying systems and linear time invariant systems. Linear time invariant systems (LTI) are easier to analyze because the system response is only affected by current state and current inputs. The type of system determines which tools and techniques are applicable to analyze and create controllers for the system. This section focuses on LTI systems and how they are applied in computing systems.

A linear time invariant system is described by linear difference equations or linear differential equations. The description of the LTI system depends whether the system is a discrete or continuous system. Equation 6.1 is an example of a differential equation versus a difference equation. In this section we only use the discrete equation.

[1] http://en.wikipedia.org/wiki/System
[2] http://en.wikipedia.org/wiki/Linear system

$$\begin{array}{ll} \sum_{k=0}^{N} a_k \frac{d^k y(t)}{dt^k} = \sum_{k=0}^{M} b_k \frac{d^k x(t)}{dt^k} & \text{Differential equations describing LTI systems} \\ \sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k] & \text{Difference equations describing LTI systems} \end{array} \quad (6.1)$$

Referring to the discrete case of the LTI equation, the order of the system N and M indicate how many previous outputs and inputs are being used to determine the current output. This can be viewed as the system having a memory, only when N=1 and M=0 is the system memoryless (M and N are referred as the model's structure). The memoryless property is important because it can simplify the analysis of the system. However, complex systems are more likely to be modeled with the memory property if they are to achieve stability. For example, a simple server and queue have a memory since the output response time is dependent on how full the queue is. The $y[n-k]$ and $x[n-k]$ in Equation 6.1 refer to system outputs and inputs, respectively, and model parameters a_k and b_k describe the relationship between the input and output at interval k.

Several issues need to be resolved if a real world computing system is modeled using Equation 6.1. The first issue is that it is a single input single output equation. For systems with more than one input and output, a set of LTI equations is necessary for each input output pair. The second issue is how to select the order (i.e., M and N in Equation 6.1) of the model to best match a managed system. Higher order system models are more difficult to work with and complicate controller design. The third issue is how to determine the models parameters (a_k and b_k). The fourth problem is how to deal with system non-linearities. Non-linear models of systems are much more difficult to work with. A typical solution is to expand the non-linear model as a Taylor series, then identify linear components and use them to construct controllers [HDPT04].

The first and second issues refer to the order of the system of equations used to create the model. Yang [YTX04] proposed the use of adaptation on simple first order and second order models to control higher order systems. The third issue is parameter

identification. A popular approach to solve this issue is to collect input and output responses for a broad range of inputs and estimate responses. This is accomplished using the least squares regression method. The fourth problem of non-linearities, is how to identify an operating region and a range of control inputs and linearize the non-linear region using least squares regression (i.e., attempt to construct a linear model of a non-linear region). This method provides an approximation and if the system drifts too far from the operating region, the system could become unstable.

Ascertaining a good system model and identifying good system parameters is difficult for real world compute systems. Figure 6.2 depicts this process of identifying a system and determining its parameters. However, setting the parameters themselves is quite a tedious process in computing systems. Figure 6.3 depicts the process of parameter selection. One difficulty is to configure the workflows (i.e., training data) to get fully exercised over a specified operating region. This is not easy according to Xu [Xu07] and requires manual evaluation. Another issue is the type of models selected for identification of the actual system. In this section only LTI systems are addressed because of the complex nature of computing systems and their inability to adhere to a static model. Other techniques are proposed to address issues such as higher order complex models, lack of consistent feedback, and dynamically changing compute environments. Figure 6.4 is the classification used in this chapter. Control systems are identified as simple first and second order models. For non-linear and higher order models, other techniques are used.

After a system model is identified and the system parameters are determined around an operating region, the controller is designed to achieve a desired response from the system. Simple models of systems make their controllers easier to construct. Equation 6.2 represents a first order model of a queuing system (i.e., one server and one queue as depicted in Figure 6.5). The input is $u(k)$ and the output is $y(k)$. The equation is consistent with a first order system of Equations 6.1 where $N = 1$, $M = 1$, $a_0 = 1$, $a_1 = -a$, $b_0 = 0$ and $b_1 = b$.

Using this system model, the parameters a, b are identified. Parameter a represents feedback of the current output at interval k to the next output at $k+1$. If a is positive then the feedback is positive, if a is negative, then the feedback is negative. Parameter b represents a weight to the current input to the system at time/interval k, the input

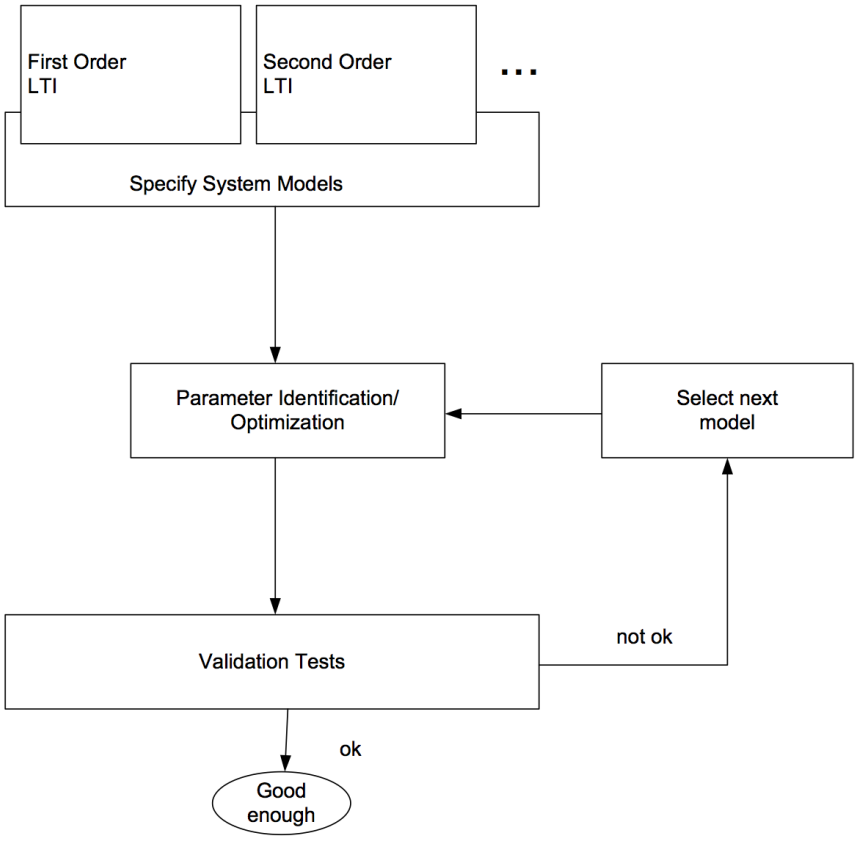

Figure 6.2: Multi-Model Selection

could add or remove from the output at $k+1$ depending on its sign and its value.

$$y(k+1) = ay(k) + bu(k) \tag{6.2}$$

There are several techniques to create controllers for first and second order LTI systems. In this section, we design a controller for a simple queuing server. For this scenario, we require the queuing service to maintain 75% processor utilization.[3] Although queuing theory techniques are better suited to handle this problem, control theory provides a valid solution that is easy to implement for more complicated systems. Before the controller is created, the system's behavioral responses need to

[3]This value is based on Microsoft's *patterns and practices* which can be found at http://msdn.microsoft.com/en-us/library/bb924364.aspx .

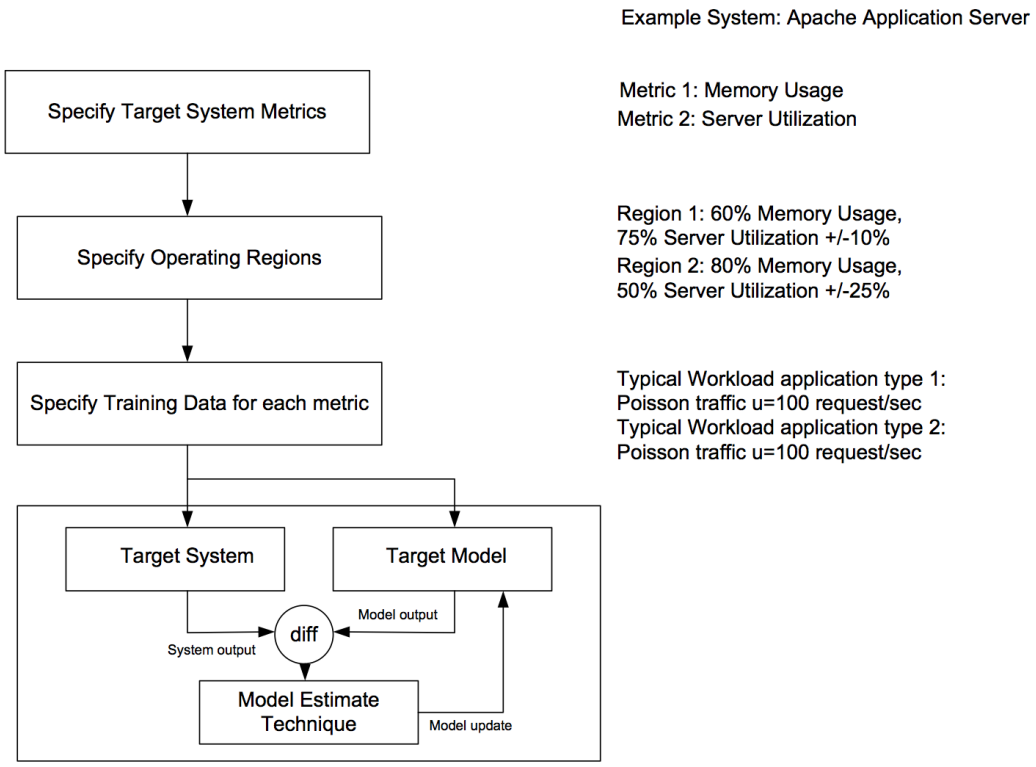

Figure 6.3: Parameter Identification / Optimization

be determined. For example, its steady state (long term) and transient (short term) responses need to be determined around the operating region of interest. These responses can be determined by time-domain solutions or by frequency-domain solutions. They both produce the solution and the value of the measured parameter of the system for a specified interval k and known inputs from $n = 0$ to $n = k - 1$. Using the solved equations, and assuming the computing system matches a first order system of Equation 6.2, the initial condition, impulse and step responses are easily calculated through either time domain or Z-transform methods, and if the parameters a and b do properly model the computing system then we should be able to determine the responses accurately. Equations 6.3, 6.4 and 6.5 provide the initial condition, impulse, and step responses for the first order system defined by Equation 6.2. Note that the step response in Equation 6.5 has two components which define the steady state and transient responses. Another important response in control analysis is the ramp response (cf. Equation 6.6) and the frequency response.

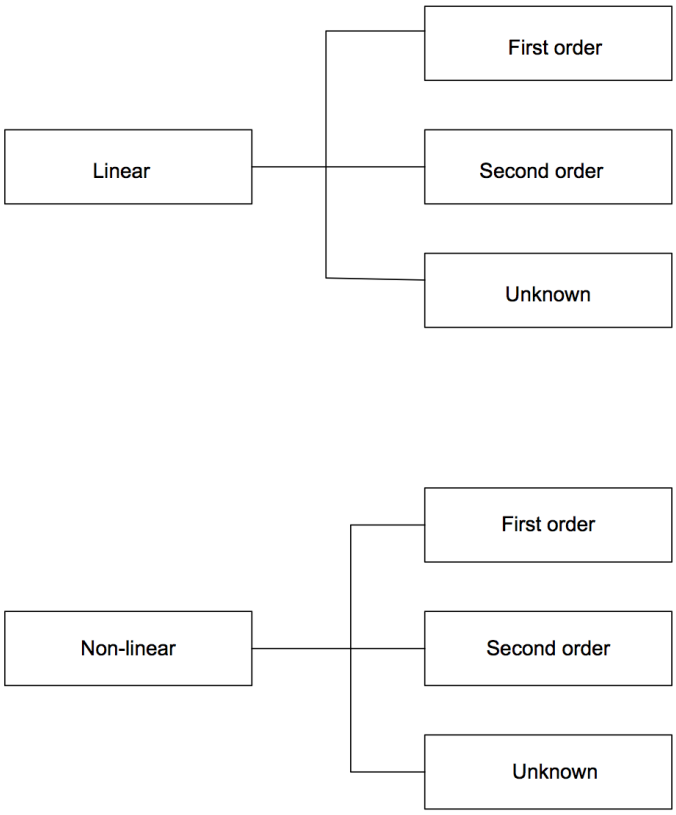

Figure 6.4: Simple control model classification

$$y(k) = a^k y(0) \tag{6.3}$$

$$y(k) = ba^{k-1} \tag{6.4}$$

$$y(k) = \frac{-ab}{1-a}a^{k-1} + \frac{b}{1-a} \tag{6.5}$$

$$y(k) = \left[\frac{b}{1-a}k - \frac{b}{(1-a)^2}\right] + \left[\frac{b}{(1-a)^2}\right]a^k \tag{6.6}$$

With a model of the system responses, we can now design the controller. A popular controller is the Proportional - Integral - Derivative controller (PID). The

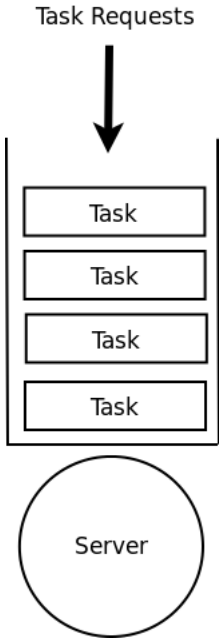

Figure 6.5: Simple Server Queue

proportional component responds to the magnitude of the error, the integral part removes the steady state error, and the derivative part reduces overshooting of the desired target.

There are two types of control, open loop control for well understood systems and closed loop control for systems whose models are sensitive and change over time. Figure 6.6 depicts a simple proportional feedback controller and Equation 6.7 describes the closed loop transfer function for the first order system. In this case we are using the z-transform to represent the closed loop transfer function. Using this form, it is easier to solve for the controller gain (i.e., K_p) which will determine the stability, accuracy, settling time, and overshoot (i.e., SASO properties) [HDPT04]. Equations 6.8, 6.9, 6.10 and 6.11 can be used with any first order system implementing a proportional feedback controller to determine a good value for gain (K_p).

$$\frac{Y(z)}{R(z)} = \frac{K_p[\frac{b}{z-a}]}{1 + K_p[\frac{b}{z-a}]} \tag{6.7}$$

$$\text{stability: } |p_1| = |a - bK_p| < 1 < 1 \text{ is the unit circle of a root locus plot} \tag{6.8}$$

$$\text{accuracy: } e_{ss} = 1 - \frac{bk_p}{z - a + bk_p} < \text{desired minimum error:} \tag{6.9}$$

$$\text{settling time: } k_s \approx \frac{-4}{log|a + bk_p|} < \text{desired settling time:} \tag{6.10}$$

$$\text{overshoot: } M_p = |a - bK_p| < \text{desired overshoot:} \tag{6.11}$$

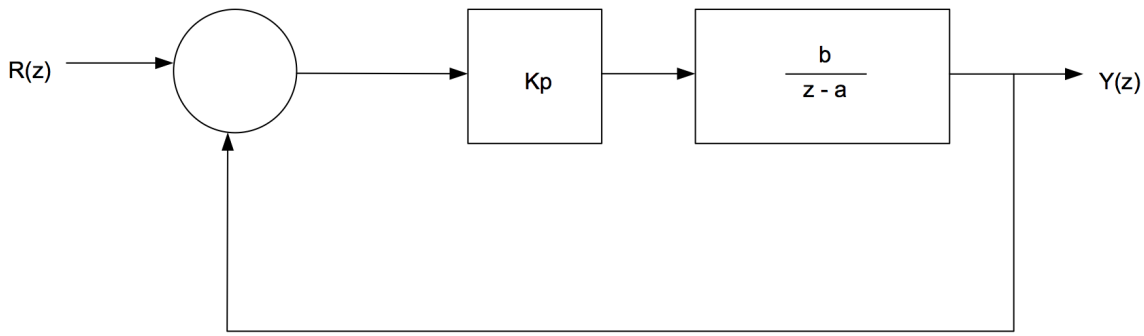

Figure 6.6: Proportional Feedback controller for a first order system

Not all SASO properties can be achieved using proportional control. More complicated controllers such as PI, PD, or PID may be required. Equation 6.12 is the discrete model of the integral component and Equation 6.13 represents its z-transform. The integral component can reduce the error of a system to zero. It does this by adding all the errors together over time. Even small errors can accumulate over a period of time and allow the control to adjust the system. Equation 6.14 is the discrete model of

the derivation component and Equation 6.15 represents its z-transform. The derivative component is sensitive to the momentum of the system. For example a large correction to a system can overshoot the desired target performance. The derivative component can adjust for this before overshoot occurs.

$$u(k) = u(k-1) + K_I e(k) \tag{6.12}$$

$$\frac{Y(z)}{R(z)} = \frac{K_I z}{z-1} \tag{6.13}$$

$$u(k) = K_D[e(k) - e(k-1)] \tag{6.14}$$

$$\frac{Y(z)}{R(z)} = \frac{K_D(z-1)}{z} \tag{6.15}$$

The PID controller is easy to design because the proportional, integral, and derivative components are easier to combine in the z-domain versus the time domain. This simplifies the construction of the controller in mathematics. Equation 6.16 depicts all three components of the controller used to control the system (cf. Equation 6.2). It is important to note that different combinations of proportional, integral and derivative can be used together to control a system.

$$\frac{Y(z)}{R(z)} = \frac{[K_P + \frac{K_I z}{z-1} + \frac{K_D(z-1)}{z}][\frac{b}{z-a}]}{1 + [K_P + \frac{K_I z}{z-1} + \frac{K_D(z-1)}{z}][\frac{b}{z-a}]} \tag{6.16}$$

For a more detailed discussion on how to apply control theory to computing systems we refer to Chapter 5 of Xu's dissertation [Xu07].

