A PEER TO PEER RESOURCE DISCOVERY SCHEME FOR PROVISIONING IN CLOUD

MD. RAYHANUR RAHMAN
Master of Science in Software Engineering,
Institute of Information Technology, University of Dhaka
Class Roll: MSSE 0102
Registration Number: HA 607
Session: 2008-09

A Thesis
Submitted to the Master of Science in Software Engineering Program Office
of the Institute of Information Technology, University of Dhaka
in Partial Fulfillment of the
Requirements for the Degree

Master of Science in Software Engineering

Institute of Information Technology
University of Dhaka
DHAKA, BANGLADESH

A PEER TO PEER RESOURCE DISCOVERY SCHEME FOR PROVISIONING IN CLOUD

MD. RAYHANUR RAHMAN

Approved:

Signature *Date*

Supervisor: Dr. Kazi Muheymin-Us-Sakib

Committee Member: Dr. Muhammad Mahbub Alam

Committee Member: Mohd. Zulfiquar Hafiz

Committee Member: Dr. Kazi Muheymin-Us-Sakib

Committee Member: Dr. Mohammad Shoyaib

To the people who are working hard to make this world a better place

Abstract

The ability to cater to large number of user demands without service unavailability has become the key to the success of Cloud computing. A fundamental challenge in building such large scale cloud based services is the scalability and fail safe techniques for discovering and provisioning of resources to meet expected Quality of Service (QoS). In the Cloud, existing resource provisioning models rely on centralized information services for resource discovery which are prone to single point failure as well as lack of scalability and fault-tolerance abilities.

In order to tackle these challenges, this thesis first proposes a resource discovery scheme based on structured Peer to Peer (P2P) architecture. As existing Distributed Hash Table (DHT) based P2P schemes inherently do not support multi attribute data publishing and range querying which are fundamental parts of obtaining information while making provisioning decision, the work also addresses this inability by proposing an attribute hub based data tuple indexing, routing and storing inside the nodes along with decentralized P2P oriented model for publishing of provisioning information, retrieval and fail safe policies for node join and leave. The thesis then presents a decentralized resource provisioning model using the aforementioned resource discovery scheme and utilizing Multi Attribute Utility Theory methods to make provisioning decisions in trade-off situations. Simulation shows that the proposed approach is 44.24% faster on average and 45.81% faster in the case of maximum number of VMs than the centralized and DHT based approach respectively in case of multidimensional range querying. The proposed system also shows less number of Service Level Objective (SLO) violations and migrations which are about 60.27% and 83.58% respectively. Thus the work demonstrates the effectiveness of the proposed decentralized approach for resource discovery and provisioning from the aspect of scalability and resilience to single point failure.

Acknowledgments

I would like to thank my supervisor Associate Professor Dr. Kazi M. Sakib for his support and guidance during my candidature. His constant encouragement was one of the main driving forces that kept me motivated. I would like to thank him for all his help with the research directions and experimentation. All his comments and suggestions were invaluable. His very demanding nature of supervising as well as pushing me to the limits have helped me grow as a researcher. I would like to thank my thesis reviewer Dr. Suraiya Pervin as well for her involvement in excellent feedbacks and suggestions to enhance this work further.

I would also like to thank my committee members for serving even at hardship. I also want to thank them for letting my defense be an enjoyable moment and for their brilliant comments and suggestions.

Also I would like to thank other faculty members for their participation and constructive feedback in the production of the thesis.

I am thankful to all other staffs at IIT, University of Dhaka for creating a pleasant work environment.

I am expressing ever gratefulness to all my fellow classmates whose advice, feedback and cooperation is truly incomparable.

I am also thankful to Ministry of Science and Technology, Government of the People's Republic of Bangladesh for granting me NST Fellowship 2013-14.

Finally, I am indebted to my parents for the many years of hard work and sacrifices they have made to support me. Without them, this thesis would never have started.

List of Publications

1. "ProvIntSec: a provenance cognition blueprint ensuring integrity and security for real life open source cloud," International Journal of Information Privacy, Security and Integrity (Inderscience), vol. 1, no. 4, pp. 360-380, 2013.

2. "A peer to peer resource provisioning scheme for cloud computing environment using multi attribute utility theory" in *Innovative Computing Technology (INTECH), 2013 Third International Conference*, London, United Kingdom, pp. 132-137, IEEE, 2013.

3. "IVRIDIO: Design of a software testing framework to provide Test-first Performance as a service" in *Innovative Computing Technology (INTECH), 2013 Third International Conference*, London, United Kingdom, pp. 520-525, IEEE, 2013.

4. "A Peer to Peer Resource Discovery Scheme for provisioning in Cloud Using Multi Attribute Range Query" in *International Conference on Electronics, Information and Vision (ICIEV)*, Dhaka, Bangladesh, 2014

Contents

Approval/Signature Page ii

Dedication iii

Abstract iv

Acknowledgments v

Publications vi

Table of Contents vii

List of Tables ix

List of Figures x

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Research Question . . . 3
- 1.3 Contribution of this Research . . . 4
- 1.4 Thesis Organization . . . 5

2 Background Study 7
- 2.1 Introduction . . . 7
- 2.2 An Overview of Cloud Computing . . . 7
- 2.3 Resource Provisioning in Cloud Computing Under the Hood . . . 11
- 2.4 Decision Making . . . 16
  - 2.4.1 Single Criteria vs. Multiple criteria Decision Making . . . 17
  - 2.4.2 Multi Attributed Decision Making Methods . . . 17
  - 2.4.3 Multi-Attribute Utility Theory (MAUT) Methods . . . 19
  - 2.4.4 Outranking Methods . . . 20
- 2.5 Distributed Hash Table and Consistent Hashing . . . 27
- 2.6 Resource Discovery and Load-Balancing Using DHT Overlay . . . 28
  - 2.6.1 Resource Discovery . . . 28
  - 2.6.2 Distributed Hash Tables . . . 29
  - 2.6.3 Designing Complex Services over DHTs . . . 30
- 2.7 Summary . . . 34

3 A Peer to Peer Resource Discovery Scheme in Cloud Using Multi Attribute Range Query 35
- 3.1 Introduction . . . 35
- 3.2 Related Work . . . 37

3.3 Proposed System Architecture of Resource Discovery 40
3.3.1 Data Model 40
3.3.2 Query Routing 41
3.3.3 Node Management 43
3.3.4 Provisioning Resource Discovery 44
3.4 Simulation Setup and Experimental Result Analysis 45
3.4.1 Simulation Setup 46
3.4.2 Experimental Result Analysis 47
3.5 Summary 52

4 A Peer to Peer Resource Provisioning Scheme for Cloud Computing Environment Using Multi Attribute Utility Theory 53
4.1 Introduction 53
4.2 Related Work 55
4.3 Overview of the Proposed Resource Provisioning 57
4.4 Proposed System Architecture for Decentralized Resource Provisioning 58
4.5 Provisioning Decision Making Model 62
4.6 Simulation Setup and Experimental Result Analysis 64
4.6.1 Simulation Setup 65
4.6.2 Experimental Result Analysis 67
4.7 Summary 70