An important issue using control theory in computing systems is how to deal with packet loss when using a feedback controller. Control feedback from a system could be lost or delayed, and without good sensor data or feedback signals, the controller can negatively influence the system. As systems become more complex, modeling them as first or second order systems may not be sufficient. Generally higher order

models are needed, but their controllers are more complex. Another issue is that computer system models tend to drift from their actual model. In this case system identification must be done again to tune the model parameters to better match the system. To manage these issues, we explore predictive approaches below. Predictive approaches address the problem concerning lack of good feedback signals by taking advantage of filtering solutions to predict what the sensor values could be based on previous values [SILI10]. Section 6.2 investigates the use of filtering feedback signals for control.

6.2 The Kalman Filter Solution

At the beginning of this chapter we outlined a scenario of a cloud provider (i.e., ECS-X1 Cloud) that is hosting an application for a set of users. ECS-X1 Cloud accepts user workload in accordance with an SLA made between the Cloud and the user. In a sample agreement, ECS-X1 Cloud guarantees a minimum response time of one minute for user requests at an average aggregated workload of 0.34 requests per minute. One of the policies implemented at ECS-X1 Cloud is to ensure that the mean user workload of all users does not exceed one third of the computing capacity used by the hosted application. However with this policy only 12 to 13 percent of the requests are executed within the QoS requirement (i.e., one minute). Users submit tasks conforming to a Poisson distribution at a mean rate of 0.34 requests per minute (i.e., one request every three minutes). ECS-X1 Cloud can easily handle one request per minute, but if a burst of requests is generated the system may violate its SLAs. This is caused by the server queue filling up and causing a temporary delay in execution. To manage the problem, ECS-X1 Cloud has an agreement with an overflow cloud (e.g., Amazon EC2) that ECS-X1 can use to offload requests. Using the overflow cloud to guarantee its SLAs costs money, so ECS-X1 Cloud needs to use the overflow cloud intelligently. Thus, ECS-X1 Cloud needs a control system to make this happen.

Using an overflow cloud is increasingly popular. However, as with many of these systems, it can be difficult to predict or measure how well an application is performing on its hosted server. Most cloud applications hosted on isolated virtual machines share the server's resources with other applications hosted on virtual machines. Measuring a server's utilization does not indicate how well the application is managing its

workload. For example, three applications (i.e., A,B and C) hosted in separate virtual machines are being provisioned resources from a single physical host. It is conceivable that at time interval k=5, virtual machines with application A and B are getting 90% of the CPU and application C is only getting 10%. Then at time interval k=6, application C is getting 90% all for itself. This makes it difficult for a controller to make a decision based on the physical host's utilization on how to direct the workload.

Interesting works related to this problem include Litoiu [SILI10], Lu [LSTS99], Hellerstein [PGH+02], and Yang [YTX04]. These works, with the exception of Litoiu [SILI10], use control theory and PI controllers to manage their systems. These approaches are challenged due to the lack of a good feedback signal (i.e., server utilization). Litoiu's [SILI10] technique, which uses a Kalman filter to predict sensor readings based on past readings, is a better solution for this situation. The idea is to use a Kalman filter to predict when to direct traffic to an overflow cloud based on how well the application is performing. Since server utilization is not generally available, the control system will need to execute test jobs to measure performance.

The remainder of this section describes our Kalman filter approach including our simulation experiment.

6.2.1 The Kalman Filter

The Kalman Filter was designed by R. E. Kalman in 1960 [K+60]. Its original purpose of design was to smooth erratic measurements using previous and predicted values. This recursive solution uses erratic measurements from the system. This concept applies nicely to our application measurements which are at best a guess derived from using the application. The use of non-deterministic control is identified by Murray [Mur03] as an area that would benefit from substantial research.

The Kalman filter uses a linear stochastic difference equation to model a controlled process. The two main Equations 6.17 and 6.18 are the key equations to characterizing the filter. The Kalman filter is recursive by design with two distinct phases, a prediction phase (Equation 6.17) and a correction phase (Equation. 6.18). These two phases work together to ascertain a better measure of the system process.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{6.17}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{6.18}$$

The equations variables are described as follows:

- $\hat{x}_k^-$ — is the a priori prediction for step k. This uses previous corrections to try to predict the next state of the process being managed.
- A — is the state transition matrix.
- $\hat{x}_{k-1}$ — is the post priori correction at step $k-1$ described below.
- Bu_{k-1} — B is the control transition matrix and u_{k-1} is the value of the control inputs (1 or 0 for this simulation). Where 1 grants access of requests to server queue and 0 denies access of requests to server queue.
- $\hat{x}_k$ — is the post priori correction at step k. This correction step uses both the a priori prediction $\hat{x}_k^-$ and measurement z_k to produce a better estimate.
- K_k — is the Gain associated with minimizing the covariance. We simulate the result using different values as discussed in Section 6.2.3.
- $z_k - H\hat{x}_k^-$ — is the error between the predicted value and the measured value.

Equations 6.17 and 6.18 are implemented in the simulation as described by Equation 6.19. The execution time of the resource is being estimated and compared to determine admission access to the service.

$$\begin{bmatrix} w_k \\ \hat{w}_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta T \\ 0 & -\Delta T \end{bmatrix} \begin{bmatrix} w_{k-1} \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ \Delta T \frac{\mu}{\lambda} \end{bmatrix} u_{k-1} \tag{6.19}$$

Equation 6.19 describes how the controller estimates the next expected execution time. This is done by sampling the traffic rate and estimating the resources processing rate ($\Delta T \frac{\mu}{\lambda}$). This is used along with the amount of processing time which has passed since the estimate was last calculated (ΔT). The equation also uses the previous controller input ($u_{k-1} = 1, 0$) to estimate the service execution time.

The Kalman filter has also other capabilities. This includes a model of process and measurement noise which have an effect on $\hat{x}_k^-$ and z_k respectively. Equations 6.17 and 6.18 were implemented in the simulation and the results are compared to other solutions.

6.2.2 The Experiment

The objective of the experiment is to evaluate how we can adapt to the application's current resource allocation and user workload so as to realize benefits. In this case, the benefit is an increase of the number of user application requests that execute within a Quality of Service requirement (i.e., Time Limit of Execution Agreement). The caveat is the application has no method to gather application measurement data for the control system. The solutions to be compared are a simple on/off controller which submits a test job and measures the application's execution time, a Kalman filtered controller which estimates and corrects using a test job, and no controller which allows all requests to enter the resource queue even if there is no possibility of executing them within its QoS requirement. It is assumed that the applications task queue is FIFO (First In First Out).

Figure 6.7 depicts the feedback simulation network. The key components are the controller and throttler.

A total of 375 scenarios as described below with the three types of controllers are compared (i.e., none, on/off, and Kalman). Each controller has five different gain variables set as 0.5, 0.75, 1.0, 1.25, and 1.5. Manual observation determined for this scenario that the Kalman gain variable most profoundly affected results between 0.5

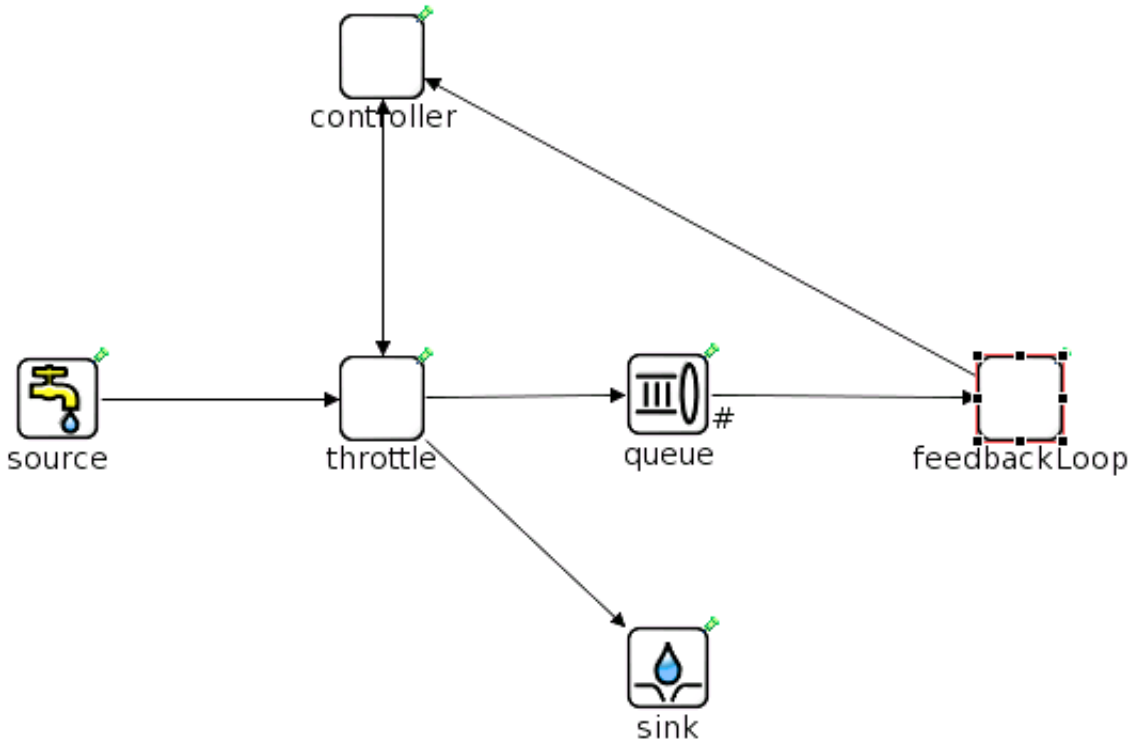

Figure 6.7: Control System Simulation

and 1.5. Varying the gain variable provides insight into the sensitivity of the predictive filter. There are also five thresholds (i.e., Quality of Service levels) used. The threshold ranges from easy to achieve (4.0 s), to the server's computational limit (1.0 s). Finally, there are five different inter-arrival rates 0.34 s, 0.4 s, 0.6 s, 0.8 s, 1.0 s (Poisson Arrivals) to represent application workload. These rates range from easy for the server to handle (1 request per second) to the server's limit (3 requests per second). The total number of possible scenarios are 375. For each scenario, five different seeds (i.e., to produce a different set of pseudo-random workloads) are used and averaged for a total of 1875 simulation runs. Each simulation run simulates 100,000 user requests (i.e., each workload consists of 100,000 jobs). In total, several hundred million requests are simulated and the results are presented in Section 6.2.3. The application processing rate was set to 0.33 seconds per request (i.e., this is the rate the server processes jobs).

6.2.3 Simulation Results and Discussion

Tables 6.1, 6.2 and 6.3 show the results of the simulation described in Section 6.2.2. The results are presented and discussed in three subsections, No Controller, On/Off Controller, and Kalman filtered controller. A comparison of the three controllers is discussed in Section 6.2.4.

Figure 6.8 depicts the data in Table 6.1. Notice how the Kalman gain variable is ignored in the table. This is because no controller was used for these 125 scenarios, therefore, the changes in gain have no effect. However, it is clear from the data and graph that when the request inter-arrival time is close to the resource processing rate, many of the requests fail their QoS requirement. Approximately 13 percent of the requests execute within their Quality of Service Requirement when the QoS is set to 1.0 second, and the user workload inter-arrival time approaches the maximum resource processing rate (i.e., 0.33 seconds per request). However, when either the inter-arrival time or threshold increase, the success rate increases.

Table 6.1: No Controller — Base line QoS response times

		Control System: None				
λ	Thresh	Kalman Filter Gain				
		0.5	0.75	1.0	1.25	1.5
0.34	1.0	12933	13039	12808	13525	13109
	1.5	20549	21854	20955	22303	20827
	2.0	25241	27833	28447	26056	27292
	2.5	33224	31532	32960	32922	32950
	4.0	51972	47467	49319	50725	49579
0.4	1.0	58620	58536	58819	58443	58363
	1.5	76872	76995	76593	76889	76155
	2.0	85916	86872	86651	86857	86058
	2.5	92421	92166	92498	92283	92513
	4.0	98545	98546	98439	98762	98665
0.6	1.0	92551	92386	92536	92600	92529
	1.5	98590	98661	98586	98600	98546
	2.0	99710	99728	99705	99705	99724
	2.5	99941	99937	99928	99937	99950
	4.0	100000	99999	100000	99998	99997
0.8	1.0	97577	97553	97525	97631	97590
	1.5	99771	99780	99780	99764	99765
	2.0	99981	99978	99980	99974	99981
	2.5	99998	99999	99998	99998	99996
	4.0	100000	100000	100000	100000	100000
1.0	1.0	98987	98935	98916	98900	98925
	1.5	99936	99937	99933	99946	99938
	2.0	99998	99996	99997	99997	99998
	2.5	99999	100000	99999	100000	100000
	4.0	100000	100000	100000	100000	100000

Figure 6.9 depicts the data in Table 6.2. The on/off controller performed well when

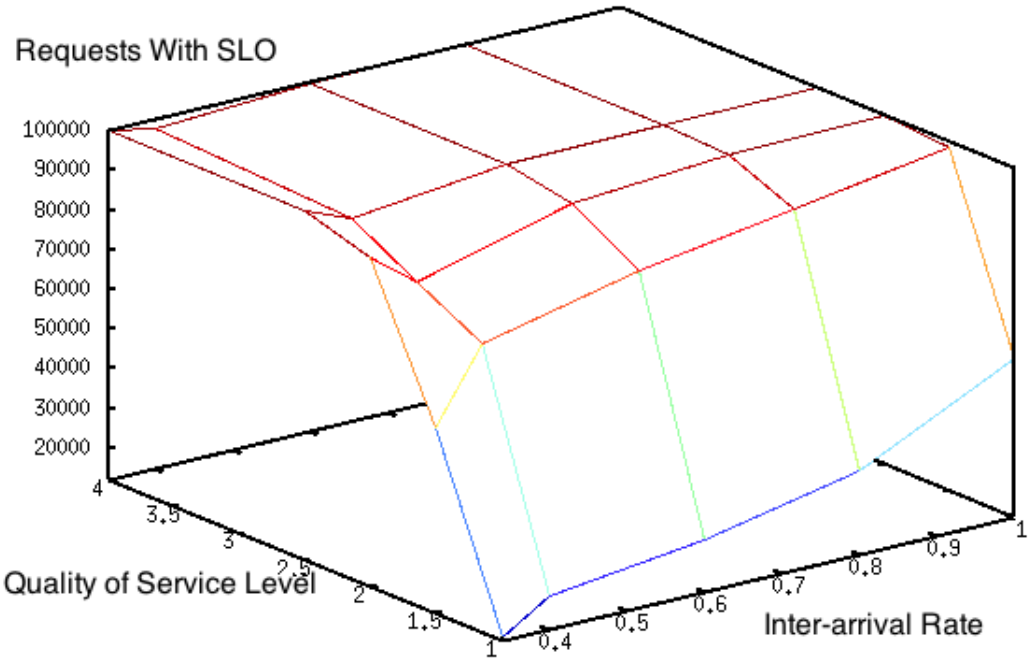

Figure 6.8: No Controller

the requests were pushing the limits of the resource, but did not perform well during light loads. This is due to turning off the controller when a single bad request took too long and having to wait for another reading before being corrected. Effectively, the feedback response no longer represented the true state of the system. Waiting for the resource response had a significant negative effect given this type of controller.

Figure 6.10 depicts the data in Table 6.3. The Kalman filtered controller performed very well and is consistent at all thresholds and inter-arrival rates. In particular, it did well for small inter-arrival times that were pushing the limits of the resource. The filter managed to succeed with approximately 38 percent of the requests.

6.2.4 Discussion

Figure 6.11 is one perspective created from Tables 6.3, 6.1 and 6.2. It is clear from the graph that the Kalman filtered controller performed well when the resource was

Table 6.2: Simple Controller — response times for an ON/OFF controller with no predictive characteristics

Control System: On/Off						
λ	Thresh	Kalman Filter Gain				
		0.5	0.75	1.0	1.25	1.5
0.34	1.0	21333	21542	21540	21460	21517
	1.5	35394	35499	35716	35512	35625
	2.0	43132	42919	43030	43345	43085
	2.5	52685	52360	52317	52434	52397
	4.0	65475	65143	65384	65199	65193
0.4	1.0	29159	29013	29168	29114	29227
	1.5	48179	47778	48022	47973	47958
	2.0	59567	59954	59499	59569	59405
	2.5	70417	70591	70882	70540	70731
	4.0	87678	87809	88172	87340	87613
0.6	1.0	54342	54458	54262	54280	54229
	1.5	79952	79627	79732	79714	79678
	2.0	92979	93113	92867	93100	92903
	2.5	97521	97401	97612	97530	97494
	4.0	99967	99967	99944	99938	99972
0.8	1.0	71981	71747	71635	71890	71959
	1.5	92630	92580	92552	92668	92663
	2.0	98745	98833	98833	98857	98816
	2.5	99781	99786	99790	99764	99782
	4.0	99998	100000	100000	99999	100000
1.0	1.0	82001	82284	81950	82210	82198
	1.5	96993	97016	97062	97029	97054
	2.0	99728	99735	99713	99742	99740
	2.5	99970	99970	99973	99967	99967
	4.0	100000	100000	100000	100000	100000

under stress and processed approximately 38 percent of user requests within the QoS requirement. This is compared to no controller which processed approximately 13 percent within QoS and the on/off controller that processed approximately 21 percent when under stress. However, it is clear that the no controller scenarios perform better when the system is lightly loaded and the resource can easily meet all QoS demands. In the ECS-X1 Cloud scenario, if the system is under stress, this would have resulted in using the overflow cloud much less by using the Kalman filter to determine which workload should be redirected to the overflow cloud. This could save ECS-X1 Cloud approximately 26% of the work load costs the overflow cloud would charge.