5 Discussion and Conclusion 71
5.1 Introduction 71
5.2 A Peer to Peer Resource Discovery Scheme in Cloud Using Multi Attribute Range Query 71
5.3 A Peer to Peer Resource Provisioning Scheme for Cloud Computing Environment Using Multi Attribute Utility Theory 72
5.4 Future Direction 73

Bibliography 75

List of Tables

2.1 Multiple Criteria with weights 18

List of Figures

3.1 Data and Query Routing 42
3.2 Response Time vs. Number of VMs 48
3.3 Response Time vs. Epoch for a Fixed Number of VMs 49
3.4 Resource Mapping Delay Time vs. VM 50
3.5 Distribution Percentage of CPU Attribute vs. Value Range 51

4.1 NA Activity Flow in Each Loop 59
4.2 SLO Violation vs. VM 67
4.3 Migration vs. VM 68
4.4 Sprawling vs. VM 69

Chapter 1

Introduction

In the Cloud, efficient resource provisioning and management is a big challenge due to its dynamic nature and heterogeneous resource requirements. Here, requests from the end users come in and come out at any time and the magnitude of the workload and resource needs are unknown. Consequently, Cloud Service Providers (CSP) have to guarantee that there is no Service Level Objective (SLO) violation as well as ensure the maximum utilization of available resources, precise decision making regarding scaling up and down as well as enhancing overall responsiveness of the cloud services. In order to keep both CSP and consumers satisfied, effective resource provisioning is mandatory.

1.1 Motivation

Resource provisioning refers to the initialization, monitoring and controlling cloud resources. However, it is not a straightforward task to perform as there are some major aspects pertinent to the Cloud such as initial static launch of virtual machine (VM), VM allocation, migration, replication, monitoring and performance profiling etc. Not only that, horizontal and vertical scaling of VMs, nodes and memory, SLO optimization, forecasting potential resource demand, cost minimization by utilizing maximum resources are also important. Thus all of these aspects are quite interrelated, for example, satisfying SLO directly depends on VM allocation and migrations.

As the Cloud has become ready for mainstream acceptance, scalability [1] of services will come under more severe scrutiny due to the increasing number of online services in the Cloud and massive numbers of global users. Thus it is becoming ex-

ponentially difficult to manage resources in dynamic real time environment making scalable resource provisioning strategies urgent these days. The extent of scaling capability of a provisioning scheme heavily depends on the underlying resource discovery technique which refers to the procurement of datacenter metadata such as machine *id*s, VM usages, allocation information, status, availability etc. To overcome the aforementioned scalability challenges in addition to single point vulnerability issues, resource discovery should also be decentralized by nature to adaptively maintain and achieve the desired system wide connectivity and behavior.

Moreover, popular cloud infrastructures used in data center such as Eucalyptus [2], Openstack [3], CloudStack [4], Amazon EC2 [5] are all based on centralized control. That means, in a datacenter facilitated with these mentioned stacks either consists of only one node controller with numerous slave nodes or hierarchical clustering of node controllers. However, those schemes invite single point failure issues in the case of unexpected burst of workload or physical damage. Not only that, this single point dependency creates bottleneck in overall system performance. In order to avoid these problems, introduction of decentralization and parallelism is urgent in resource discovery and provisioning for cloud computing.

In response to these challenges, researchers have put a lot of efforts in this field. As a result, there are several dynamic and task specific provisioning algorithms in practical use (for example, Eucalyptus, OpenStack, CloudStack, Amazon EC2) which are based on single point resource discovery and decision making methods. These frameworks perform well in small to medium dimensioned datacenter but struggles in the performance and reliability section in case of large scale datacenter specific operations. Meanwhile, current enterprise and consumer market segments are constantly being attracted by the cloud services for the ability to deploy the application services without worrying about computational resource utilization and deployment

stack. So, in order to ensure maintaining service level agreements and availability of services, resource provisioning along with underlying resource discovery scheme must be invincible to scalability bottleneck and single point failure.

1.2 Research Question

Considering those issues mentioned above, the objective of this research is to develop a scalable resource provisioning scheme. In this research, this following research question will be answered,

How can a decentralized and scalable resource discovery scheme be developed for provisioning in Cloud so that cloud services can scale for a large datacenter configuration without the issue of single point vulnerability.

More specifically,

1. How Peer to Peer (P2P) scheme can be utilized in resource discovery for managing services in large cloud environments. As existing P2P approach are tailored to the storing and locating of contents indexed by their hash value, those approaches must be used in such a manner suitable for resource discovery. Hence, instead of locating the data indexed by a single string literal, those must provide a mechanism so that multi dimensional search, for example, *find a node which has the maximum CPU and IO usage*, can be answered. Apart from that, routing of provisioning data stored inside the nodes, their indexing and retrieval techniques and fail safe policies node join as well as leave need to be addressed as well.

2. Based on aforementioned resource discovery technique, how an effective resource provisioning scheme can be developed where provisioning decisions are made in decentralized manner. As major cloud frameworks are master-salve architecture

oriented and global cloud controller based, how P2P can replace this centralized architecture is the main question that needs to be answered. Moreover, how can every node in the datacenter maintain awareness of neighborhood nodes considering the lack of a global resource arbiter; how nodes can procure provisioning data with aforementioned proposed discovery scheme. Apart from these questions, decentralized provisioning decision making approach also needs to addressed.

1.3 Contribution of this Research

In answering these above research questions, this work contributes to scalable resource discovery scheme for provisioning in Cloud. These contributions are summarized as follows.

First, unlike traditional approaches, a decentralized resource discovery scheme has been proposed which is based on structured P2P network. The discovery scheme does not depend upon any global or hierarchal resource arbiter but allows publishing and querying provisioning data stored in a decentralized manner inside all the physical machines in the datacenter. The discovery scheme adopts P2P approach for routing data tuples and publishing them via storing those inside nodes, searching appropriate provisioning data stored in the nodes from the datacenter and fail-safe policies in case of physical machine failure. This technique answers the first question stated above. Result obtained from the simulation implies, the proposed discovery method features 44.24% faster on average and 45.81% faster response time in the case of largest datacenter setup than centralized and DHT based approaches respectively for discovering provisioning resources in the datacenter.

Second, a decentralized resource provisioning scheme has been proposed which is based on structured multi attribute range query P2P network [6]. Provisioning information from peer nodes are achieved via proposed resource discovery method which supports multi dimensional range queries. The provisioning scheme does not depend upon any global provisioning decision maker but delegates each node its own provisioning responsibility. It also uses Multi Attribute utility Theory (MAUT) methods [7] for allocating VMs into suitable physical machines (PMs) and VM migrations. Procured result demonstrates that the proposed system has shown less number of SLO violation and migration which is about 60.27% and 83.58% than the lone resource arbiter based provisioning scheme causing slightly lower resource utilization.

1.4 Thesis Organization

The rest of this thesis is organized as follows.

1. **Chapter 2: Background Study** In this chapter, overview of cloud computing, its classification, resource provisioning, Multi Attribute Utility Theory and discovery approach of datacenter information will be discussed

2. **Chapter 3: A Peer to Peer Resource Discovery Scheme in Cloud Using Multi Attribute Range Query** In this chapter, decentralized P2P based resource discovery approach for cloud provisioning will be presented

3. **Chapter 4: A Peer to Peer Resource Provisioning Scheme for Cloud Computing Environment Using Multi Attribute Utility Theory** In this chapter, scalable P2P based resource provisioning scheme for Cloud will be proposed which is built on top of aforementioned proposed resource discovery scheme

4. **Chapter 5: Discussion and Conclusion** In this chapter, retrospect on overall proposed approach for achieving a scalable resource discovery for cloud provisioning will be focused along with future research direction

Chapter 2

Background Study

2.1 Introduction

Cloud computing has introduced a new paradigm for developers, service providers and consumers in case of spreading out computing services to the mass people. For a better understanding regarding the environment, this chapter first introduces the reader with overall cloud computing scenarios, their classifications and appliances. One of the key factors in building an effective cloud computing platform is resource provisioning which is discussed followed by the overview of cloud. As resource provisioning in cloud needs to make critical decisions, multiple attribute decision making techniques has been discussed later. Finally, the discovery approach of datacenter information in order to provision the cloud is focused.

2.2 An Overview of Cloud Computing

Cloud computing is a new terminology in the modern era of technology which is a commercial realization of the evolution of computing services as utilities. It has made the software services more attractive as a total business solution because cloud computing just revolutionized the way hardware and software are designed, purchased and deployed. For example, developer these days need not worry about the issues regarding not only software deployment but also proper utilization of the hardware resources. These days IT industries can afford to compute huge data processing or to serve staggering amount of requests regardless of their magnitude and time. Thus Cloud offers scalability and cost optimization that is unprecedented in IT history.

Successful transformation of hardware/IT infrastructure, storages, application stacks and software from products to services can be considered as a remarkable achievement of the Cloud. From technical point of view, cloud computing might be called sister of grid computing but these following commercial aspects made cloud computing more attractive and revolutionary [8],

- Satisfying demand for computational resources on the fly. This results in elimination of pre-planning the provisioning of the hardware resources
- Enabling service providers to dynamically adjust their hardware resources according to the demand
- Introducing pay-per-use concept, which means resources should be used on a short term need basis according to a business plan. This process facilitates the optimal utilization of not only resources but also investment of the consumers

In principle, cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers providing those services. Thus, as a whole, the data center along with the hardware and software is what we will call the Cloud. The primary services provided by the Cloud are categorized into IaaS, PaaS and SaaS.

Infrastructure as a Service (IaaS)

IaaS refers to Infrastructure as a Service [9]. It is one of the most basic cloud service models where cloud providers offer computers, as physical or more often as Virtual Machines (VMs), and other intangible resources such as service. The VMs are run as guests OS by a hypervisor (it is simply a virtual machine manager) such as Xen or KVM. Management capability of VM pools by hypervisors supports large scale VM deployment to application environment. Other resources in IaaS clouds include images

in a VM image library, raw (block) and file-based storage, firewalls, load balancers, IP addresses, virtual local area networks (VLANs) and software bundles. IaaS cloud providers supply these resources on demand from their large pools installed in data centers. To deploy their applications, cloud users then install operating system images on the machine as well as their application software. In this model, it is the cloud user who is responsible for patching and maintaining the operating systems and application software. Cloud providers typically bill IaaS services on a utility computing basis, that is, cost will reflect the amount of resources allocated and consumed. IaaS refers not to a machine that does all the work, but simply to a facility given to businesses that offers users the leverage of extra storage space in servers and data centers. Examples of IaaS include Amazon EC2, Rackspace, Openstack, CloudStack etc.

Platform as a Service (PaaS)

PaaS refers to Platform as a Service. In this model, cloud providers deliver application stacks which include operating systems, programming language execution environments, databases and web servers. Application developers can develop and deploy their software solutions on a cloud platform but they do not need to worry about the cost and complexity of buying and managing the underlying hardware and software layers. PaaS has capability of dynamically scaling up and down the underlying computer and storage resources horizontally and vertically to match application demand such that cloud users do not have to allocate resources manually. Examples of PaaS include Cloud Foundry, Heroku, OpenShift, Google App Engine and Microsoft Azure etc.

Software as a Service (SaaS)

SaaS refers to Software as a Service. In this model, cloud providers install and operate application software in the cloud and cloud users access the software from

cloud clients. The users do not need to maintain infrastructure and platform regarding application environment. This eliminates the need to install and run the application on the cloud user's own computers simplifying maintenance and support. Single most notable feature of SaaS is its elasticity. This can be achieved by cloning tasks onto multiple virtual machines at run-time to meet the changing work demand [10]. Load balancers distribute the work over the set of virtual machines. This process is inconspicuous to the cloud user who sees only a single access point. Examples of SaaS include: Google Apps, Quickbooks Online and Microsoft Office 365.

Public cloud

Public cloud applications, storage and other resources are made available to the general public by a service provider. These services are free or offered on a pay-per-use model.

Community cloud

Community cloud shares infrastructure between several organizations from a specific community with common concerns such as security, compliance, jurisdiction etc. However, these are managed internally or by a third-party and hosted internally or externally.

Private cloud

Private cloud is cloud infrastructure operated solely for a single organization, whether managed internally or by a third-party and hosted internally or externally.

Hybrid cloud

Hybrid cloud is a composition of two or more clouds (private, community or public) that remain unique entities but are bound together, offering the benefits of multiple deployment models.

As a whole, the five characteristics that define the cloud computing are on demand self-service, broad network access, resource pooling, rapid scalability and measured service.

2.3 Resource Provisioning in Cloud Computing Under the Hood

These days, modern server side applications are hosted in cloud as it is easier to deploy, manage and implement the use of utility computing. This section provides an insight on further details of server side applications which are hosted in the cloud environment, workload characteristics, deployment strategies with their pros and cons and virtualization. It also highlights motivation for introducing cloud in mass scale, resource provisioning and decentralization in cloud computing.

Server Application

Server applications are programs which execute some operations and/or access data on the behalf of a user group usually referred to as clients. Most common example of a server is a simple web server. Main characteristics of server applications are that they are built with modularity in mind. Hence, a typical server application consists of three key tiers named presentation tier, business logic tier and storage tier. As server applications grow in complexity by offering different services to clients and grow in size by serving thousands of requests per second, the diversity and the number of server-side tiers also increases.

Performance Measurement

Load of a server side application is usually determined by its workload. Workload denotes a set of parameters and their values which contains various information of the application over a time interval. In principle, it denotes the intensity of request from clients. Whenever, the request is constant with very little fluctuation, the workload is called static. However, when there is a high variance in workload, it is called dynamic.

Performance of a server application is usually measured by throughput and response time. Throughput means the number of completed requests per time interval and response time means the time elapsed between the arrival of a client request to the server and the server's response to the client. In real life scenario, it is quite important to maintain a certain level of performance. In fact, there are dedicated contracts, called Service Level Agreements (SLAs, called as Service Level Objectives or SLO as well) that denote the Quality of Service (QoS) level the server should provide to its clients. The QoS is expressed by a number of performance/workload metrics and their appropriate values.