There are other control paradigms and scenarios that can be tested. To evaluate the response for an application with other erratic behaviors, or to evaluate these control solutions with dynamically allocated resources. Another interesting evaluation would be to use the data gathered in this simulation and apply a self-adaptive strategy to select among several controllers based on environmental state (i.e., request rate). For example, if the adaptive controllers detect that a threshold has been violated,

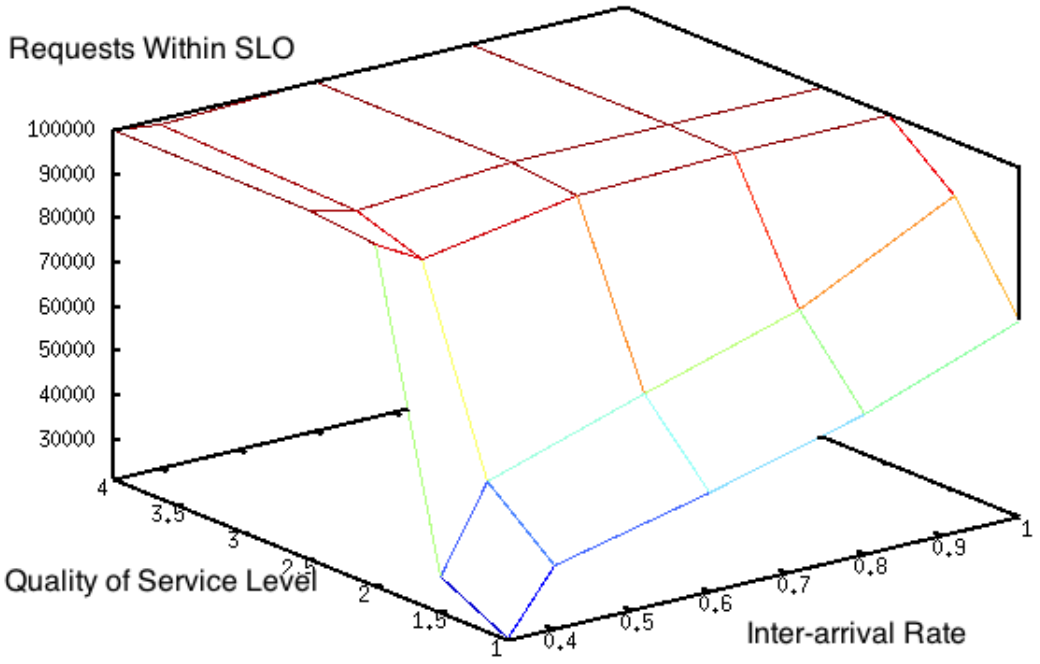

Figure 6.9: Simple Controller - an ON/OFF controller

then self-adaptation must take over and make a change. In the case of the ECS-X1 Cloud scenario, the controller at a specified threshold could activate the Kalman filter to start directing workload to the overflow cloud to ensure adherence to the SLAs and to reduce the amount of reliance on the overflow cloud.

In this chapter, we presented a comparison of the Kalman filtered controller to improve the Quality of Service achievable rate of a cloud hosted application. We performed the evaluation using the OMNET simulation environment.

From the data collected, it is clear that the Kalman filtered controller provides improved adaptability compared to no controller and the on/off controller for situations in which the resource is near maximum throughput. However, under less resource loaded conditions, no controller with no adaptation performed as well as Kalman, but with poor performance at peak loads. This is because the Kalman prediction is worse than the statistical probability that the workload density causes a violation in the SLA. It is clear from these results that a decision point is necessary if the Kalman filter is to be used. This would require server utilization feedback and knowledge of

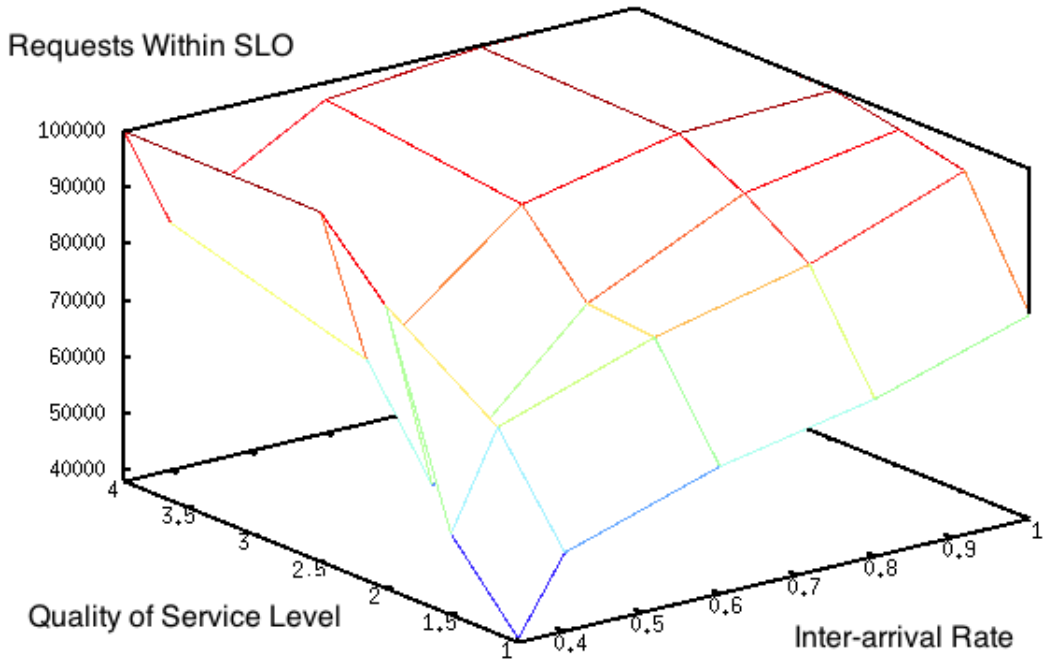

Figure 6.10: Kalman Filtered Controller - a predictive controller

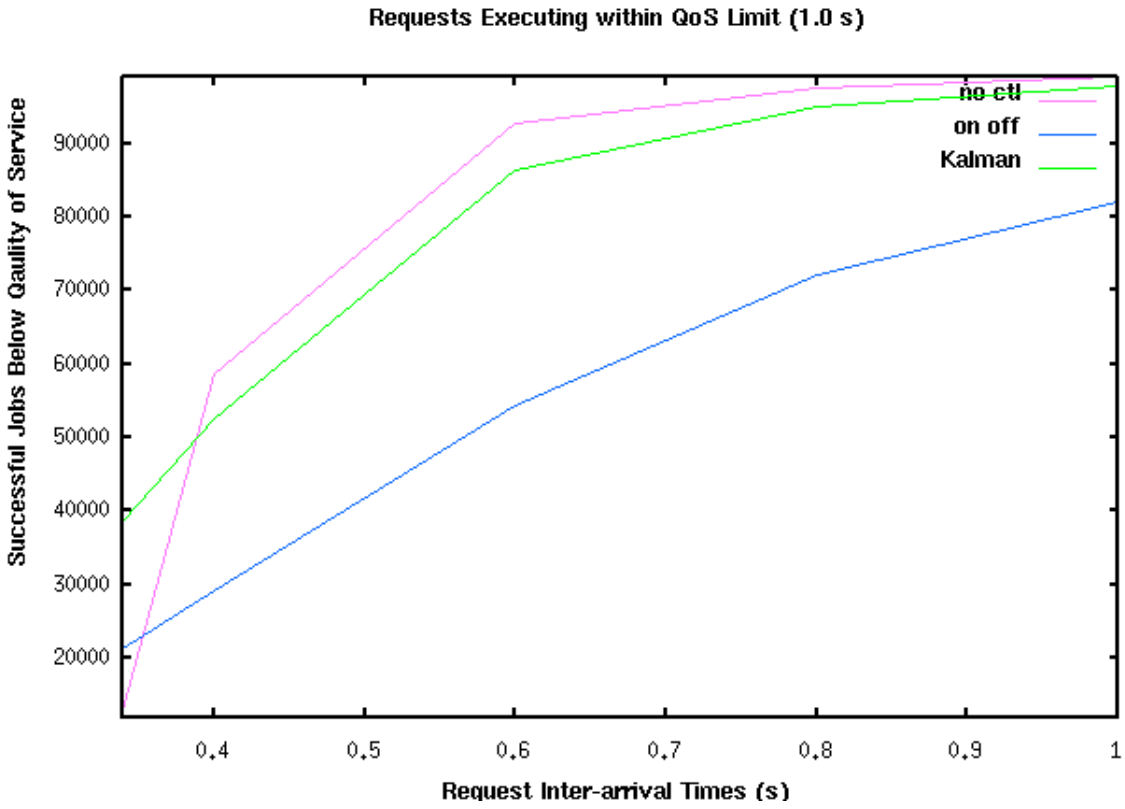

Figure 6.11: Controller Comparison

Table 6.3: Kalman Filtered Controller — response times using a predictive controller

		Control System: Kalman Filtered				
λ	Thresh	Kalman Filter Gain				
		0.5	0.75	1.0	1.25	1.5
0.34	1.0	38605	38807	38159	38300	38202
	1.5	51754	52991	52117	51198	50863
	2.0	60609	61067	60143	59717	59198
	2.5	65634	66829	66263	65916	65303
	4.0	74263	76586	77559	77345	76984
0.4	1.0	52418	52359	51717	51181	50327
	1.5	69237	68828	68675	67630	66061
	2.0	78558	78962	78280	77096	75785
	2.5	84981	85569	85176	83837	82552
	4.0	95010	95001	95012	94267	93223
0.6	1.0	86316	85859	85893	85437	84287
	1.5	96411	96345	96111	95827	95356
	2.0	99048	99088	99032	98861	98614
	2.5	99786	99796	99757	99674	99578
	4.0	99998	99996	99997	99996	99975
0.8	1.0	95057	95016	94973	94854	94671
	1.5	99369	99384	99366	99340	99222
	2.0	99930	99928	99924	99930	99886
	2.5	99984	99995	99983	99986	99988
	4.0	100000	100000	100000	100000	100000
1.0	1.0	97642	97679	97576	97559	97580
	1.5	99821	99845	99818	99819	99800
	2.0	99980	99984	99988	99987	99978
	2.5	99999	99999	99999	99998	99998
	4.0	100000	100000	100000	100000	100000

the current workload density. The manager could then swap in and out the Kalman filter as needed. This would provide an improvement to minimizing third part costs.

6.3 Stochastic Open Loop Control

In this section we explore a stochastic open loop control model to manage an admission control system. Figure 6.12 depicts the model used in this section. In this scenario, the broker negotiates with a cheaper resource for a computational time slot. The broker's clients use this time slot to execute their jobs. If the clients submit too much work, then the server will not execute it and the server will fill the time slot with work negotiated from other brokers. If the client's work is rejected, the clients can use an overflow queue to submit their jobs to other more expensive resources. The artifacts depicted in Figure 6.12 are described as follows:

- Client — are client applications producing work load.

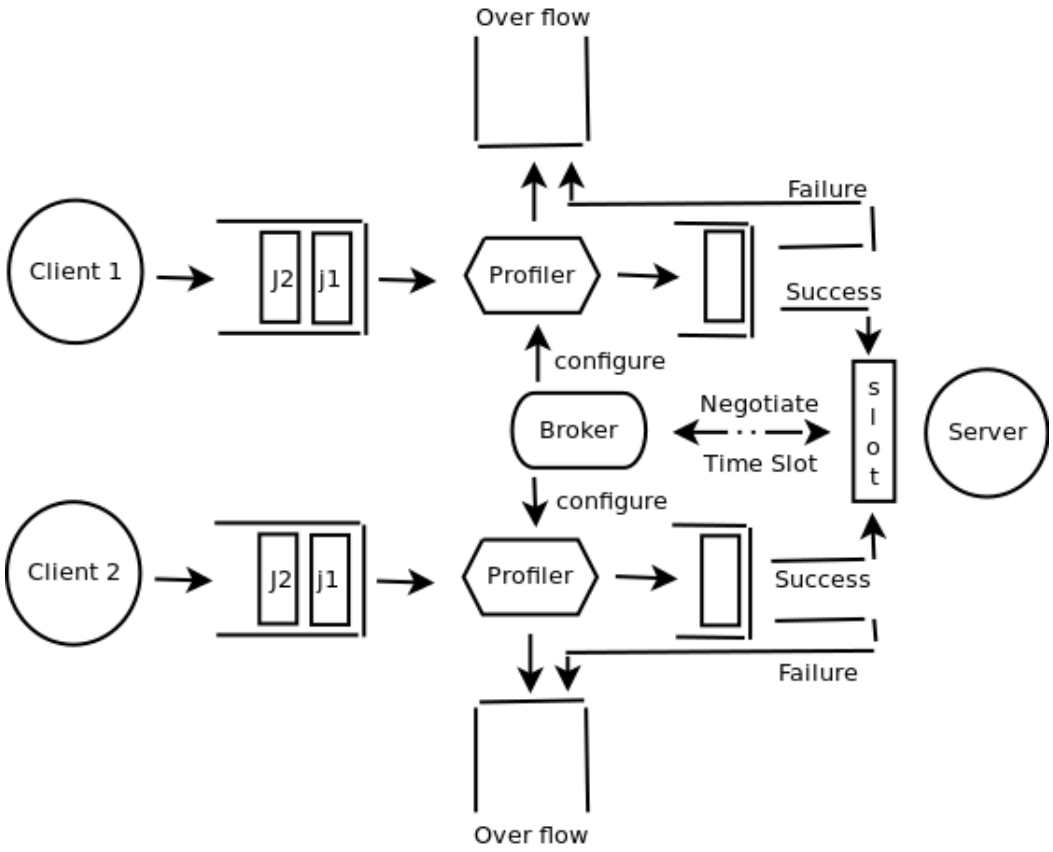

Figure 6.12: Brokerage Negotiation of a Computational Time Slot for Two Client Applications

- Broker — negotiates a time slot for it's clients to access, and configures the client's Profiler according to probability models.
- Time Slot — is a slot of time the server can use to execute client jobs.
- Profiler — is configured by the broker to determine which jobs may benefit from the negotiated time slot offered by the server.
- Overflow — jobs directed by the profiler or jobs rejected by the server are placed in the overflow queue to be executed by other resources.

In this scenario, a client job is assumed to require the entire negotiated time slot to execute. Therefore, for each scheduling cycle only one of the broker's clients will have their job executed by the resource. We also assume that for every scheduling cycle, each client will submit a job with probability p and will not submit a job with

probability $1-p$. If two of more clients submit jobs during the same scheduling cycle, then no jobs will be executed by the server. Then the probability of success (S) is described by Equation 6.20 where k is the number of broker clients.

$$S = k \times p(1-p)^{k-1} \tag{6.20}$$

The objective of the broker is to configure the profiler of each client. The profiler is in effect a filter. Client jobs that pass through the filter get sent to the server, otherwise they get sent directly to the overflow cloud. The filter determines if a job fits into the server's time slot and if it can execute within a certain number of scheduling cycles without violating its execution deadline. It is the job of the broker to set the number of scheduling cycles.

The broker uses a stochastic model and a probability threshold value to determine how many scheduling cycles the profiler's filter should be set to. The stochastic model is generated using two variables, the number of clients k and the probability a client will submit a job during a scheduling cycle p.

6.3.1 The Stochastic Model

To create a model, we first need to articulate the problem. *What is the probability that each client can successfully execute their job by scheduling cycle n? What is the probability that the last client can successfully execute its job on the n^{th} scheduling cycle?*. The answers to these questions allow the broker to setup the profiler's filter properly.

To create the stochastic model to answer our questions, we create a probability tree. Figure 6.13 maps the first four scheduling cycles for three clients ($k = 3$) exhibiting the different possible paths of success (i.e., only one client submitted a job) and failure (i.e., two or more clients submitted jobs during the same scheduling cycle). For example, as depicted in Figure 6.13, at scheduling cycle four (i.e., the bottom row of the tree) a possible path is if there are four successes (i.e., meaning

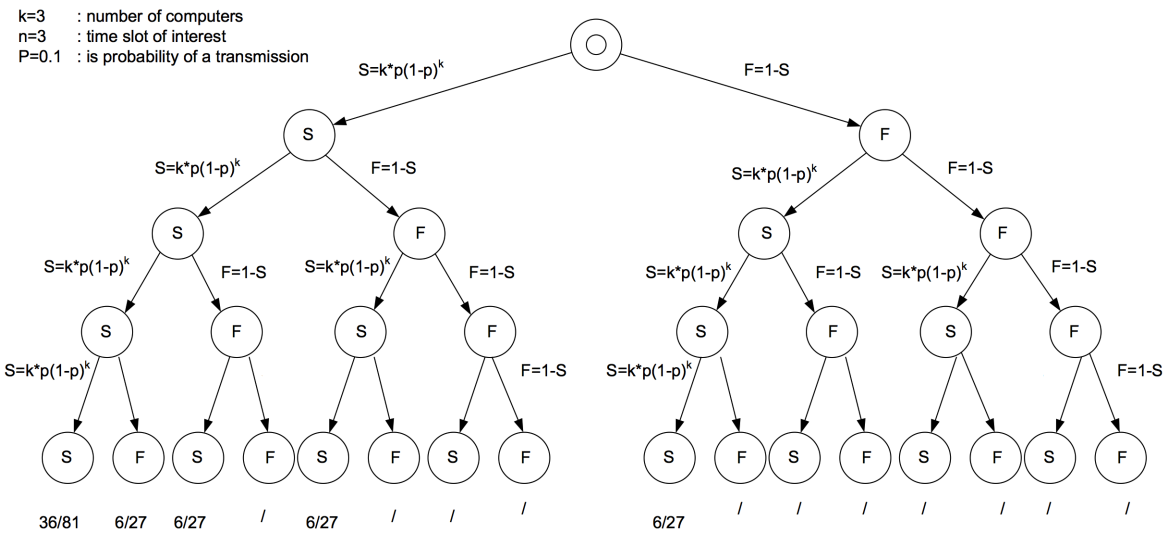

Figure 6.13: Decision Tree Representing Probability only One of Three Clients will use the Time Slot per Scheduling Cycle

that 1 of k clients successfully executes a job at scheduling cycles 1, 2 ,3 and 4). The probability of this case is S^4, and there are $k^4 = 81$ possible combinations of success. A good combination of success could be A, B, C, C, meaning that client A succeeded in scheduling cycle 1, client B succeeded in scheduling cycle 2, client C succeeded in scheduling cycle 3 and client C again succeeded in scheduling cycle 4.). This is a valid path because each of the three clients succeeded at least once. A bad combination could be A, A, B, B, which is bad because client C did not success in having its job executed. In this example, for 4 scheduling cycles and 3 clients, there are 36 good combinations out of 81. Therefore the success probability, given 4 successes in a row, is $S^4 \times \frac{36}{81}$.