Workload Characteristics

From the observation, web servers have to handle dynamic, diversified and bursty workload. This happens due to variation of arrival rates and request types over time. Additionally, extreme server loads also occur in the form of flash crowds where an unusual increase in the number of clients causes unique resource demands at the server side. Some flash crowds have been observed because of very popular, known in advance events, such as the FIFA World Cup or the Olympic Games.

Resource Management

Resource Provisioning means provisioning of computational resources such as CPU time, memory, disk and network bandwidth. In order to meet its QoS performance

goals in the presence of time-varying workloads, it is one of the most important tasks in cloud computing.

Planning of a resource provisioning scheme mainly involves two steps: workload characterization and system modeling. First one is derived from the incoming request patterns while the second one is derived from the application demand. In particular, system modeling is a process of associating server operations with various performance metrics such as response time, throughput etc. Together, system modeling and workload characterization provide a thorough view of the server's performance and identify the major contributing factors.

Resource provisioning can be achieved either in proactive or reactive manner. Proactive allocation means all resources are provided in advance of predictable workload changes. Future demands can be forecasted so workload characterization and system modeling can be derived in advance. However, accurate proactive decision is not always possible. Resource provisioning can also be done in a reactive manner which involves resources being updated after a workload change. In both cases, the main goal is to minimize the deviation from desired QoS. In addition, resource provisioning hugely depends on workload patterns. It is usually more complicated when the workload pattern is very dynamic as, in this situation, no static allocation technique is useful.

Deployment

Back in 1970, applications were deployed in mainframe. But as the hardware cost fell rapidly and capability rose exponentially, commodity server machines were used as a group to host applications with diversified computational and storage demands. However, modern server applications are so complex that server machines are usually deployed in dedicated places with lots of extra facilities such as cooling. These ded-

icated places were called data center. These days, different hosting models arrived for executing applications in modern data centers. There are two different types of hosting platforms. The first one is dedicated hosting where disjoint sets of machines are dedicated to different applications. The main drawback of dedicated hosting is under-utilization of computational resources. In some particular cases, dedicated hosting cannot provide sufficient resources to hosted applications. To overcome this limitation, shared hosting technique is used where applications are co-located on the same machines and share physical resources. Although, shared hosting utilizes more resources compared to dedicated one, it also complicates the management situation with increase of application size and workload variance.

In order to overcome this problematic situation, server virtualization is considered as a means to combine the advantages of both dedicated and shared hosting: on one hand benefiting from performance isolation and on the other increasing the resource utilization. It is a technique to transform physical resources into a set of logical resources that can be used by applications in the same manner as using physical ones. There are three basic functionalities prominent in a virtualized environment. These are Virtual Machine controls, resource management and migration.

The virtual machine control functionalities include the process of creating, deleting, pausing and resuming VMs on demand. When a new VM is created, a new execution environment is also created as new operating system instance running with it. Moreover, a subset of physical resources is allocated to that instance. This is exactly equivalent to a new server machine which can be configured on demand. When the VM is paused, all applications running inside are also paused. Although, when it is paused, it is still holding the resources. Hence, the VM can be shut down to free up those resources which is identical to the scenario of terminating a server machine.

Resource management actions involve the process of specifying the allocation of computational resources during the creation of the VM. However, initial allocation can be changed throughout the lifetime of the VM. It is possible to reconfigure a running VM online to a new memory allocation, CPU sharing policy, disk space and network allocation.

Migration tasks denote transferring an existent VM from one physical host to another. During this process, VM is temporarily stopped, snapshotted, moved and then resumed on the new host. A snapshot is the state of a virtual machine and generally, its storage devices at an exact point in time. Migration process is also called teleportation.

Provision of adequate resources for VMs is critical for high-performance data centers. On the other hand, it is very important for the applications hosted inside the VMs to always have the resources necessary to achieve their performance goals. However, resource provisioning in server virtualized applications is a critical task. This following example can describe the situation.

Suppose, there are two applications with a single server machine. Assume that each application has a workload with known resource requirements and the sum of resources from both applications does not exceed the total available resources for the server machine. Hence, two VMs (VM A and VM B) are created with proper configuration and thus, both applications are served and the total resource utilization of the physical machine is now increased by augmenting the number of running servers. Let a scenario where workloads of both application change. In VM A, it increases while in VM B, it decreases. As the configurations of these two VMs are not changed, VM A suffers from under provisioning as it needs more resources while VM B suffers from over provisioning as it is holding up resources that it does not need. Thus, resource allocation scheme should be smart enough to respond to such scenarios.

From technical point of view, cloud computing enables the opportunity to run server side applications in virtualized environment with proper metering with associated business value. Thus, enterprise and consumers are served in *pay as you use* scheme. As the use of the Cloud rises exponentially at the end user level, smart resource provisioning and decentralization of control are the most critical challenges in cloud computing these days.

2.4 Decision Making

Decision making is the study of identifying and choosing alternatives based on the values and preferences of the decision maker [11]. Making a decision implies that there are alternative choices to be considered and in such a case it is not only expected to identify as many of these alternatives as possible but also to choose the one that best fits with goals, objectives, desires, values and so on. A general decision making process can be divided into the following 8 steps [12]:

1. define the problem
2. determine requirements
3. establish goals
4. identify alternatives
5. define criteria
6. select a decision making tool
7. evaluate alternatives against criteria
8. validate solutions against problem statement

2.4.1 Single Criteria vs. Multiple criteria Decision Making

It is very important to make distinction among the cases whether we have a single or multiple criteria. A decision making problem may have a single criterion or an aggregate measure like cost. Then the decision can be made implicitly by determining the alternative with the best value of the single criterion or aggregate measure. Then the classic form of an optimization problem is reached [13]. Here the objective function is the single criterion and the constraints are the requirements on the alternatives. Depending on the form and functional description of the optimization problem, different optimization techniques can also be used such as linear programming, nonlinear programming, discrete optimization etc.

2.4.2 Multi Attributed Decision Making Methods

Let's consider a multi-attribute decision making problem with m criteria and n alternatives. Let $C_1, \ldots, C_m$ and $A_1, \ldots, A_n$ denote the criteria and alternatives, respectively. A standard feature of multi-attribute decision making methodology is the decision table 2.1 as shown here. In the table each row belongs to a criterion and each column describes the performance of an alternative. The score a_{ij} describes the performance of alternative A_j against criteria C_i. For the sake of simplicity it is assumed that a higher score value means a better performance since any goal of minimization can easily be transformed into a goal of maximization.

As shown in decision Table 2.1, weights $w_1, \ldots, w_m$ are assigned to the criteria. Weight w_i reflects the relative importance of criteria C_i to the decision and is assumed to be positive. The weights of the criteria are usually determined on subjective basis.

		x_1	...	x_n
		A_1	...	A_m
w_1	C_1	a_{11}	...	a_{1n}
...	...	...	...	...
w_m	C_m	A_{m1}	...	A_{mn}

Table 2.1: Multiple Criteria with weights

They represent the opinion of a single decision maker or synthesize the opinions of a group of experts using a group decision technique as well.

The values $x_1, \ldots, x_n$ associated to the alternatives in the decision table 2.1 are used in the MAUT methods and are the final ranking values of the alternatives. Usually, higher ranking value means a better performance of the alternative, so the alternative with the highest ranking value is the best of the alternatives.

Multi-attribute decision making techniques can partially or completely rank the alternatives. For example, a single most preferred alternative can be identified or a short list of a limited number of alternatives can be selected for subsequent detailed appraisal. The two main families in the multi-attribute decision making methods are those based on the Multi-Attribute Utility Theory (MAUT) and Outranking methods.

The family of MAUT methods consists of aggregating the different criteria into a function, which has to be maximized. Thereby the mathematical conditions of aggregations are examined. This theory allows complete compensation between criteria, i.e. the gain on one criterion can compensate the loss on the another [14].

Meanwhile, the concept of outranking was proposed by Roy [15]. The basic idea is such that alternative A_i outranks A_j if on a great part of the criteria A_i performs at least as good as A_j (concordance condition), while its worse performance is still acceptable on the other criteria (non-discordance condition). After having determined for each pair of alternatives whether one alternative outranks another, these pair wise outranking assessments can be combined into a partial or complete ranking.

Contrary to the MAUT methods, where the alternative with the best value of the aggregated function can be obtained and considered as the best one, a partial ranking of an outranking method may not render the best alternative directly. A subset of alternatives can be determined such that any alternative not in the subset will be outranked by at least one member of the subset. The aim is to make this subset as small as possible. This subset of alternatives can be considered as a shortlist, within which a good compromised alternative should be found by further considerations or methods.

2.4.3 Multi-Attribute Utility Theory (MAUT) Methods

In most of the approaches based on the Multi-Attribute Utility Theory (MAUT), the weights associated to the criteria can properly reflect the relative importance of the criteria only if the scores a_{ij} are from a common, dimensionless scale [16]. The basis of MAUT methods is the use of utility functions. Utility functions can be applied to transform the raw performance values of the alternatives against diverse criteria, both factual (objective, quantitative) and judgmental (subjective, qualitative), to a common, dimensionless scale. In the practice, the interval $[0, 1]$ or $[0, 100]$ is used for this purpose. Utility functions play another very important role as they convert the raw performance values so that a more preferred performance obtains a higher utility value. A good example is a criterion reflecting the goal of cost minimization. The associated utility function must result in higher utility values for lower cost values.

It is common that some normalization is performed on a nonnegative row in the matrix of the a_{ij} entries. The entries in a row can be divided by the sum of the entries in the row, by the maximal element in the row or by a desired value greater than

any entry in the row. These normalizations can also be formalized as applying utility function.

Simple Multi-Attribute Ranking Techniques (SMART)

Simple Multi-Attribute Ranking Techniques (SMART) [17] is the simplest form of the MAUT methods. The ranking value x_j of alternative A_j is obtained simply as the weighted algebraic mean of the utility values are associated to it.

$$x_j = \frac{\sum_{i=1}^{m} w_i a_{ij}}{\sum_{i=1}^{m} w_i}; j = 1, ..., n \tag{2.1}$$

2.4.4 Outranking Methods

The principle of outranking methods assumes data is broadly similar to the data required for the MAUT methods [18, 19]. That is, they require alternatives and criteria to be specified and use the same data of the decision table 2.1, namely the a_{ij}'s and w_i's. In this section two most well-known outranking methods are discussed.

PROMETHEE

The decision table 2.1 is the starting point of the PROMETHEE methodology introduced by Brans and Vincke et al. [20] The scores a_{ij} need not necessarily be normalized or transformed into a common dimensionless scale. It is only assumed that, for the sake of simplicity, a higher score value means a better performance. It is also assumed that the weights w_i of the criteria have been determined by an appropriate method. Furthermore it is assumed that,

$$\sum_{i=1}^{m} w_i = 1 \tag{2.2}$$

In order to take the deviations and the scales of the criteria into account, a preference function is associated to each criterion. For this purpose, a preference function $P_i(A_j, A_k)$ is defined representing the degree of the preference of alternative A_j over A_k for criterion C_i. It is considered that the degree is in normalized form so that $0 \leq P_i(A_j, A_k) \leq 1$ and

- $P_i(A_j, A_k) = 0$ means no preference or indifference
- $P_i(A_j, A_k) \approx 0$ means weak preference
- $P_i(A_j, A_k) \approx 1$ means strong preference
- $P_i(A_j, A_k) = 1$ means strict preference

In most practical cases $P_i(A_j, A_k)$ is function of the deviation $d = a_{ij} - a_{ik}$ when $P_i(A_j, A_k) = p_i(a_ij, a_ik)$ where p_i is a non-decreasing function. It is assumed that,

$p_i(d) = 0 \ for \ d \leq 0 \ and \ 0 \leq p_i(d) \leq 1 \ for \ d > 0.$

A set of six typical preference functions was proposed by Brans and Vincke and Brans et al.[20]. The simplicity is the main advantage of these preferences functions such as no more than two parameters in each case, each having a clear economical significance.

A multicriteria preference index $\pi(A_j, A_k)$ of A_j over A_k can then be defined considering all the criteria,

$$\pi(A_j, A_k) = \sum_{i=1}^{m} w_i P_i(A_j, A_k) \tag{2.3}$$

This index also takes values $[0, 1]$, and represents the global intensity of preference between the couples of alternatives.

In order to rank the alternatives, the following precedence flows are defined in the following equations 2.4 and 2.5,

Positive outranking flow,

$$\phi^+(A_j) = \frac{1}{n-1}\sum_{k=1}^{n}\pi(A_j, A_k) \tag{2.4}$$

Negative outranking flow,

$$\phi^-(A_j) = \frac{1}{n-1}\sum_{k=1}^{n}\pi(A_j, A_k) \tag{2.5}$$

The positive outranking flow expresses how much each alternative is outranking all the others. The higher $\phi^+(A_j)$ the better the alternative. $\phi^+(A_j)$ represents the power of A_j which is called its outranking character.

The negative outranking flow expresses how much each alternative is outranked by all the others. The smaller $\phi^-(A_j)$ is, the better the alternative. $\phi^-(A_j)$ represents the weakness of A_j, its outranked character.

Meanwhile, A_j is preferred to A_k when $\phi^+(A_j) \geq \phi^+(A_k), \phi^-(A_j) \leq \phi^-(A_k)$ and at least one of the inequalities holds as a strict inequality.

- A_j and A_k are indifferent when $\phi^+(A_j) = \phi^+(A_k)$ and
- A_j and A_k are incomparable otherwise

In this partial ranking some couples of alternatives are comparable, some others are not. This information can be useful in concrete applications for decision making. Thus PROMETHEE I method outputs partial rankings.