Unfortunately the probability tree depicted in Figure 6.13 does not answer the articulated questions. To answer these questions, we need to combine the probability Equation 6.20 with counting theory to determine the number of successful paths. Equation 6.21 describes the solution to the first problem; *what is the probability that each client can successfully execute its job by scheduling cycle n?*

$$P\{X \leq n\} = \sum_{i=k}^{n} \binom{n}{i} S^i (1-S)^{n-i} \frac{\left[\sum_{t=0}^{k} (-1)^t \binom{k}{k-t} (k-t)^i\right]}{k^i} \tag{6.21}$$

In Equation 6.21 $\sum_{i=k}^{n}$ adds all paths with i clients, note that i starts at k which is the minimum number of successes so that there are at least k successful job executions. $\binom{n}{i}$ is the number of paths that have i successes. $S^i(1-S)^{n-i}$ is the probability that a path has i successes and $n-i$ failures. $\sum_{t=0}^{k}(-1)^t\binom{k}{k-t}(k-t)^i$ is the key to the solution. It is a sterling number of the second kind. It is used to determine the number of possible combinations if there are n identical positions of k distinct objects such that at least one of each distinct object must be present. k^i is the number of possible combinations for a specified path.

Equation 6.22 describes the solution to the second problem; *what is the probability that the last client successfully executes its job on the n^{th} scheduling cycle?* Interestingly, this problem is an extension of the first problem. Equation 6.22 is extended from Equation 6.21, these equations represent the probability mass function (PMF) and the cumulative distribution function (CDF), respectively.

$$P\{X = n\} = \sum_{i=k}^{n} \binom{n-1}{i-1} S^i (1-S)^{n-i} \frac{\left[\sum_{t=0}^{k-1} (-1)^t \binom{k-1}{k-t-1} (k-t-1)^{i-1}\right] \binom{k}{k-1}}{k^i} \tag{6.22}$$

Figure 6.14 depicts the CDF and the PMF of Equations 6.21 and 6.22 for three clients ($k = 3$) and four different probabilities (p) of client job submission. From the CDF, it is clear that with $p = 33\%$ chance of client job submission, and at approximately 7-8 scheduling cycles (i.e., time slots on Figure 6.14), there is a 50% chance that all three clients will successfully execute their jobs on the server and avoid using the overflow. The PMF graph indicates the highest probability that the last of k clients will success at scheduling cycle 7 or 8.

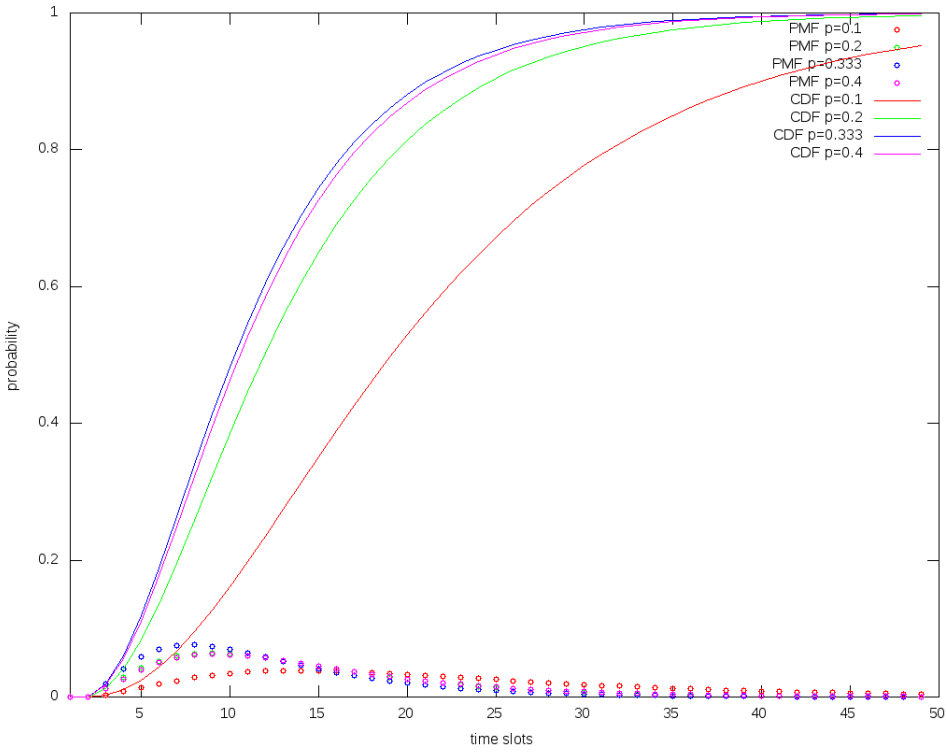

Figure 6.14: PMF and CDF of Three Clients Access of a Single Resource

The broker uses Equations 6.21 and 6.22 along with a threshold value (i.e., 50%) to set the client's profiler filter. In this example, if the broker has three clients, and the clients require at least a 50% probability of success on the server, then the broker will set the profiler's filter to 7 scheduling cycles. The filter will screen the client's jobs with a deadline less than 7 scheduling cycles and select it to execute on the server.

6.3.2 Stochastic Discussion

The brokerage model for open loop control as depicted in Figure 6.12 along with our novel stochastic models (cf. Equations 6.22 and 6.21) to control the client's profiler adaptively constitutes an effective approach to manage client applications.

6.4 Summary

In this chapter, we explained how to use control theory to manage computing systems as recommended by Murray [Mur03] for further research. We identified several issues such as lack of good feedback signals especially with cloud applications in which the current resource provisioning is dynamically changing. We proposed a control approach based on Kalman filters to address this issue. We showed through simulation that such techniques can increase adherence to service level agreements by predicting application behavior over time. In the ECS-X1 Cloud scenario, using such a system improved its QoS objectives. Contributions include the application of the Kalman filter for cloud application workflow management. We also presented a stochastic open loop control model for brokers and SaaS providers to manage their clients. Finally, we presented a novel stochastic model (cf. Equations 6.22 and 6.21) to capture the controller's requirements.

Chapter 7

Yakkit

A key research question derived from Chapter 1 is 'How can we support cloud applications distributed over several clouds?'. To answer this, we propose a new approach to building localized, context-driven social networking applications to allow people to communicate, interact, collaborate, and socialize in a truly innovative manner. In particular, the underlying infrastructure provides mechanisms to form communities of people who do not necessarily know each other, but are in close proximity to each other. In other words, it allows a user to communicate with unfamiliar people with whom the user does not know how to get in touch with.

To achieve these goals we designed and implemented an infrastructure and an application using proximity based structures hashed over a set of clouds to distribute the user load effectively. Our infrastructure, called *iCon*, provides layers of APIs to separate concerns and support the key features of our approach. To evaluate this layered architecture we developed *Yakkit*, an innovative social networking application. This locality based end-user application, which runs on workstations as well as mobile phones, allows users to identify persons who are in close proximity and interact with them. The premise is that people are tied to the places and time periods of their life's experiences. Communicating in such a way is not only novel but also presents a wide range of capabilities and opportunities for smart web applications.

In particular, we designed a layered architecture to provide seamless integration and interoperability among heterogeneous web applications. The key question addressed is how the various distributed services can be orchestrated together effectively and intelligently. The problem is exacerbated when many different vendors or

cloud sites, which support some subset of the services, are being orchestrated. Moreover, user devices have access to much contextual and environmental information. This along with the web's vast data store of user profiles and user habits provides a rich environment for research into integrating all these capabilities and information sources so as to provide a shared user experience for all users while still using resources efficiently and meeting real user needs effectively.

The focus of this chapter is how to support and manage localized services across various cloud sites using a distributed hash table. The key service explored is the locality service which provides applications with the ability to track how users are positioned relative to each other. This service is supported through a distributed hash data structure. The Yakkit application was originally intended as a proof of concept for the underlying infrastructure and services but turned out to be much more and is now a impressive and innovative social networking application.

The remainder of this section describes the design and implementation of *Yakkit,* and its infrastructure including *iCon* (iCon—inter Cloud overlay network), *AII* (Application Indirection Interface), and *MII* (Management Indirection Interface). The chapter concludes with related work on locality based social networking applications and a discussion on the privacy and security issues encountered in such an open social network.

7.1 Locality Problems and Solutions

From an application perspective, there are many locality based messaging applications available. *Aka-Aki*[1] facilitates information exchange with others near you; *Askaround*[2] lets you join conversations taking place near you; *Askalo*[3] allows you to ask questions about the city you are in; *Badoo*[4] facilitates communication with people near you; and *Block Chalk*[5] which helps neighbors stay connected with part of a neighborhood. All these have a common theme of using locality to determine

[1]http://www.crunchbase.com/company/akaaki
[2]http://www.intomobile.com/2011/03/03/ask-around-location-app-iphone/
[3]http://www.askalo.ca/
[4]http://badoo.com/
[5]http://blockchalk.com/

groups dynamically. Yakkit is similar in this way, but adds the temporal feature of communicating either backward or forward in time, and adds a feature that allows you to appear to be in several places at once. These two extra features make Yakkit truly innovative.

From a framework perspective, there are several solid supportive frameworks including *OpenSocial*, *SNS (Social Network Services)*, and *SMSF (Social Media Service Framework)*. The OpenSocial framework, developed by Google and MySpace, would be ideal for Yakkit to use. It provides all of the necessary capabilities to integrate other social data, such as locality, with Yakkit's application data. However, one of the key ideas behind Yakkit is to apply structure to data across clouds. It is unclear if the OpenSocial framework supports this requirement. OpenSocial sites are completely separate from other OpenSocial sites. To distribute data over the sites would either require support at the site, or a data abstraction over the sites. The SMSF framework, which extends the SNS framework, provides insight into how to use distributed hash tables to support social networked services in an intelligent way. It is clear that a merge between the SMSF and the OpenSocial framework provides the necessary capabilities to support the next generation smart web applications. This is the space within which iCon is currently being developed to support these requirements.

There are many other interesting tools emerging to take advantage of context. A fairly new and exciting one is the Google+ project with its Circles feature. Circles allow users to group friends based on topic so users can filter content. The contents on a homepage appears in a similar fashion to what one sees on Facebook. Users can post text, pictures, and video and direct the information to selected Circles. Similarly, users see what others are posting and can filter that content by selecting a particular Circle. Thus Google+ and Yakkit both drive towards smart social networking.

7.1.1 iCon Locality Data Structure

The iCon overlay itself is a registry and deployment infrastructure in which services can be made available through the overlay. In this case iCon provides the *locality service.* The locality service design itself is a *quad tree* data structure as presented

in Figure 7.1 [FB74]. The idea is to store objects in buckets until the bucket is too full and then sub-divide. The locality based service designed for deployment in iCon follows a similar approach except that buckets are forced to divide even with a single object until a desired depth is reached in order to achieve a desired resolution. The two dimensions of the quad tree correspond to the latitude and longitude axes in cartography. The deeper one goes in the tree, the more granular the location becomes (e.g., zooming in when exploring with Google Earth). In the context of this dissertation, for this research it is paramount to identify which services the Orchestration Layer of the ACRA model requires to mange the iCon overlay when dispersed over many cloud providers.

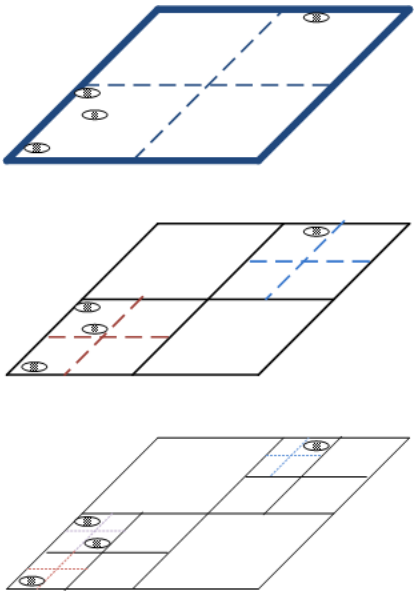

Figure 7.1: Simple quad tree structure to allocate users [FB74]

During the design of iCon and Yakkit we investigated and assessed several related data structures including *cell* and *kd-trees* [Ben75], *locality based hashing* [DIIM04], and *cover trees* [KR02]. Kd-trees and cover trees often require a selection of a pivot point in order to guide the growth of the tree. Moreover, locality based hashing is really designed for problems where there are several dimensions involved. These are serious issues for modern social tools in which the churn of users is high and the users are continuously in motion. These trees would have to be rebuilt continually with good pivot points or user positions selected as the root. In the end we selected the

quad tree data structure for its simplicity, strong support for dynamic change, and the key feature of simply using bins to store users. To increase granularity, we simply allow the quad tree to be divided further by increasing the maximum depth. Having selected the quad tree for ease of implementation and for its static nature, there are some drawbacks as well.

The quad tree has a maximum depth depending on the map coverage area and how fine grained the system needs to group users into bins. For example, if the map area (i.e., *map_area*) covers 100 m^2 and the system requires a granularity (i.e., *des_granularity*) of 6.25 meters, then the tree depth (d) needs to be three (cf. Equation 7.1). Therefore, when a user is added to the locality service, at least d checks must be performed to find the appropriate bin for this user. If a bin does not exist, it needs to be created. Bin creation and deletion are used to limit the amount of memory needed to store all the tree nodes in use. If a tree node has no user keys, it is simply deleted along with all of its children.

$$d = \log_2 \left\lceil \frac{map\_area}{des\_granularity} \right\rceil - 1 \tag{7.1}$$

The depth of the quad tree is the reason for additional service operations such as *adduser, find coverage, moveuser,* and *removeuser*. The *adduser* operation is expensive if there are no child nodes since they need to be created. In the worst case, add user requires $d \times a \times 16$ operations, where d is the maximum tree depth, a is the time to allocate memory for a new node, and 16 is the number of comparisons required to determine in which quadrant a new user is to be inserted (i.e., insert child node). The best case scenario is that all nodes are already created and then the user key is simply added to each node as it progresses down the quad tree until it finds its proper bin (i.e., $d \times 16$). The *coverage* operation, which determines if the desired coverage radius completely covers a quadrant at each level, performs one key operation per quadrant. If it does completely cover a quadrant, then all user keys registered in a quadrant node are included in the final coverage set. If not, another coverage operation must be performed at the next lower level. At each lower level, it is more likely that all quadrants are covered. The coverage search can become a bottleneck under certain conditions. Fortunately, the depth level d is bounded by the desire for granularity, and in reality it is not a large number (i.e., 24 to cover

the entire planet). Unfortunately, the worst case for the number of coverage searches is $O(2^d)$. This worst case is unlikely to occur in practice, but possible in very high density situations. Coverage operations in areas of high density using a quad tree are expensive. The *moveuser* function is efficient, since one simply moves up the tree and then back down again. Thus, the trade-off here is between moving users around the tree structure and determining coverage. Fortunately, the time complexity of the *coverage* operation is bounded by the tree depth, while the computation for moving users is dependent on the number of users registered in the system. This time complexity analysis is useful for designing an effective controller and scheduling system.

7.1.2 Distribution

The quad tree is implemented as a *distributed hash table (DHT)* as depicted in Figure 7.2. The hash ring supports distributed locality searching by having each node at a specific level maintain all those keys between it and its peer nodes at the same level with the same type. For example, the quad tree is divided into four quadrants A, B, C, and D at level four. However, due to the load, quadrant B at level 4 is replicated so that any children—which would be at level 5—of B are divided between the two B nodes. The address based routing tables have to be updated for the case of a search in which both nodes need to be checked. The benefit is that the coverage operation is spread among computers and thus sub coverage searches should be efficient. This leads to a key scheduling question: at which level and how many extra peer nodes should there be to optimize the coverage operation? The challenging part is to determine a good heuristic to find a possible solution. The hash ring DHT for locality provides the necessary functionality for the Yakkit application.

The idea is to spread the user objects uniformly over several cloud provider systems, as shown in Figure 7.3, and provide a cross-cloud routing structure to find and locate objects using keys. The three main operations needed and provided by the DHT structure are *coverage, adduser,* and *addnode*. Coverage is managed with a set of routing tables for each node in the quad tree. These routing tables point to peer nodes on different machines which may have user keys included in the coverage operation. The *adduser* and *addnode* operations use another set of routing tables to help locate keys on the hash ring. This setup is similar to the CHORD [SMK+01],

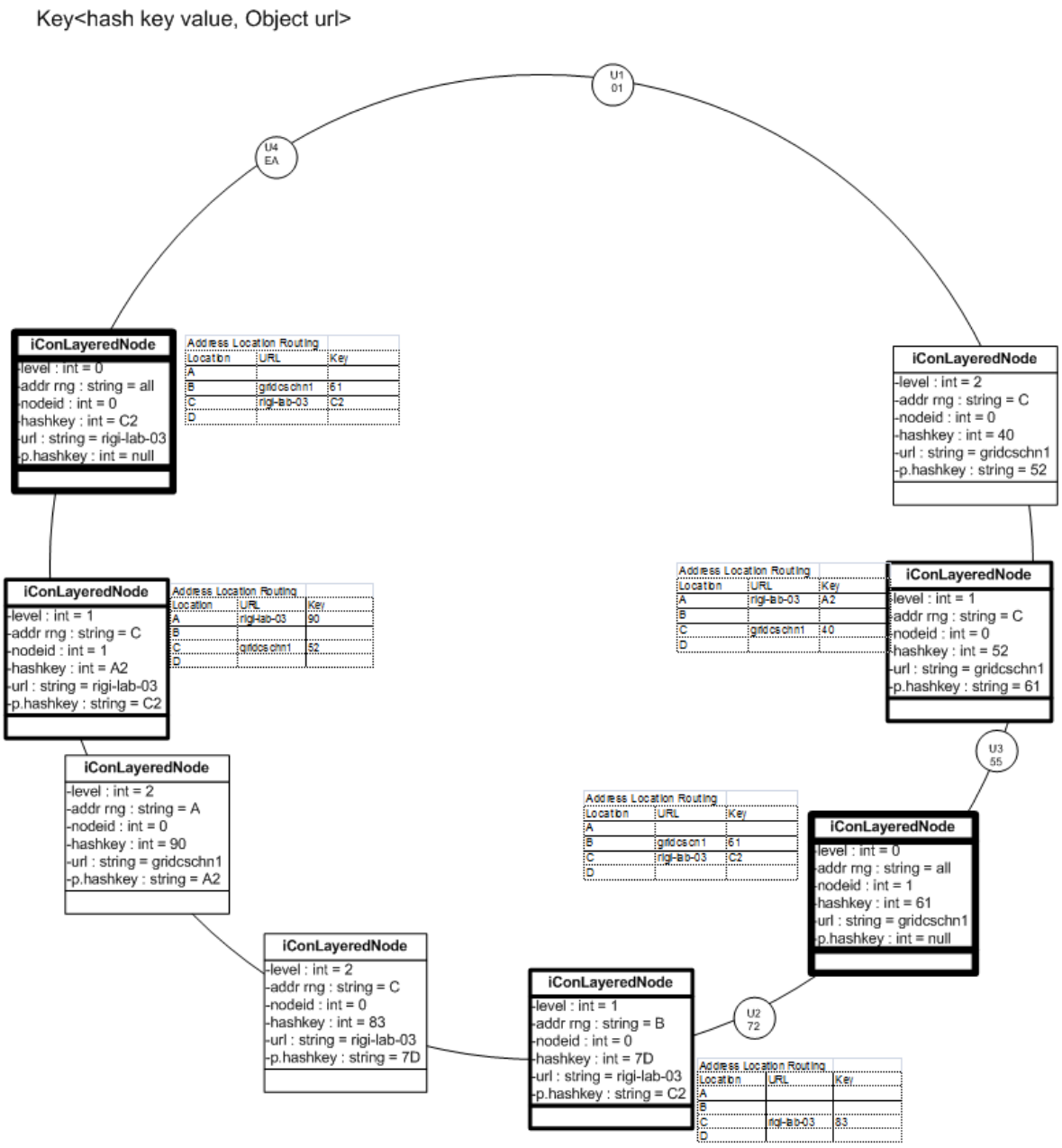

Figure 7.2: iCon routable hash ring

CAN [RFH+01], or RON [ABKM01] lookup services.