However, if a complete ranking of the alternatives is requested by the decision maker, avoiding any incomparabilities, the net outranking flow can be considered by,

$$\phi(A_j) = \phi^+(A_j) - \phi^-(A_j) \tag{2.6}$$

Then the complete ranking is defined by,

- A_j is preferred to A_k when $\phi(A_j) > \phi(A_k)$
- A_j and A_k are indifferent when $\phi(A_j) = \phi(A_k)$

All alternatives are now comparable, the alternative with the highest $\phi(A_j)$ can be considered as best one. This is PROMETHEE II method which completely orders the alternative on the basis of outranking relations. Nonetheless, a considerable part of information gets lost by taking the difference of the positive and negative outranking flow.

ELECTRE

The simplest method of the ELECTRE family is ELECTRE I [19]. The ELECTRE methodology is based on the concordance and discordance indices defined as follows. Starting from the data of the decision matrix it is assumed here that the sum of the weights of all criteria equals to 1. For an ordered pair of alternatives (A_j, A_k), the concordance index c_{ik} is the sum of all the weights for those criteria where the performance score of A_j is least as high as that of A_k

$$c_{ij} = \sum_{i:a_{ij} \geq a_{ik}} w_i; j, k = 1, ..., n; j \neq k \tag{2.7}$$

Clearly, the concordance index lies between 0 and 1.

Meanwhile, the computation of the discordance index d_{jk} is a bit more complicated. Here, $d_{jk} = 0$ if $a_{ij} > a_{ik}$ where $i = 1, ..., m$ denoting that the discordance index is 0 if A_j performs better than A_k on all criteria. Otherwise,

$$d_{jk} = \max_{i=1,\ldots,m} \frac{a_{ik} - a_{ij}}{\max_{j=1,\ldots,n} a_{ij} - \min_{j=1,\ldots,n} a_{ij}} \tag{2.8}$$

For each criterion where A_j is outperformed by A_k, the ratio is calculated between the difference in performance level between A_j and A_k and the maximum difference in score on the criterion concerned between any pair of alternatives. The maximum of these ratios (which must lie between 0 and 1) is the discordance index.

Then, a concordance threshold c^* and discordance threshold d^* are defined such that $0 < d^* < c^* < 1$. Consequently, A_j outranks A_k if the $c_{jk} > c^*$ and $d_{jk} < d^*$.

This outranking defines a partial ranking on the set of alternatives. Consider the set of all alternatives that outranks at least one other alternative and are themselves not outranked. This set contains the promising alternatives for this decision problem. Interactively changing the level thresholds, we can also change the size of this set.

Having outranking relations, which can be represented by a digraph, a subset of alternatives is sought such that

- any alternative which is not in the subset is outranked by at least one alternative of the subset
- the alternatives of the subset are incomparable

This type of set is called a kernel of the graph. However, if the graph has no cycle, the kernel exists and is unique. Each cycle can be replaced by a unique element.

Meanwhile, ELECTRE I outputs a partial outranking which is formed from a pair of two alternative where one outranks another. In order to get the complete outranking, ELECTRE II method is used. The basic steps are as simple as ELECTRE I method,

- two concordance thresholds and a discordance threshold (or a discordance set) are defined

- a strong outranking relation S^F and a weak outranking relation S^f are built
- a complete preorder is obtained by calculating the degrees of the graph's vertices (based on S^F)
- ties are eliminated on the basis of S^f

The degree of an alternative p is represented by a vertex denoted by $d(p)$ which means the difference between the number of alternatives which are strongly outranked by the alternative and the number of alternatives which strongly outrank that alternative.

TOPSIS

The Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) method is a technique for order preference by similarity to ideal solution and proposed by Hwang and Yoon [21]. The ideal solution (also called the positive ideal solution) is a solution that maximizes the benefit/criteria/attributes and minimizes the cost /criteria/attributes, whereas the negative ideal solution (also called the anti-ideal solution) maximizes the cost criteria/attributes and minimizes the benefit criteria/attributes. The so-called benefit criteria/attributes are those for maximization, while the cost criteria/attributes are those for minimization. The best alternative is the one that is closest to the ideal solution and farthest from the negative ideal solution.

The TOPSIS process is carried out as follows:

1. Create an evaluation matrix consisting of m alternatives and n criteria, with the intersection of each alternative and criteria given as x_{ij}, we therefore have a matrix $(x_{ij})_{m\times n}$.
2. The matrix $(x_{ij})_{m\times n}$ is then normalized to form the matrix $R = (r_{ij})_{m\times n}$, using

the normalization method $r_{ij} = x_{ij}/pmax(v_j), i = 1, 2, ..., m, j = 1, 2, ..., n$, where $pmax(v_j)$ is the maximum possible value of the indicator $v_j, j = 1, 2, ..., n$.

3. Calculate the weighted normalized decision matrix $T = (t_{ij})_{m\times n} = (w_j r_{ij})_{m\times n}, i = 1, 2, ..., m$ where $w_j = W_j/\sum_{j=1}^{n} W_j, j = 1, 2, ..., n$ so that $\sum_{j=1}^{n} w_j = 1$, and W_j is the original weight given to the indicator $v_j, j = 1, 2, ..., n$.

4. Determine the worst alternative (A_w) and the best alternative (A_b) as following,

   $A_w = \{\langle max(t_{ij}|i = 1, 2, ..., m)|j \in J_-\rangle, \langle min(t_{ij}|i = 1, 2, ..., m)|j \in J_+\rangle\} \equiv \{t_{wj}|j = 1, 2, ..., n\}$,

   $A_b = \{\langle min(t_{ij}|i = 1, 2, ..., m)|j \in J_-\rangle, \langle max(t_{ij}|i = 1, 2, ..., m)|j \in J_+\rangle\} \equiv \{t_{bj}|j = 1, 2, ..., n\}$,

   where, $J_+ = \{j = 1, 2, ..., n|j\}$ associated with the criteria having a positive impact, and $J_- = \{j = 1, 2, ..., n|j\}$ associated with the criteria having a negative impact

5. Calculate the distance between the target alternative i and the worst condition A_w:

   $d_{iw} = \sqrt{\sum_{j=1}^{n}(t_{ij} - t_{wj})^2}, i = 1, 2, ..., m$ and the distance between the alternative i and the best condition A_b

   $d_{ib} = \sqrt{\sum_{j=1}^{n}(t_{ij} - t_{bj})^2}, i = 1, 2, ..., m$ where d_{iw} and d_{ib} are distances from the target alternative i to the worst and best conditions, respectively.

6. Calculate the similarity to the worst condition: $s_{iw} = d_{ib}/(d_{iw} + d_{ib}), 0 \leq s_{iw} \leq 1, i = 1, 2, ..., m$. s_{iw} =1 if and only if the alternative solution has the worst condition; and $s_{iw} = 0$ if and only if the alternative solution has the best condition.