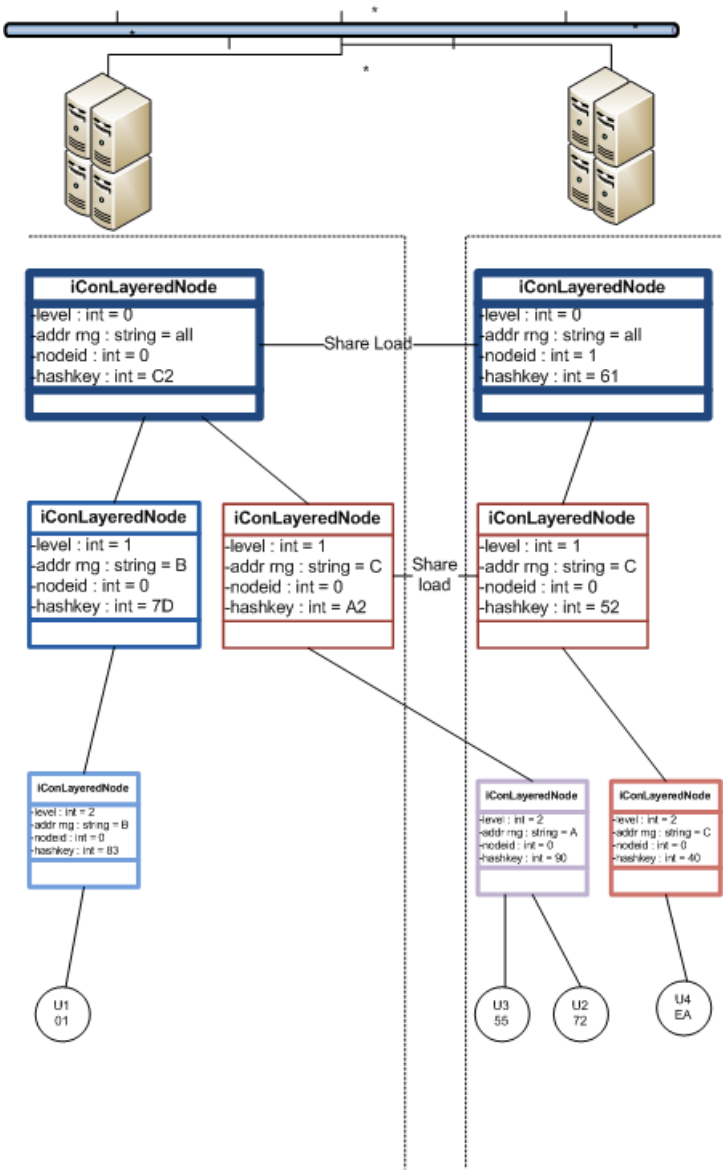

Figure 7.3: iCon network deployment example

7.2 iCon: Inter-Cloud Overlay Network

Applications use iCon to register or instantiate services provided by iCon (i.e., new services can be registered for deployment and management by iCon). Figure 7.4 illustrates the design of iCon. The primary interface to iCon is via the iCon servlet class. This class registers services and provides a way to create third party services. The storage and search of services is provided by our DHT data structure.

For the Yakkit application two services are registered in iCon: *locality service* and *user management service*. However, other services such as *messaging* and *orchestration services* are also available through the iCon infrastructure. The locality and user management services provide the following functionality via iCon to the AII (Application Indirection Interface) (i.e., a Javascript file called *aii.js* used by the Yakkit web application):

- *adduser* — allows a user active object or database object to be stored somewhere in the DHT (i.e., user objects should be evenly distributed over all database instances that support the iCon overlay).
- *moveuser* — allows a user to set its position geographically;
- *getcover* — allows a user to find other users/registrants in close proximity to the user's position;
- *deleteuser* — removes a user from the overlay.
- *sendmsg* — sends text/audio/video/data to users within a coverage area.

To allow for management capabilities, such as scheduling and control, iCon supports the following functions to the Management Indirection Interface (MII). The MII consists of two parts. (1) *Scheduling* deals with efficiency and optimization (i.e., what is the minimum amount of servers/services required to support the current user load). (2) *Control* deals with performance and feedback from the system under management (i.e., resources or services that are not performing as expected). The scheduling interface includes the following operations:

- *addservernode* — adds another node to the iCon overlay.
- *addservice* — adds another instance of a service onto one of the overlay nodes.
- *moveservice* — may find a closer location to users where this service is being invoked from.
- *moveservernode* — may have found cheaper hardware to deploy server node.
- *removeservernode* — removes server node if system load does not require it.

- *removeservice* — removes service if it is obsolete or being replaced.
- *getinfo* — collects statistics or sensor data from the system to make informed scheduling decisions.

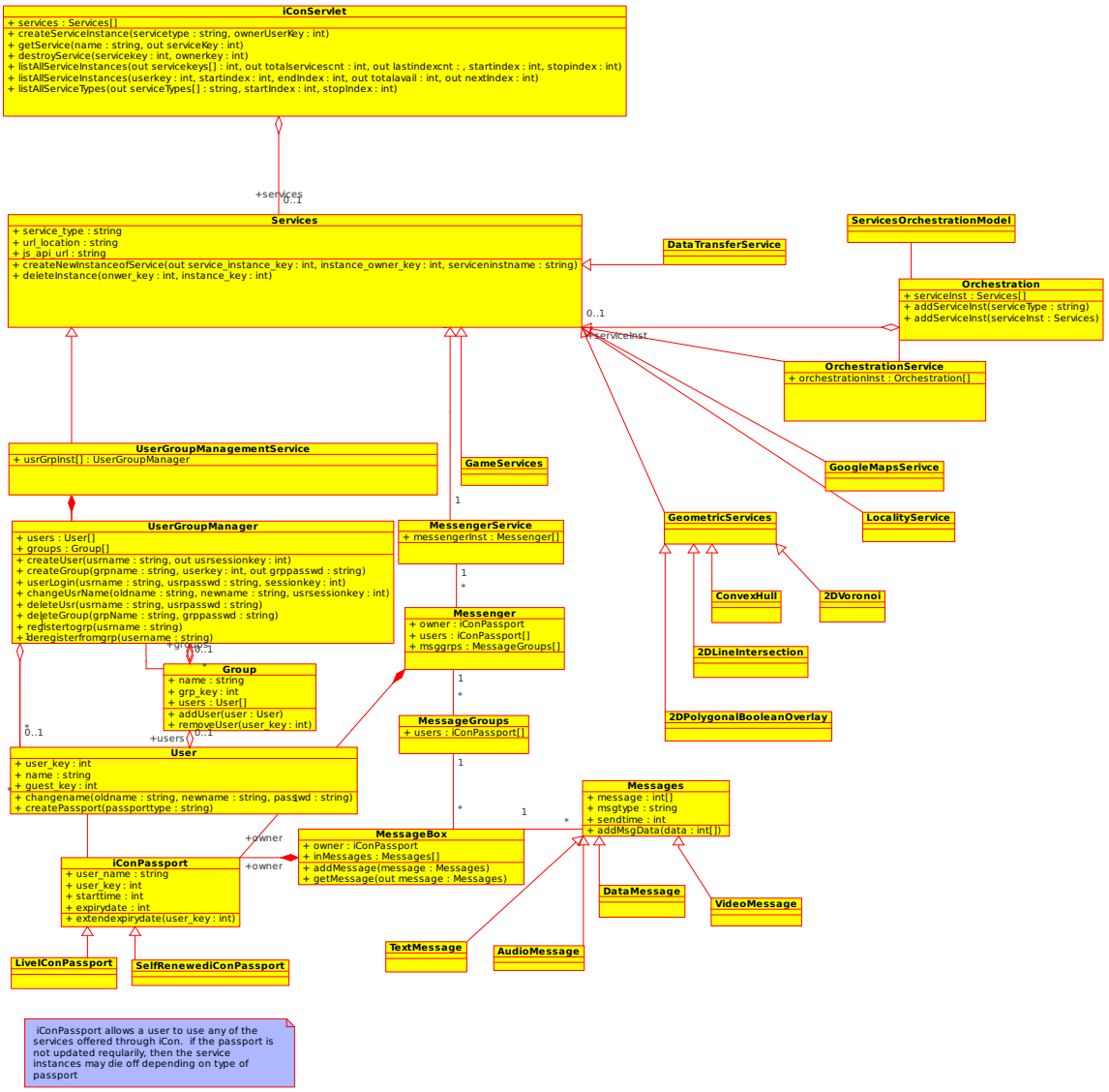

Figure 7.4: iCon Class Diagram

The control interface includes the following operations:

- *changeRoutingTable* — to bypass a server node that does not respond or has stalled.

- *shutdownServerNode* — to shut down a server node if a node causes problems or does not perform as expected.
- *rebootServerNode* — to reboot a server node, if a node misbehaves.
- *moveService* — to move a service to another node if it does not perform well.
- *shutdownService* — to shut down a service if it does not run well anywhere.
- *restartService* — to re-initialize a service if it does not perform as expected.

These two services provide the functionality that the Application API and the Management API use.

7.3 iCon Architecture

We designed the iCon infrastructure as a three layer architecture, as depicted in Figure 7.5, to separate concerns and to facilitate the construction of smart social networking applications such as Yakkit.

- Layer 0 — The iCon (inter Cloud overlay network) API supports access and management of distributed services such as orchestration, storage, persistence, distribution, sharing, locality, and messaging. iCon is an overlay within which web applications and services communicate with each other. This overlay allows for services to be deployed as a multi-tiered routable DHT in which users, web applications, or service instances of the system have their active objects stored.
- Layer 1 — This layer consists of two APIs: the Application Indirection Interface (AII) and the Management Indirection Interface (MII). AII provides mechanisms to join or use the underlying iCon overlay. AII's purpose is to simplify client application design by providing a rich API (i.e., similar to the Google maps API). This may be a client or server side Javascript, or a Java Servlet, or a set of Java or C/C++ packages which provide access to the configurable options of the iCon overlay. For example, AII may provide a way to create a new locality service or to access a web application that already uses the iCon Overlay. MII provides mechanisms to schedule and control resources and services which are currently supported by the iCon DHT overlay. For example,

MII may support an interface which can add another service instance to be deployed and managed by iCon.

- Layer 2 — Finally, sophisticated social networking applications, such as Yakkit, can be implemented effectively and elegantly by using the iCon, MII and AII APIs.

The remainder of this section describes the design and implementation of the three infrastructure layers, iCon, AII, and MII as well as the Yakkit application itself.

7.3.1 AII: Application Indirection Interface

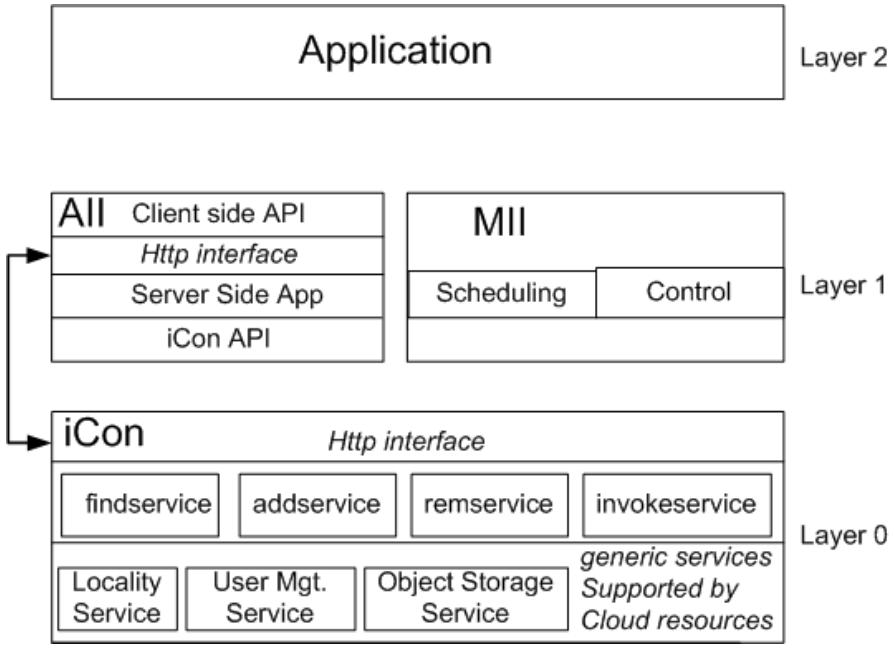

Figure 7.5: Three Layer Architecture to Compose Next Generation Social Application

This API has knowledge of key servers that provide a boot strapping process into the iCon overlay. It also manages the information exchange between the overlay HTTP interface and the API. JSON is the data exchange format between the client side API and the iCon overlay. Key functions of this API include:

- a set of functions for the client application to use. These functions provide access and registration into the iCon overlay and are configurable on how to use iCon;

- the ability to configure several types of network architectures, that is, the client application may wish to set up a peer to peer server assisted architecture; or a peer to peer architecture using iCon as a registry only; or a direct proxy to the iCon overlay; or not use iCon at all; and
- the ability to create service instances to be managed by iCon.

7.3.2 MII: Management Indirection Interface

There are two primary goals for MII. The first goal is to be efficient to limit the use of expensive services when cheaper ones are available. This cost-conscious approach is achieved through scheduling (assigning) resources and services to iCon and benefits users because iCon is more responsive to user requests and changes in the environment. The second goal is to be resilient to performance and environment changes so that the system can adapt when some parts or aspects of the system are not performing as expected (e.g., a server turned off or suffering from hardware failure; a machine has bad RAM; or a router is dropping packets). Performance issues are addressed through controllers designed to compare what is expected from the system to what is actually happening in the system. In effect, the system has selected self-healing capabilities and thus can deal with removal of bad hardware/software from the overlay and then prevent the scheduling system from using the bad hardware or software.

It is important to note that both scheduling and control can be applied at several levels. The first level is directly in the controls of the application. At this level, the control and scheduling systems perform tasks that directly benefit or hinder the application under management. The second level has an altruistic agenda which attempts to benefit iCon itself in order to execute as efficiently as possible and still maintain all of the different web applications requirements. Section 7.3.3 discusses both of these perspectives.

7.3.3 MII: Application scheduling and control

This type of scheduler and controller deals specifically with the iCon services required to keep the web application performing well. For example, if there are too many users, maybe another peer node is needed in the locality tree to help distribute user load;

or if the bandwidth requirement is increasing due to a shift in traffic from text to voice messages, then more bandwidth is be allocated to the object location service. These types of scheduling questions can be formulated as *Knapsack* or *Cutting Stock* optimization problems to compute optimal answers to these questions.

To re-iterate, at the application level control and scheduling are dedicated to an application or possibly a set of applications. For the Yakkit application, there are issues if too many users are clustered. In this case the performance can be adjusted by limiting the depth of the quad tree which is the responsibility of the scheduler. However, the scheduling system could also realize that more physical resources are needed to subdivide the coverage search and thus the scheduling system may request more resources. The controller on the other hand would have a concept of the scheduling decisions and the capability of the services and resources that the scheduler uses. If the scheduling system does not alleviate the computational load, then it is up to the controller to shut down the newly booted resource, mark it as inoperable and request that the scheduler boot another or try another scheduling based solution.

7.3.4 MII: Scheduling and Control for iCon Overlay

Schedulers and Controllers deal with how well the overlay is achieving its goals. At this level, the scheduling system may determine that overall, the load of the servers is too high. Therefore more resources should be booted and allocated, and the scheduler informed when those resources are available. The controller in this example would expect that the newly booted resource would absorb some of the workload, and if not, something has gone wrong.

7.4 Yakkit Performance

Yakkit is a social media application that uses location and time to facilitate communication channels. A data structure (cf. Figure 7.1) and search algorithm are used to locate relevant users for messaging. The functionality is provided as a service called the 'locality service'. The 'locality service' takes advantage of iCon's application control (cf. Section 7.3.3) to support distribution and access of the data structure.

Distribution of the data structure is desirable to support load balancing of data and support data structure operations (i.e., searching, adding, and removing of users). In this section, we examine the impact of using the iCon overlay to support service distribution.

To evaluate the performance of iCon, we propose two service deployment scenarios. The first scenario uses iCon to manage a single service instance of the 'locality service', effectively forcing iCon to manage the service as a non distributed application. The second scenario deploys multiple distributed instances of the 'locality service' and registers them with the iCon overlay, effectively forcing iCon to manage the service as a distributed application. In both scenarios, the data structure is populated with the same number of users, and the same search criteria are used to find relevant users.

Three metrics are employed to determine the impact of using iCon. The first metric is response time. Response time is used to determine how long a request takes to complete. For a distributed data structure, requests may take longer to complete. The second metric is server utilization. A single server handling all data structure operations can degrade service performance. The third metric is bandwidth utilization. The iCon overlay often has to relay requests to multiple services, this causes an increase in bandwidth usage.

Figure 7.6 depicts the results of the two scenarios described. The top graph indicates response times as the number of requests per second increases from 10 to 100. It is clear that the single service has a faster response time than the multi-service. This is due to the iCon layer relaying requests to other servers and having to wait for results. The middle graph indicates server utilization. The server utilization for both scenarios is the same as expected. However, what is missing is the server utilization of the other distributed services (i.e., the aggregated server utilization of the entire distributed system). The bottom graph indicates the system bandwidth. It is clear that the amount of bandwidth used by iCon increases as the request rate increases.

From the evaluation, we found that the iCon overlay negatively impacts response times, global server utilization, and bandwidth. However, iCon achieves a level of indirection for service orchestration and data management that simplify the deployment and access of services. iCon has the ability to distribute data structures and manage

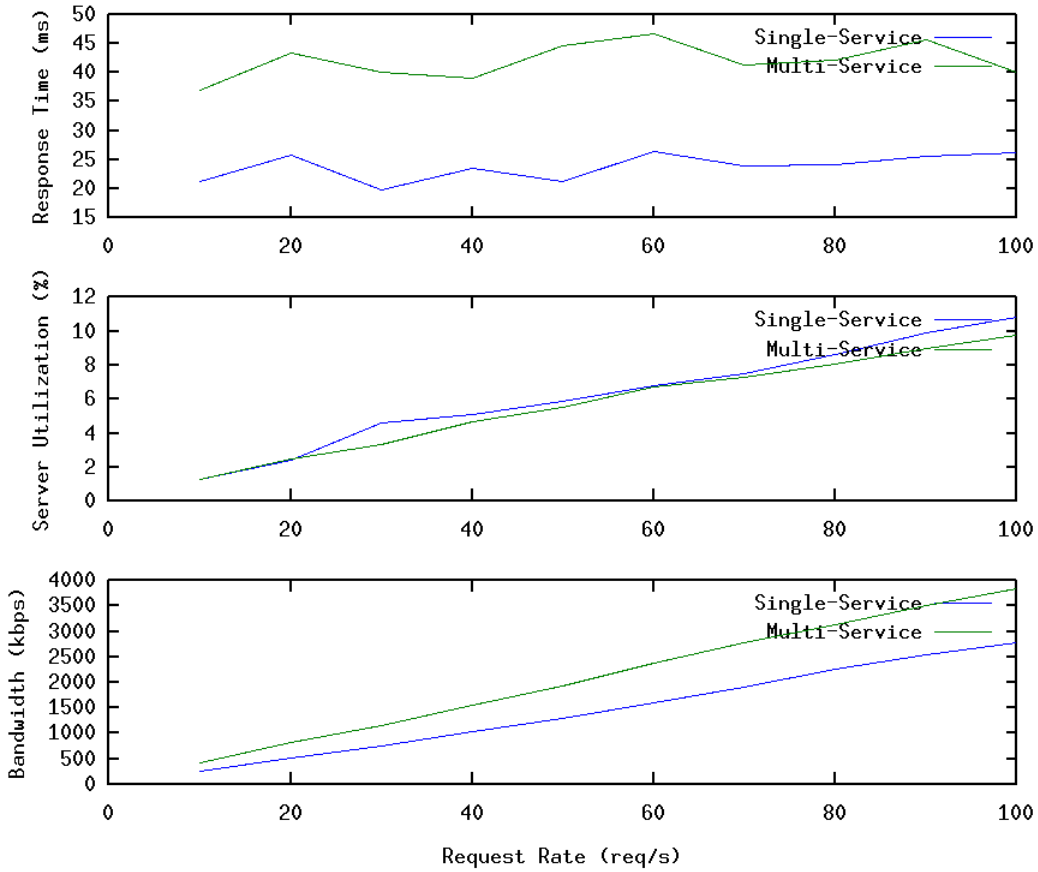

Figure 7.6: iCon's Resource Impact to support Yakkit's Locality Service

their operations. The benefits of distributed data structures and service operations need to be balanced with the requirements of the service in terms of response times, server utilization, and bandwidth. For example, if a service requires faster response times, then iCon will have to ensure only a single instance of its data structure exists or that multiple instances are in close proximity.

7.5 Summary

This chapter presented a new and innovative approach to building location-based, context-driven social networking applications in the context of deployment over a set of cloud provided resources. We designed and implemented an infrastructure, called *iCon*, and an application, called *Yakkit*, using proximity based structures hashed over

a set of clouds to distribute user load effectively. Yakkit runs on all kinds of platforms including Android, and iOS and allows the identification of persons who are in close proximity. Yakkit overlays the users' positions on top of Google maps within a broadcast range. Yakkit also supports beacons to allow users to mark positions and times of interest. This allows for truly innovative and smart web applications.

Chapter 8

Rigi Cloud

This chapter introduces RIGI Cloud, a small ten core cloud we created at the University of Victoria. RIGI Cloud allows developers of scientific applications to test their applications in a cloud testing environment before deployment. One of the key features of RIGI Cloud is to provide a platform for resource management and quality of service (QoS) testing.

Before we explore how scientific application development is supported in a cloud environment, it is advantageous to explore how it is done in a Grid computing environment. Many scientific computing communities use the computational Grid of the European Organization of Nuclear Research (CERN). This type of compute sharing paradigm arranges global access to clusters of computers from around the world. They are organized into tiers and accessed through access points that perceive groups of clusters as a single computational element. This type of system requires that application developers adhere to specific platforms and software packages. It is a burden on system administrators to support different versions of a software package. This limits application developers to use Grid resources with the proper software packages installed. This is somewhat problematic because Grid resources are used in a heterogeneous manner. These are serious issues and a motivation to provide an application testing support framework. Fortunately, virtualization software can abstract physical resources from the operating environment [ADG+06] to operate at speeds close to bare metal (i.e., native performance). We investigated web services protocols to normalize the interaction between software components [MPC+07] [AAC+10]. Interesting projects, such as BaBar, used Xen and Grid software (i.e., Globus) to deploy applications [ADG+08] [AAD+08]. Several cloud-based approaches have proven to be

useful when applied to scientific workload management [FAA+10] [LB11] [AAB+10].

This chapter describes the scientific application development process in a cloud environment in general and the design and implementation of RIGI Cloud in particular. Specifically we present an application development, testing, and an evaluation framework for scheduling and control.

8.1 Scientific Application Testing

Scientific applications that simulate natural processes can benefit from traditional testing methods. Concepts such as modularization, object-orientated programming, and functional and non-functional testing are just as important as they were decades ago. However, scientific applications are unique in that they typically execute computationally intensive simulations. This type of testing is intensive and time consuming to validate. Developer repeatedly tune application input parameters using an iterative approach. Once the developer is content with the fine tuned models, many jobs are submitted to produce high precision models that explain natural processes observed in nature. This time consuming process requires many computational resources to complete in a reasonable amount of time.

It is important to understand how scientists develop and test their applications. Scientists typically use software packages that have already been developed by other researchers. The software is used to develop scientific applications that produce models of real world processes. A sample application package is the Monte-Carlo simulation package. For example, a scientist develops an application that simulates a natural process. The scientist executes it with a sample of random input to generate an crude output model. The precision of the output model depends on how many random events are generated. The model is then checked to see if it is correct and adjustments are made accordingly. This process continuous until the scientist is confident in the quality of the application model. The application is then deployed to the Grid for execution. After execution, the output model is analyzed determine whether if it support the scientist's hypothesis. This process continues until the scientist has enough information to defend his or her claims.

The majority of development time is on testing and tweaking the application to gain confidence before deployment to the Grid. This is where the proposed cloud testing architecture should assist in alleviating some of the burden by automating some of the testing. The cloud testing architecture should support the following:

- Target Architecture — supports the specification of one or more target architectures/platforms to test applications.
- Operating Environment — supports the identification of the operating environment applications require.
- Software Packages — supports the specification of software packages applications require.
- Application Analysis — supports interactive and non-interactive analysis of the application's results. Simple testing such as comparing results to expected output is straight forward, but other checks may require human interaction.
- Application Deployment Topology Support — supports simulation of complicated deployment topologies.
- Production Testing — support the ability to test deployment on a cloud production system (e.g., use a cloud scheduling system as described in Armstrong [AAB+10]).

Fortunately, support for a cloud production system is being developed in the Physics and Astronomy department at the University of Victoria under the leadership and direction of Randall Sobie and Ian Gable. They developed an innovative approach to integrating both commercial and scientific clouds for High Performance Computing applications [AAB+10]. In this system, application developers build their tested applications in a VM image. They use a cloud scheduler to boot their VMs on a selected cloud provider (e.g., Amazon's EC2). The developer then submits jobs to the job scheduler which targets the VM image they developed their application on. Their job scheduler employs the Condor (now HTCondor) job scheduler.[1] This scheduler listens for resources to publish their availability which is ideal for a multi-site cloud environment. This scheduler is an example of using Infrastructure as a Service (IaaS).

[1] http://research.cs.wisc.edu/htcondor/

8.2 RIGI Cloud Design and Implementation

RIGI Cloud's design adheres to the three layer cloud architecture. The first layer provisions physical infrastructure, the second layer provisions platform requirements, and the third layer provisions software packages and services.

Figure 4.3 depicts the usage of the first layer of the cloud known as Infrastructure as a Service (IaaS). In this case, the scientific user requested the system boot a Linux machine with two processors and two gigabytes of RAM. The user then installs the application platform (i.e., Apache Tomcat) and their applications and then submits their work load (i.e., jobs).

Figure 8.1 depicts the usage of the second layer of the cloud known as Platform as a Service (PaaS). In this case, the scientific user requested the system boot a Linux machine and install the Apache Tomcat services container. The user configures their web applications in the Tomcat container and submits their workload (i.e., jobs) for that web application.

Figure 8.2 depicts the usage of the third layer of the cloud known as Software as a Service (SaaS). In this case, the scientific user requested the system boot a Linux machine and install the Apache Tomcat development and deployment platform along with their web application fully configured and deployed. The user then submits their workload (i.e., jobs) to the web application.

In the context of this dissertation, the thee layers of cloud provisioning (IaaS, PaaS, and SaaS) reside at the management and orchestration layers of the ACRA model as depicted in Figure 2.2 and take advantage of Buyya's economic model to request and bid for resources (cf. Figures 2.1 and 2.2). For example, software products, such as Apache Tomcat, can be purchased for installation and configuration for a web application. The SaaS layer uses the Broker (cf. Chapter 2) to bid on infrastructure to host the Apache Tomcat engine. After the software product is installed and configured, and the web application is deployed and started, SaaS users can use the web application. Section 8.3 proposes and explores a framework to support scientific application development and testing prior to deployment on a set of cloud providers.

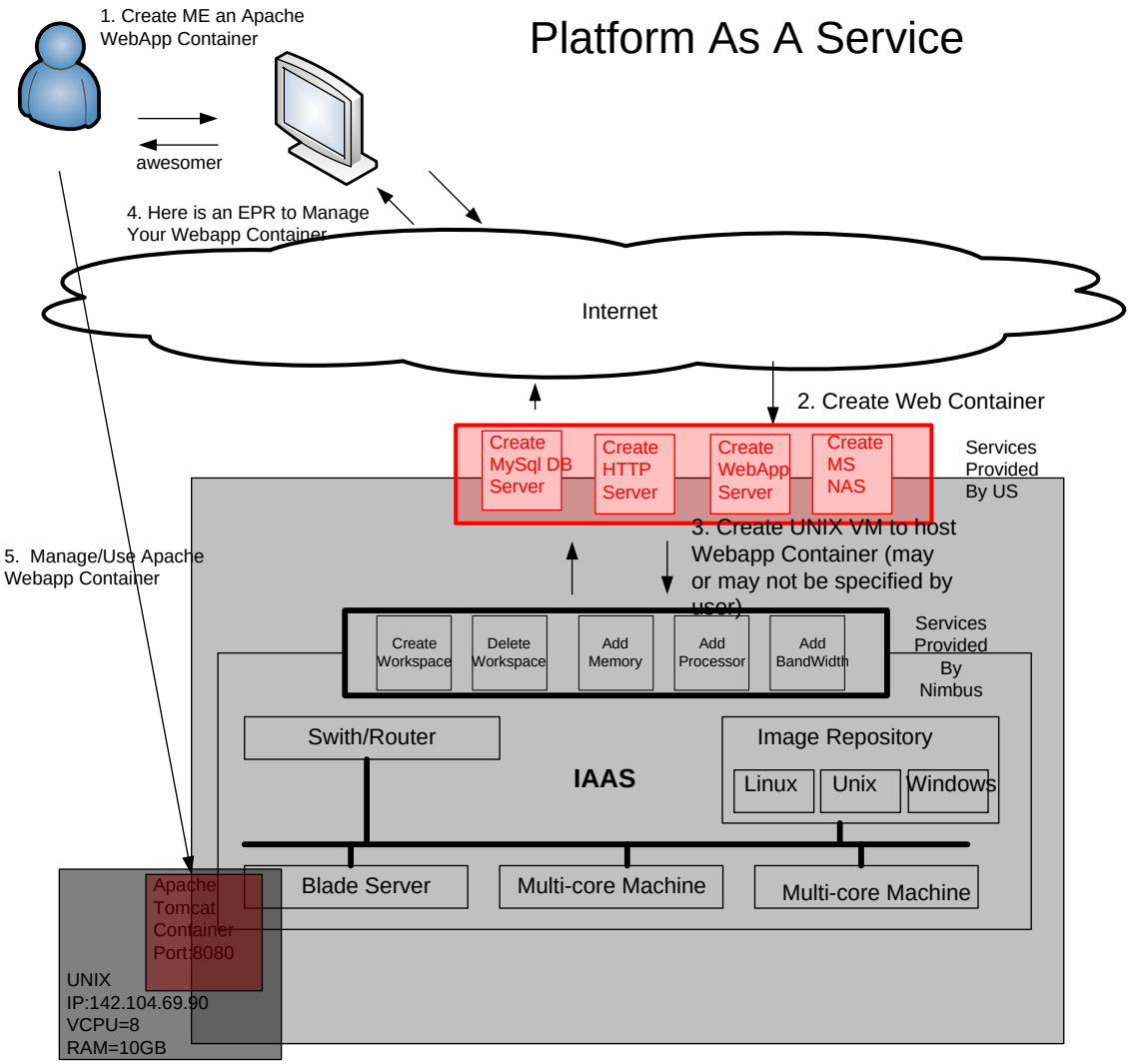

Figure 8.1: Platform as a Service

8.3 Application Development and Testing Framework

Figure 8.3 illustrates the architecture to support scientific testing on the cloud. Abstractly, the architecture is divided into three layers. The User Interface Layer, the Management and Orchestration Layer, and the System Layer. This is consistent with the ACRA model depicted in Figure 2.2.

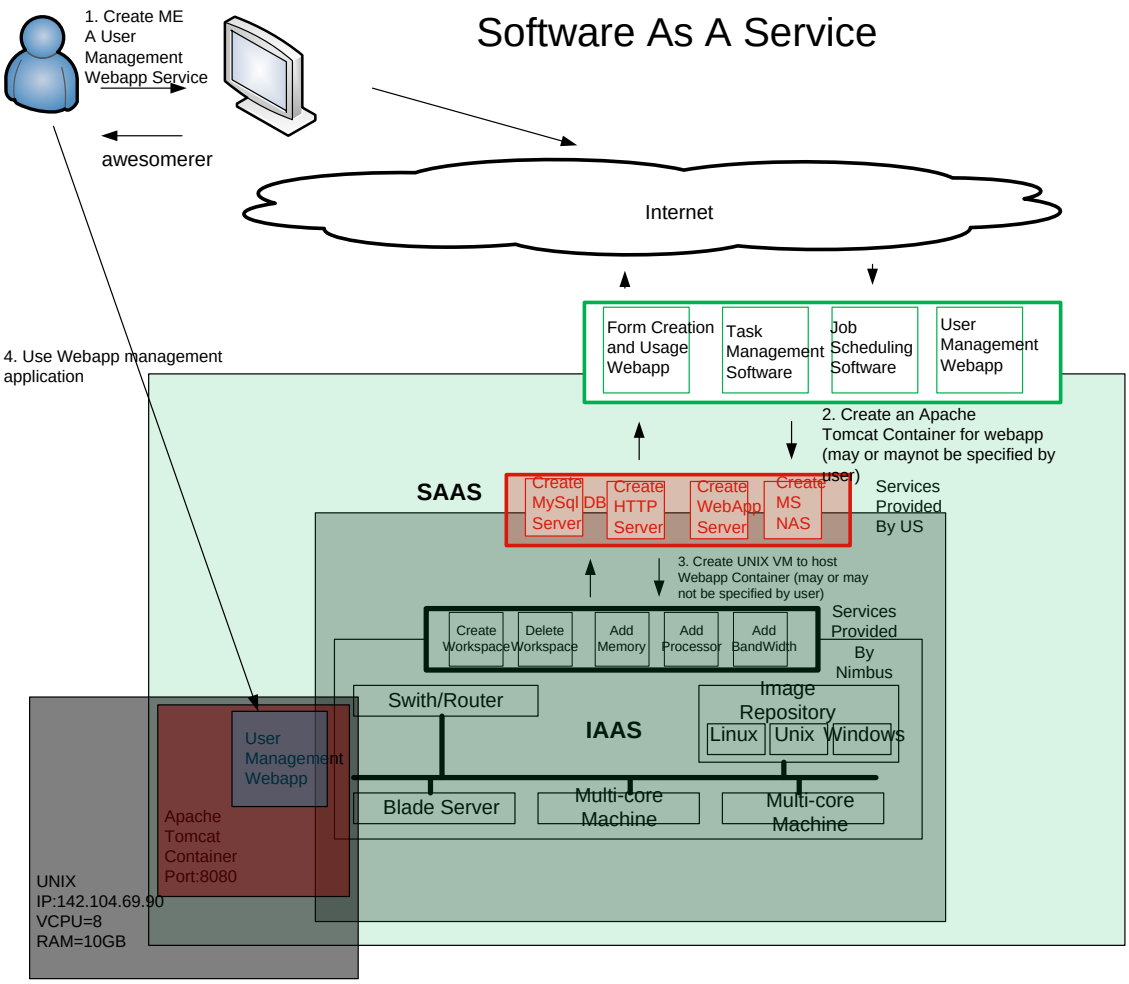

Figure 8.2: Software as a Service

The System Layer is responsible for provisioning, booting, and storing of virtual machine images, managing software, and networking. This layer provisions three types of virtual machines; a development machine, an application testing machine, and a system simulation machine. The development machine provides the developer with a virtual desktop, as well as development and analysis tools. The application test machine is specified by the developer as a target architecture and platform that the application executes on. The system simulation machine is configurable to simulate network components such as routers, gateways, or simulated software services that are made available in the production environment. Figure 8.4 depicts the architecture of this layer. To support dynamic testing of applications, a virtual cluster is managed on