7. Rank the alternatives according to $s_{iw}(i = 1, 2, ..., m)$.

2.5 Distributed Hash Table and Consistent Hashing

A Distributed Hash Table (DHT) [22] is a well-known example of decentralized distributed systems. It facilitates a lookup service identical to a hash table. Like typical hash tables key and value pairs are stored in a DHT network and any member node can efficiently retrieve the value associated to a given key. Maintenance of these key and value pairs is distributed among the nodes in such a way so that any change in the participating member nodes causes minimal impacts. This enables a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

The foundation of a DHT is an abstract keyspace which is typically 160 bit long string. This space is distributed among the participating nodes. To store an entity (usually a file) with a given key, hash value of the attribute of that entity (usually filename) is generated and mapped to the keyspace and then sent to the owner of that keyspace. Retrieving of the entity is simply a straightforward process identical to the storing mechanism.

In order to maintain the nodes, variants of consistent hashing [23] is used. Consistent hashing is of special type as it needs only $\frac{k}{n}$ number of remapping on average where, k is the number of keys and n is the number of buckets while in typical hash algorithms, all keys need to be remapped once the number of buckets change. It is based on mapping objects to a point on the edge of a circle. In order to determine the location of an object in the system, first it is needed to find the object's key on the circle edge and then walk around the circle until encountering the first bucket. Thus, each bucket contains all the resources located between its point and the next bucket point. In case of a bucket being unavailable, the objects which are mapped to that

missing bucket will be reassigned to the next bucket walking around the circle edge. However, unlike traditional hash table systems, values mapping to other buckets will still do so and do not need to be moved. The addition of a new bucket in the system is handled in the same way.

Thus DHT has the essential property that removal or addition of one node changes only the set of keys owned by the nodes with adjacent IDs and leaves all other nodes unaffected. In contrast with a traditional hash table, addition or removal of one bucket causes nearly the entire keyspace to be remapped.

2.6 Resource Discovery and Load-Balancing Using DHT Overlay

Resource discovery refers to the identification and maintenance of the information needed to keep provisioning functioning in the datacenter. In this section, know how's of resource discovery in cloud will be discussed along with existing policies of discovery of resources in datacenter.

2.6.1 Resource Discovery

Resource provisioning information usually contains current state of physical and virtual machines, their resource usage, Service Level Objective parameters and violation information. In a datacenter, each of the physical machine has their own set of provisioning information. These information are necessary for any provisioning decision making, for example, virtual machine migration. Such decision making process involves obtaining knowledge about other physical machines in the datacenter. This is why, all provisioning information of all the nodes in the datacenter should be

published periodically in a quick accessible manner so that they can be discovered whenever needed.

Scalability and resilience to single point failure of provisioning in cloud solely depends on how underlying resource discovery approach scales and tackles node failures. Traditionally, these provisioning resources can be obtained from a global or hierarchical resource arbiter which provides all sorts of provisioning information to corresponding nodes [5, 24, 25]. It can be claimed that none of such centralized approaches is an appropriate solution to this purpose, due to concerns of scalability and single point vulnerability arising from the large queues of provisioning queries [26, 27, 28]. Decentralized Peer to Peer (P2P) approaches have also been addressed in this context using Distributed Hash Table overlay network which is discussed below.

2.6.2 Distributed Hash Tables

Structured P2P architecture such as DHTs facilitate deterministic query search results within logarithmic upper bounds on network messaging complexity. Peers in DHT based P2P schemes such as Chord, CAN, Pastry, and Tapestry keep an index for $O(\log n)$ peers where n is the total number of peers in the system. These following issues are inherent to the design of DHT, [29]: (i) generation of node ids and object ids, namely keys using cryptographic/randomized hash functions for example, SHA-1 [29, 30, 31, 32] the objects and nodes are mapped on the overlay network based on their key value and each node is assigned accountability for managing a small number of objects, (ii) building up routing information at various nodes in the network where each node maintains the neighborhood information of a few other nodes in the network and (iii) an efficient look-up query resolution scheme.

Whenever a node in the overlay network gets a query request, it need to resolve it within acceptable upper computational bound such as in $O(\log n)$ routing hops. This is achieved by re-routing the query request to the nodes that are most likely to store the information of the desired object. Such potential nodes are discovered by using the routing table entries. Although at the core numerous DHT based schemes (Chord, CAN, Pastry, and Tapestry, etc.) are identical, still there lies a substantial differences in the real implementation of algorithms including the overlay network build up, network graph architecture, routing table maintenance and node join/leave management. The performance metrics for evaluating a DHT include fault-tolerance, load-balancing, efficiency of look-ups and inserts, and proximity awareness[33].

2.6.3 Designing Complex Services over DHTs

As traditional DHT schemes are inherently unable to answer multi dimensional query, this section discusses the limitation followed by the data indexing and routing techniques.

Limitations of Basic DHT Implementations and Query Types

Traditionally, DHTs have been efficient for single-dimensional queries such as *finding all resources that match the given attribute value.* Since Cloud computing IaaS and PaaS level services such as servers, VMs, enterprise computers (private cloud resources), storage devices and databases are queryable by several attributes, a search query for those services is always multidimensional. These dimensions can include processor speed, available memory, architecture, service type, installed operating system and network bandwidth. In Amazon EC2 CloudWatch service, each Amazon

Machine Image (AMI) instance has seven performance metrics and four dimensions associated to it, obtained from [34]. Additionally, these AMIs can host different application service types such as web hosting, content-delivery, social networking and high-performance computing, that have dynamic request arrival, access and distribution patterns. These types of application services are dependent on the business needs and scientific experiments. In these cases, a Cloud resource discovery query, issued by provisioning scheme will combine the aforementioned attributes related to AMI instances and application service types and therefore can have the following semantics:

Cloud Service Type = web hosting && *Host CPU Utilization* $< 50\%$ && *Instance OS Type = WinSrv2003* && *Host Processor Cores* > 1 && *Host Processors Speed* $> 1.5GHz$ && *Host Cloud Location = Europe*

Meanwhile, VMs deployed in the Cloud have to publish their current information so that provisioning scheme can query and discover them. VM instances update their configuration, status and the deployed availability by sending update query to the DHT overlay. An update query has the following semantics:

Cloud Service Type = web hosting && *Host CPU Utilization* $= 30\%$ && *Instance OS Type = WinSrv2003* && *Host Processor Cores* $= 2$ && *Host Processors Speed* $= 1.5GHz$ && *Host Cloud Location = Europe*

Enhancing DHTs to support indexing and matching multidimensional range queries or update queries as well as to index all resources whose attribute value overlaps a given search space, is a complex problem. Multidimensional range queries are dependent on ranges of values for dimensions rather than on specific value. Compared to one-dimensional queries, resolving multidimensional queries is far more complicated because there is no specific total ordering of the points in the dimension space. In addition, the query interval has dynamic size, aspect ratio, and position such as a

window query. The main challenges is to facilitate multidimensional queries in a DHT overlay include designing efficient service attribute data: (i) distribution or indexing techniques and (ii) query routing techniques.

Data Indexing Techniques for Mapping Multidimensional Range and Point Queries

A data indexing mechanism partitions the multidimensional space of attributes into the set of VMs in a DHT overlay network. Efficiency of this distribution scheme directly dictates how the query processing load is distributed among the peers. Such a good distribution technique should have the following properties [35]: (i) locality: data located nearby in the attribute space should be mapped to the same peer, hence lessening the distributed lookup complexity, (ii) load balance: the number of data points indexed by each peer should be approximately the same to ensure uniform distribution of query processing, (iii) minimal metadata: prior information needed for mapping the attribute space to the overlay should be minimal and (iv) minimal management overhead: during VM initialization and destruction task, update policies such as the migration of data points to a fresh peer should cause insignificant traffic.