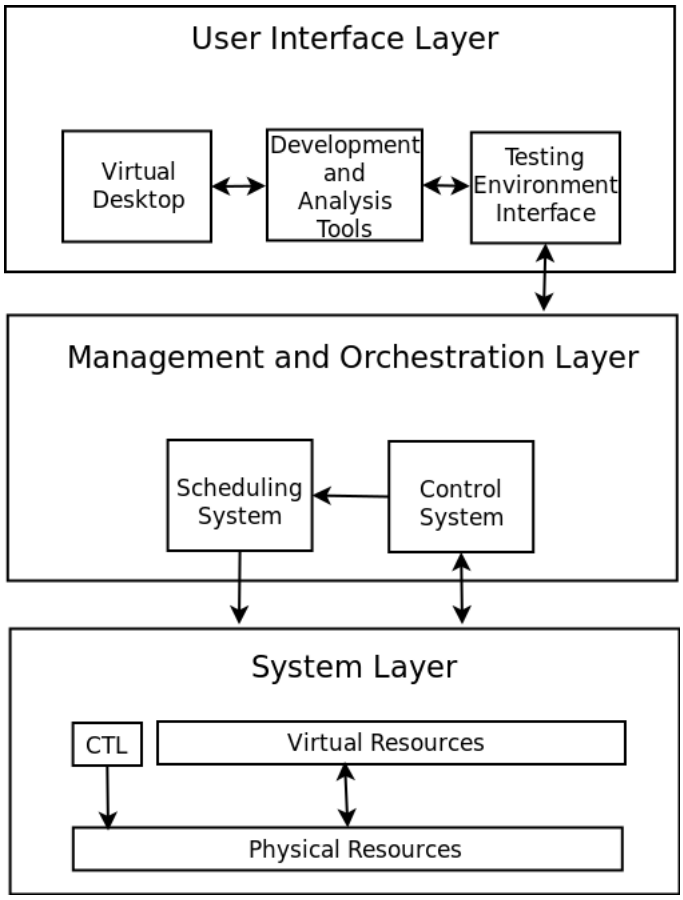

Figure 8.3: Three Layer Testing Architecture

top of the physical one. The virtual cluster is managed through special pilot jobs to the physical cluster. The pilot jobs boot and shutdown virtual machines of a specified platform. There are five types of jobs that manage the virtual cluster.

- Boot/Shutdown Jobs —These types of jobs start and stop virtual machines.
- Software Install/Remove Jobs —These types of jobs install and remove software from the virtual machines.
- Sensor Jobs —These jobs get information out of the virtual machines that is useful by the scheduler and control systems in the management and orchestration layer.
- Actuator jobs —These types of jobs configure or change virtual machines.

- Test Application Jobs —These are the jobs specified by the developer that test their application's performance.

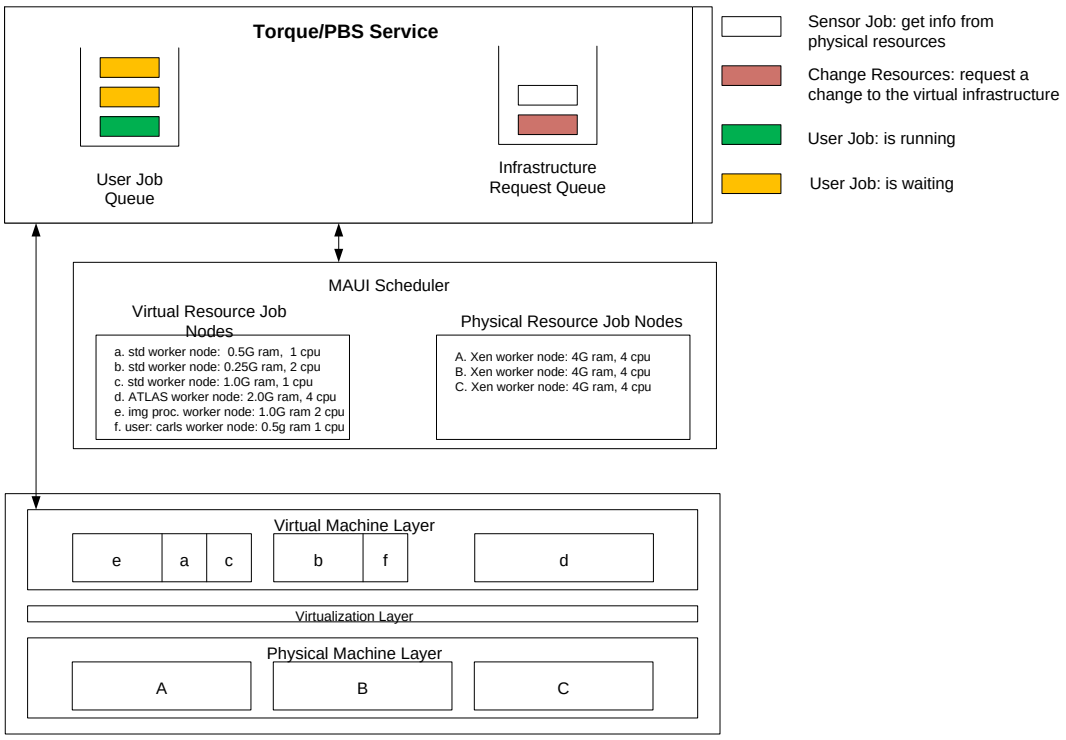

Figure 8.4: System Layer Architecture

The idea is that the management and orchestration layer uses these jobs to create infrastructure for scientific applications. For example, the developer may be interested to know if their application executes on three different versions of Linux. This layer will boot these versions of Linux and execute the applications on all three. The output of the applications can be compared for consistency and accuracy.

The management and orchestration layer is responsible for managing requests from the developer, scheduling requests to manage virtual resources, and scheduling requests for installing or removing software. This layer provides scheduling and control capabilities that are configurable by the developer. In effect, testing is realized through this layer by taking advantage of dynamic capabilities provided by the Sys-

tem Layer.

The user interface layer is responsible for how the developer interacts with the system. This layer provides a virtual desktop environment such as VNC[2] or No Machine[3] and provides the development environment including analysis tools.

8.4 Scheduling and Control Evaluation Framework

Developers test their applications using the management and orchestration layer. This layer determines which resources are necessary to test the application as specified by the testing policy. The testing policy is specified by the user, but is constrained by the physical resources and the control and scheduling systems. For example, a developer could request that an application be deployed over at least two Linux machines with 500 MB of RAM. Once the application completes execution, the application's output is compared to the expected output. If the output is valid, then a larger test may proceed with a larger number of random valued inputs using more Linux machines. If the output model is not reasonable, then a report is generated.

The heart of the management and orchestration layer is the scheduling system. In this case, the scheduling system models the physical layer as a Cutting Stock Problem as described in Chapter 4. Formulating the problem this way, the scheduling system has many possible configurations it can choose from to deploy the user applications that have to share a limited set of physical resources.

The controller manages the starting and stopping of the virtual machines, along with the installing and removing of software. The controller is also responsible for tuning the scheduling system, this allows the scheduler to monitor virtual machine performance.

Figure 8.5 depicts the scheduling and control system that manages applications and the underlying physical system.

[2]http://www.realvnc.com/
[3]http://www.nomachine.com/

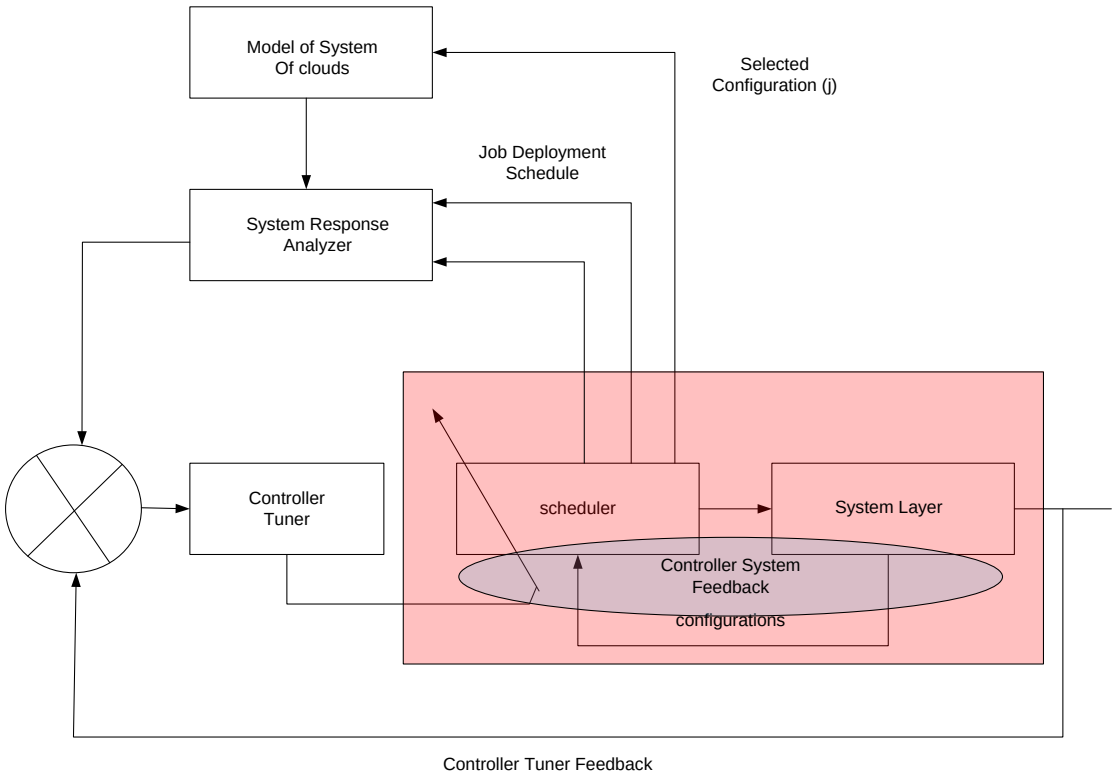

Figure 8.5: Scheduling and Controller Provisioning System

The scheduling part uses algorithms to solve the partitioning of the resources and the determining which virtual machines need to be booted or shutdown. The controller manages resources and determines the health of the System Layer. The controller has the ability to fine tune the scheduler in the case virtual machines are not operating properly.

8.5 Experience Using Rigi Cloud

Rigi Cloud was demonstrated at CASCON 2011 in Toronto. We presented an environment for a developer to deploy and test their applications on a localized test cloud. The objective is to provide the developer with a sense of how well their application performs when distributed with different topologies in a cloud environment.

The Rigi Cloud demonstration used two quad processor servers with Xen virtualization software and a dual core headnode with Torque and the Maui scheduling system installed as described in Section 8.3. The scheduling system is managed by a MAPE-K control system implemented with bash shell scripts. These scripts monitored the system for user requests and system state. The control system manages resources according to the user's policy file. For example, a user could request the MAPE-K controller automatically boot more instances of their application if the number of application requests reaches a critical threshold. With this ability provided to the developers, they could determine a good threshold level for their application by testing and evaluating different deployment configurations. Once the developer understands how their application performs on the test cloud, they can deploy to a production cloud such as Amazon's EC2 and set a limit on the number instances they require to support their users.

The demonstration performed well and generated alot of interest in how we setup our system using Torque (i.e., a free off the shelf product). The system proved that a simple cluster with virtualization could be extremely useful for users to test their applications using different deployment topologies. The innovative contribution is using Torque to schedule infrastructure (i.e., boot and shutdown VMs) and using a MAPE-K control system to manage the virtual infrastructure and schedule developer test jobs.

8.6 Summary

In this chapter we presented RIGI Cloud as a development and deployment architecture. We first demonstrated Rigi Cloud at CASCON 2011 in Toronto. The demonstration proved that users could rely on our system to construct a deployment topology for their applications. Thus, developers can experiment with different strategies in order to find a good deployment strategy for production clouds.

Chapter 9

Conclusion

9.1 Summary

This dissertation investigated approaches for resource provisioning and task allocation for clouds within a novel economics based provisioning paradigm managed within an ACRA model (cf. Figure 2.2). In particular, we presented and evaluated approaches using scheduling theory (cf. Chapters 4 5) and control theory (cf. Chapter 6). These approaches indicated improvements in job scheduling revenue and provide scheduling guarantees for scenarios that require a minimum performance in terms of revenue.

From the perspective of Buyya's and Fox's economics based provisioning system (cf. Figure 2.1), the scheduling approaches described in Chapters 4 and 5 are applicable to the brokerage, cloud exchange and cloud coordinator elements. The brokerage element can take advantage of scheduling guarantees to ensure that their client applications are executing above a minimum threshold of performance. The cloud exchange can take advantage of the cutting stock formulation to get the best price from resource providers to the brokers. The cloud coordinators can also use the cutting stock formulation to minimize their usage of over flow clouds (which cost more money) to achieve their SLAs.

From the perspective of using the ACRA model to manage the economics based provisioning system (cf. Figure 2.2), the control approaches described in Chapter 6 are applicable at ACRA's autonomic managers layer, and scheduling approaches described in Chapter 4 are applicable to ACRA's orchestration layer. ACRA's auto-

nomic managers layer can use feedback control to tune the brokerage's scheduling model (cf. Figure 8.5). ACRA's orchestration layer can take advantage of the cutting stock model to manage multiple brokers so user applications get the best price from the most reliable broker.

9.2 Contributions

The contributions of this dissertation is as follows:

- Chapter 2 — describes current research along several dimensions in the form of taxonomies. Eleven taxonomies are proposed to categorize literature aspects for our cloud domain (cf. Figure 2.2).

- Chapter 3 — dimensions the cloud research domain and classifies relevant literature using several of the taxonomies developed in Chapter 2. We use the cloud paradigm developed by Buyya et al. [BYV+09] and Fox et al. [AFG+10] within a feedback framework developed by IBM [IBM06] along with a taxonomy on cloud computing to categorize the literature. In specifics, we selected the control and scheduling taxonomies to focus our research. These taxonomies proved useful to suggest solutions to targeted problems in the cloud domain related.

- Chapter 4 — introduces the cutting stock problem and maps cloud resource provisioning problems to the cutting stock problem. Results show improved schedules when using the cutting stock formulation. The model has been adapted and published as a case study for scheduling to a distributed set of clouds [BDM+11].

- Chapter 5 — uses a model-driven approach to provide scheduling guarantees. The problem formulation is manipulated so it has specific structural properties. Problem formulations with specific structural properties provide guarantees with respect to solution quality. In this case, we use the greedy algorithm to solve the scheduling problems [BDM+14]. In these works, we demonstrated through simulation our novel scheduling formulations. The results indicate that for an exponential distributed workload, the greedy algorithm produces more schedules with close to optimal revenue versus schedules with guaranteed revenue.

- Chapter 6 — presents experiments and works with feedback and utility as applied to computing systems. A taxonomy is presented which categorizes feedback with utility, along with a description of the PID controller, autonomic controllers, and Kalman filters. This work describes control based strategies using Kalman filters and stochastic modeling, these novel approaches provided insights for predictive admission control. This work further describes how feedback and utility can be applied with scheduling and where it is applicable in the cloud paradigm.

- Chapter 7 — presents Yakkit, a case study that investigates the use of an inter-cloud overlay network (iCON). The overlay provides support for a distributed data structure and an interface for searching and message passing algorithms. This work investigates how an overlay could potentially make inter-cloud communication ubiquitous. This work was published in the 2011 CASCON Proceedings [DLM11].

- Chapter 8 — presents RIGI cloud, a test bed for cloud experiments and exploring scheduling/control scenarios. A scheduling and testing framework is presented to boot and execute virtual infrastructure for experiments. This work extended previous work on providing an autonomic grid management system which was published in Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2007) [DM07]. We demonstrated this work at CASCON (2010) as a cloud cluster queue in which we match virtual worker-node resources dynamically to the user workload queued on the head-node.

9.3 Future Work

Cloud computing is still a growing field with plenty of opportunities to conduct research. In particular, more research to integrate scheduling and control systems, along with the identification of autonomic patterns (i.e., natural balances of computing components in their environment). Moreover, we need more research on economic benefits and payment systems.

We have implemented a prototype of the iCon framework (cf. Chapter 7) on Amazon's EC2. Using this service we can manually add and remove applications from the overlay. In the future we plan to use the overlay to manage our Yakkit

applications as part of our business infrastructure for Yakkit Media Corporation Inc. This management infrastructure will likely be able to take advantage of most of the research results of this dissertation to be competitive in providing a series of mobile login-less context based social messaging applications.

Bibliography

[AAB+07] A. Agarwal, M. Ahmed, A. Berman, B. Caron, A. Charbonneau, D. Deatrich, R. Desmarais, A. Dimopoulos, I. Gable, L. Groer, R. Haria, R. Impey, L. Klektau, C. Lindsay, G. Mateescu, Q. Matthews, A. Norton, W. Podaima, D. Quesnel, R. Simmonds, R. Sobie, B. St. Arnaud, C. Usher, D. Vanderster, M. Vetterli, R. Walker, and M. Yuen. GridX1: A Canadian computational grid. *Future Generation Computer Systems (FGCS)*, 23(5):680–687, 2007.

[AAB+10] P. Armstrong, A. Agarwal, A. Bishop, A. Charbonneau, R. Desmarais, K. Fransham, N. Hill, I. Gable, S. Gaudet, S. Goliath, C. Leavett-Brown, J. Ouellete, M. Paterson, C. Pritchet, D. Penfold-Brown, W. Podaima, D. Schade, and R. Sobie. Cloud Scheduler: a resource manager for distributed compute clouds. *Arxiv preprint arXiv:1007.0050*, 2010.

[AAC+10] A. Agarwal, P. Armstrong, A. Charbonneau, H. Chen, R. Desmarais, I. Gable, D. Goodenough, A. Guan, R. Impey, B. Moa, W. Podaima, and R. Sobie. Service-Oriented Grid Computing for SAFORAH. In *High Performance Computing Systems and Applications*, volume 5976 of *Lecture Notes in Computer Science*, pages 283–291. Springer Berlin Heidelberg, 2010.

[AAD+08] A. Agarwal, P. Armstrong, R. Desmarais, I. Gable, S. Popov, S. Ramage, S. Schaffer, C. Sobie, R. Sobie, T. Sulivan, D. Vanderster, G. Mateescu, W. Podaima, A. Charbonneau, R. Impey, M. Viswanathan, and D. Quesnel. BaBar MC Production on the Canadian Grid using a Web Services Approach. *Journal of Physics: Conference Series (JoP)*, 119:072002, 2008.

[ABKM01] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 131–145. ACM, 2001.

[AdAM09] S. Andrade and R. de Araujo Macedo. A non-intrusive component-based approach for deploying unanticipated self-management behaviour. In *Proceedings 4th IEEE International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 152–161. IEEE, 2009.

[ADG+06] A. Agarwal, R. Desmarais, I. Gable, A. Norton, R. Sobie, and D. Vanderster. Evaluation of Virtual Machines for HEP Grids. Technical report, University of University of Victoria (UVIC), Canada, 2006.

[ADG+08] A. Agarwal, R. Desmarais, I. Gable, D. Grundy, D. P-Brown, R. Seuster, D. Vanderster, A. Charbonneau, R. Enge, and R. Sobie. Deploying HEP Applications Using Xen and Globus Virtual Workspaces. *Journal of Physics: Conference Series (JoP)*, 119(6):062002, 2008.

[AFG+10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[ÅW11] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems: Theory and Design*. Courier Dover Publications, 2011.

[BAGS02] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in Grid computing. *Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15):1507–1542, 2002.

[BDG+13] Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, and M. Smit. A Design Space for Self-Adaptive Systems. In *Collection Software Engineering for Self-Adaptive Systems II (SEAMS)*, volume 7475 of *Lecture Notes in Computer Science (LNCS)*, pages 33–50. Springer Berlin Heidelberg, 2013.

[BDM+11] S. Balasubramanian, R. Desmarais, H. A. Müller, U. Stege, and S. Venkatesh. Characterizing Problems for Realizing Policies in Self-

Adaptive and Self-Managing Systems. In *Proceedings 6th ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 70–79. ACM, 2011.

[BDM+14] S. Balasubramanian, R. Desmarais, H. A. Müller, U. Stege, and S. Venkatesh. Characterizing Problems for Realizing Policies in Self-Adaptive and Self-Managing Systems. *To be published in ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2014.

[Ben75] J. L. Bentley. Survey of Techniques for Fixed Radius Near Neighbor Searching. Technical report, Stanford Linear Accelerator Center (SLAC), USA, 1975.

[BRC10] R. Buyya, R. Ranjan, and R. Calheiros. Intercloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *Collection Algorithms and Architectures for Parallel Processing (AAPP)*, volume 6081 of *Lecture Notes in Computer Science (LNCS)*, pages 13–31. Springer, 2010.

[Buy02] R. Buyya. *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, Monash University, Australia, 2002.

[BYV+09] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype and reality for delivering computing as the 5th utilty. *Future Generation Computer Systems (FGCS)*, 25(6):599–616, 2009.

[CJ01] B. Cirou and E. Jeannot. Triplet: a Clustering Scheduling Algorithm for Heterogeneous Systems. In *Proceedings IEEE International Conference on Parallel Processing Workshops (ICPPW)*, pages 231–236. IEEE, 2001.

[CRB+11] R. N. Calheiros, R. Ranjan, A. Belonglozov, C. A. De Rose, and R. Buyya. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience (SPE)*, 41(1):23–50, 2011.

[DA06] F. Dong and S. G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Technical report, School of Computing, Queens University, Kingston, Ontario, 2006.

[DDKM08] D. Dawson, R. Desmarais, H. M. Kienle, and H. A. Müller. Monitoring in Adaptive Systems using Reflection. In *Proceedings International ACM Workshop on Software engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 81–88. ACM, 2008.

[DHP+05] Y. Diao, J. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung. A Control Theory Foundation for Self-Managing Computing Systems. *Selected Areas in Communications (IEEE)*, 23(12):2213–2222, 2005.

[DIIM04] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *Proceedings 20th ACM Annual Symposium on Computational Geometry (SCG)*, pages 253–262. ACM, 2004.

[DLM11] R. J. Desmarais, P. Lach, and H. A. Müller. YaKit: A Locality Based Messaging System Using iCon Overlay. In *Proceedings International IEEE Conference of the Center for Advanced Studies on Collaborative Research (CASCON)*, pages 148–159. IEEE, 2011.

[DM07] R. Desmarais and H. Müller. A Proposal for an Autonomic Grid Management System. In *Proceedings International IEEE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 11–11. IEEE, 2007.

[DMK+13] D. Duplyakin, P. Marshall, K. Keahey, H. Tufo, and A. Alzabarah. Rebalancing in a multi-cloud environment. In *Proceedings 4th ACM Workshop on Scientific Cloud Computing*, pages 21–28. ACM, 2013.

[DNB05] C. H. Ding, S. Nutanong, and R. Buyya. Peer-to-Peer Networks for Content Sharing. *Peer-to-Peer Computing: The Evolution of a Disruptive Technology*, pages 28–65, 2005.

[DRM+10] X. Dutreilh, N. Rivierre, A. Moreau, J. Malenfant, and I. Truck. From Data Center Resource Allocation to Control Theory and Back. In

Proceedings 3rd IEEE International Conference on Cloud Computing (CLOUD), pages 410–417. IEEE, 2010.

[Edm71] J. Edmonds. Matroids and the Greedy Algorithm. *Mathematical Programming (MP)*, 1(1):127–136, 1971.

[FAA+10] K. Fransham, A. Agarwal, P. Armstrong, A. Bishop, A. Charbonneau, R. Desmarais, N. Hill, I. Gable, S. Gaudet, S. Goliath, R. Impey, C. Leavett-Brown, J. Ouellete, M. Paterson, P. C, D. Penfold-Brown, W. Podaima, D. Schade, and R. Sobie. Research computing in a cistributed cloud environment. *Journal of Physics: Conference Series (JoP)*, 256:012003, 2010.

[FB74] R. A. Finkel and J. L. Bentley. Quad Trees A Data Structure for Retrieval on Composite Keys. *Acta Informatica (AI)*, 4(1):1–9, 1974.

[FK03] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Elsevier, 2003.

[FS07] G. Folino and G. Spezzano. An autonomic tool for building self-organizing Grid-enabled applications. *Future Generation Computer Systems (FGCS)*, 23(5):671–679, 2007.

[GB12] N. Grozev and R. Buyya. Inter-Cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, 2012.

[GG61] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[GL09] H. Ghanbari and M. Litoiu. Identifying Implicitly Declared Self-Tuning Behavior Through Dynamic Analysis. In *Proceedings IEEE ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 48–57. IEEE, 2009.

[GSLI12] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai. Feedback-based optimization of a private cloud. *Future Generation Computer Systems (FGCS)*, 28(1):104–111, 2012.

[HB05] M. Hakem and F. Butelle. Dynamic Critical Path Scheduling Parallel Programs onto Multiprocessors. In *Proceedings 19th IEEE International on Parallel and Distributed Processing Symposium (PDPS)*, pages 203b–203b. IEEE, 2005.

[HDPT04] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. Wiley. com, 2004.

[HDPT05] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. Control Engineering for computing systems-Industry experience and research challenges. *Control Systems (CS)*, 25(6):56–68, 2005.

[Hel04] J. L. Hellerstein. Challenges in Control Engineering of Computing Systems. In *Proceedings American Control Conference (ACC)*, volume 3, pages 1970–1979. IEEE, 2004.

[HMGW07] C. Hyser, B. Mckee, R. Gardner, and B. J. Watson. Autonomic Virtual Machine Placement in the Data Center. Technical report, Hewlett Packard Laboratories (HPL), 2007.

[HWIL09] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu. Resource Provisioning for Cloud Computing. In *Proceedings Conference of the Center for Advanced Studies on Collaborative Research (CASCON)*, pages 101–111. IBM Corp., 2009.

[IBM06] IBM Corporation. An Architectural Blueprint for Autonomic Computing. *IBM White Paper*, 2006.

[K+60] R. E. Kalman et al. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering (JoBE)*, 82(1):35–45, 1960.

[KAB+12] K. Keahey, P. Armstrong, J. Bresnahan, D. LaBissoniere, and P. Riteau. Infrastructure outsourcing in multi-cloud environment. In *Proceedings of the 2012 workshop on Cloud services, federation, and the 8th open cirrus summit*, pages 33–38. ACM, 2012.

[Kal09] E. Kalyvianaki. Resource Provisioning for Virtualized Server Applications. Technical Report UCAM-CL-TR-762, University of Cambridge, Computer Laboratory, UK, 2009.

[KR02] D. R. Karger and M. Ruhl. Finding Nearest Neighbors in Growth-Restricted Metrics. In *Proceedings 34th ACM Annual Symposium on Theory of computing (TOC)*, pages 741–750. ACM, 2002.

[LB11] C. Leavett-Brown. Simulation and user analysis of babar data in a distributed cloud. In *Proceedings International Symposium on Grids and Clouds and the Open Grid Forum (ISGC)*, volume 1, page 86, 2011.

[LNM+12] O. Lardière, R. Nash, J.-P. Markes, D. Andersen, C. Bradley, C. Blain, R. Desmarais, D. Gamroth, M. Ito, K. Jackson, et al. Final opto-mechanical design of Raven, a MOAO science demonstrator for Subaru. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 844753–844753. International Society for Optics and Photonics, 2012.

[LP98] J.-C. Liou and M. A. Palis. A New Heuristic for Scheduling Parallel Programs on Multiprocessor. In *Proceedings IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 358–365. IEEE, 1998.

[LSTS99] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Design and Evaluation of a Feedback Control EDF Scheduling Algorithm. In *Proceedings 20th IEEE Real-Time Systems Symposium (RTSS)*, pages 56–67. IEEE, 1999.

[Mes06] J. Mestre. Greedy in Approximation Algorithms. In *Collection Algorithms ESA*, volume 4168 of *Lecture Notes in Computer Science*, pages 528–539. Springer Berlin Heidelberg, 2006.

[MK12] I. A. Moschakis and H. D. Karatza. Evaluation of gang scheduling performance and cost in a cloud computing system. *The Journal of Supercomputing*, 59(2):975–992, 2012.

[MPC+07] G. Mateescu, W. Podaima, A. Charbonneau, R. Impey, M. Viswanathan, A. Agarwal, P. Armstrong, R. Desmarais, I. Gable, S. Popov, et al. The GridX1 computational Grid: from a set of service-specific protocols to a service-oriented approach. In *Proceedings 21st IEEE International Symposium on High Performance Computing Systems and Applications (HPCS)*, pages 8–8. IEEE, 2007.

[MPS08] H. Müller, M. Pezzè, and M. Shaw. Visibility of Control in Adaptive Systems. In *Proceedings 2nd ACM International ICSE Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS)*. ACM, 2008.

[Mül06] H. A. Müller. Bits of History, Challenges for the Future and Autonomic Computing Technology. In *Proceedings 13th IEEE Working Conference on Reverse Engineering (WCRE)*, pages 9–18. IEEE, 2006.

[Mur03] R. M. Murray. *Control in an information rich world: Report of the panel on future directions in control, dynamics, and systems*. SIAM, 2003.

[NCS12] A. Nathani, S. Chaudhary, and G. Somani. Policy based resource allocation in IaaS cloud. *Future Generation Computer Systems (FGCS)*, 28(1):94–103, 2012.

[NWF78] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions—I. *Mathematical Programming (MP)*, 14(1):265–294, 1978.

[Oga87] K. Ogata. *Discrete-Time Control Systems*, volume 1. Prentice-Hall Englewood Cliffs, NJ, 1987.

[PDC08] F. Pop, C. Dobre, and V. Cristea. Performance Analysis of Grid DAG Scheduling Algorithms using MONARC Simulation Tool. In *Proceedings IEEE International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 131–138. IEEE, 2008.

[PGH+02] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using Control Theory to Achieve Service Level Objectives in Performance Management. *Real-Time Systems (RTS)*, 23(1-2):127–141, 2002.

[PHVD04] R. Parra-Hernandez, D. Vanderster, and N. J. Dimopoulos. Resource Management and Knapsack Formulations on the Grid. In *Proceedings 5th IEEE/ACM International Workshop on Grid Computing (IWGC)*, pages 94–101. IEEE, 2004.

[PSS+12] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski. Introducing STRATOS: A cloud broker service. In *Proceedings 5th IEEE International Conference on Cloud Computing (CLOUD)*, pages 891–898. IEEE, 2012.

[QKP+09] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma. Towards Autonomic Workload Provisioning for Enterprise Grids and Clouds. In *Proceedings 10th IEEE International Conference on Grid Computing (GC)*, pages 50–57. IEEE, 2009.

[Ran07] R. Ranjan. *Coordinated Resource Provisioning in Federated Grids*. PhD thesis, University of Melbourne, Australia, 2007.

[Rav00] O. Ravn. On-line System Identification and Adaptive Control using the Adaptive Blockset. *the 2nd IFAC Symposium on System Identification (IFACSI)*, 2000.

[RCL09] B. P. Rimal, E. Choi, and I. Lumb. A Taxonomy and Survey of Cloud Computing Systems. In *Proceedings 5th IEEE International Joint Conference on INC, IMS and IDC (NCM)*, pages 44–51. IEEE, 2009.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 161–172. ACM, 2001.

[SER+10] M. Sama, S. Elbaum, F. Raimondi, D. S. Rosenblum, and Z. Wang. Context-Aware Adaptive Applications: Fault Patterns and Their Automated Identification. *Transactions on Software Engineering (TSE)*, 36(5):644–661, 2010.

[SILI10] B. Solomon, D. Ionescu, M. Litoiu, and G. Iszlai. Autonomic Computing Control of Composed Web Services. In *Proceedings ACM ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 94–103. ACM, 2010.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applica-

tions. In *Proceedings ACM Computer Communication Review (SIGCOMM)*, volume 31, pages 149–160. ACM, 2001.

[SSWS07] Y. Song, Y. Sun, H. Wang, and X. Song. An Adaptive Resource Flowing Scheme Amongst VMs in a VM-Based Utility Computing. In *Proceedings 7th IEEE International Conference on Computer and Information Technology (CIT)*, pages 1053–1058. IEEE, 2007.

[TMMVL12] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems (FGCS)*, 28(2):358–367, 2012.

[Van08] D. C. Vanderster. *Resource Allocation and Scheduling Strategies using Utility and the Knapsack Problem on Computational Grids*. PhD thesis, University of Victoria, Canada, 2008.

[VDPHS06] D. C. Vanderster, N. J. Dimopoulos, R. Parra-Hernandez, and R. J. Sobie. Evaluation of Knapsack-based Scheduling using the NPACI JOBLOG. In *Proceedings 20th IEEE International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS)*, pages 15–15. IEEE, 2006.

[VDS07] D. C. Vanderster, N. J. Dimopoulos, and R. J. Sobie. Intelligent Selection of Fault Tolerance Techniques on the Grid. In *Proceedings IEEE International Conference on e-Science and Grid Computing (ICSGC)*, pages 69–76. IEEE, 2007.

[WKGB12] L. Wu, S. Kumar Garg, and R. Buyya. SLA-based admission control for a Software-as-a-Service provider in cloud computing environments. *Journal of Computer and System Sciences*, 78(5):1280–1299, 2012.

[WLW+07] X. Wang, D. Lan, G. Wang, X. Fang, M. Ye, Y. Chen, and Q. Wang. Appliance-based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center. In *Proceedings 4th IEEE International Conference on Autonomic Computing (ICAC)*, pages 29–29. IEEE, 2007.

[Xu07] W. Xu. Feedback for Compute Systems. Master's thesis, University of Melbourne, Austrailia, 2007.

[YB06] C. S. Yeo and R. Buyya. Managing Risk of Inaccurate Runtime Estimates for Deadline Constrained Job Admission Control in Clusters. In *Proceedings IEEE International Conference on Parallel Processing (ICPP)*, pages 451–458. IEEE, 2006.

[YG94] T. Yang and A. Gerasoulis. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 5(9):951–967, 1994.

[YTX04] Y. Yang, L. Tan, and N. Xiong. A Resource-Based Admission Control Algorithm for Grid Computing Systems. In *Proceedings 4th IEEE International Conference on Computer and Information Technology (CIT)*, pages 364–368. IEEE, 2004.

[ZC05] A. Y. Zomaya and G. Chan. Efficient Clustering for Parallel Tasks Execution in Distributed Systems. *International Journal of Foundations of Computer Science (IJFCS)*, 16(02):281–299, 2005.

[ZZ00] J.-X. Zhou and W.-M. Zheng. A DAG-Based Partitioning-Reconfiguring Scheduling Algorithm in Network of Workstations. In *Proceedings 4th IEEE International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region (ICEPCAPR)*, volume 1, pages 323–326. IEEE, 2000.