There are several types of database indices [27] that can manage mapping of multi-dimensional objects such as the space filling curves, k-d tree, MX-CIF Quad tree, and R*-tree in a DHT overlay. These indices are referred to as spatial indices [36] in the literature. Spatial indices are well tailored to the need for managing the complexity of multidimensional queries. Although some spatial indices can have issues regarding the case of routing load-balancing of skewed datasets, all of those are generally scalable in terms of the number of nodes traversed and messages generated during searching and routing multidimensional resource discovery and update queries. However, there

are different tradeoffs involved with each of these spatial indices but basically all support scalability and resource discovery. Some spatial index would perform optimally in one scenario but the performance could degrade if the attribute/data distribution changed significantly.

Routing Techniques for Handling Multidimensional Queries in DHT Overlay

DHTs guarantee deterministic query with logarithmic upper bounds on network messaging cost for one dimensional queries. However, resource discovery and update query in Cloud are multidimensional as discussed in previous sections. Hence, the existing DHT routing schemes need to be improved in order to efficiently answer multidimensional queries. Several data structures based on overlay types effectively create a logical multidimensional index space over a DHT overlay. A look-up process involves searching for an index or set of indexes in a multidimensional space. However, the exact query routing path is directly dictated by the data distribution mechanism. In this context, various approaches have proposed different routing/indexing heuristics based on tree based architectures. However, the root VM presents a single point of failure and load imbalance [35, 26]. To overcome this, all the query processing and the data storage should start at that minimal level of the tree rather than at the root. There are a number of techniques available for distributed routing in multidimensional space. The performance of schemes varies based on the distribution of data in the multidimensional space and VM in the underlying DHT overlay.

2.7 Summary

In this section, cloud computing environment is discussed here in a nutshell. Not only that, but also the under the hood detail about decision making, resource discovery along with DHT based policies of such discovery are also highlighted. In principle, these highlighted topics will be really insightful to understand the proposed decentralized resource discovery and provisioning in cloud computing environment.

Chapter 3

A Peer to Peer Resource Discovery Scheme in Cloud Using Multi Attribute Range Query

3.1 Introduction

From technical aspects, virtualization is the key player in deploying dynamic number of applications in cloud computing environment. In such environments, numerous physical machines (PMs) host virtual machines (VMs) depending on its computational capacity. VMs are configured with proper environments in order to host the deployed applications. VMs are loosely coupled from PMs as well because of their ability of reconfiguration and migration from one PM to another. Such configuration and utilization of those computational resources in the datacenter with dynamic adjustment is called resource provisioning [37, 26]. It also scales up and out applications according to their resource needs and consumer provided constraints. The effectiveness of cloud computing depends on this process significantly because it manages all deployed applications intelligently to satisfy consumers through fulfilling their Service Level Objectives (SLO).

Discovery of resources regarding provisioning is critical to datacenter operations as it is necessary to know about the status of the nodes such as resource usage, VM allocation, deployed applications and SLO updates. Provisioning jobs such as VM allocation and migration require knowledge about neighbor nodes which can accommodate the aforementioned VMs. Such provisioning information are referred as resources throughout this thesis. Scalability and resilience to single point failure of provisioning in cloud solely depend on how underlying resource discovery approach scales and tackles node failures.

Traditionally, these provisioning resources can be obtained from a global or hierarchical resource arbiter which provide all sorts of provisioning information to the corresponding nodes [5, 24, 25]. It can be claimed that none of such centralized approaches is an appropriate solution to this purpose, due to concerns of scalability and single point vulnerability arising from the large queues of provisioning queries [26, 27, 28]. Peer to Peer (P2P) approach in resource discovery scheme could have been an answer to such problems. However, existing Distributed Hash Table (DHT) based P2P schemes such as Chord, CAN, Pastry inherently do not support multi attribute data publishing and range querying [29]. Therefore, architecting a P2P based decentralized resource discovery for provisioning in cloud is an open research question.

This chapter proposes a resource discovery scheme based on structured P2P network for provisioning based on multi attribute range query presented in [6]. This chapter also proposes P2P approach for routing data tuples and publishing those via storing inside nodes, searching appropriate provisioning data stored in the nodes from the datacenter and fail-safe policies in case of physical machine failure. According to P2P architecture, there is no dedicated client or server node in the network resulting in making the discovery scheme inherently decentralized and scalable.

For the simulation purpose, P2P overlay network environment was developed in C# programming language. On top of that, the proposed discovery scheme was implemented for provisioning in cloud. Then the approach was plugged in the simulation environment described in the work on P2P based resource provisioning in cloud [38]. The scheme was tested to observe their query response time and resource publish time for a large number of VMs which generated a lot of provisioning resource publish and query requests. Moreover, DHT based implementation proposed in [39] has also been simulated in the aforementioned testbed .The outcome demonstrated that the pro-

posed provisioning scheme generated a significantly less query response time which is about 44.24% and 45.81% in comparison with centralized and DHT implementation respectively on average. It also showed that the proposed approach demonstrates a logarithmic delay on resource publish time with increased number of virtual machines which is also 14.75% less than the DHT implementation. However slightly non uniform load balancing of resources was also observed compared to other two implementations. Such outcome proves that the proposed resource discovery scheme is better scalable than centralized architecture. However, it is also faster than the DHT based implementation paying the penalty of a slightly uneven data distribution.

3.2 Related Work

In this section, provisioning in current state of the art key players in cloud computing domain such as Amazon EC2 [40], Microsoft Azure [25], Google App Engine [41], Eucalyptus [2] and GoGrid [42] will be focused. A diversified number of built in services for monitoring, managing and provisioning resources are incorporated with those. Although the way these techniques are implemented in each of these clouds vary significantly, all of those are centralized in nature.

At present, Amazon Web Services (AWS) uses three centralized services for overall provisioning. Those are Elastic Load Balancer [24] for load balancing, Amazon CloudWatch [5] for monitoring and Amazon Auto-Scaling [43] for auto scaling purposes. Both Elastic Load Balancer and Auto-Scaling services depend on the information regarding resource status reported by the CloudWatch service. Elastic Load Balancer service automatically makes provisioning decisions involving incoming service workload across available Amazon EC2 instances. On the other hand, the Auto Scaling service is used to dynamically scale in or out Amazon EC2 VM instances in order

to handle changes in service demand. However, CloudWatch can only monitor the status information at VM level. In reality, a VM instance can host more than one services such as web server, database backend, image server, etc. Therefore, there is a critical requirement of monitoring, maintaining and searching the status of individual services in a scalable and decentralized manner.

Eucalyptus [2] is an open source cloud computing environment which is composed of three controllers namely Node, Cluster and Cloud Controller. Management of VMs on physical resources, coordinating load balancing decisions across nodes in same availability zone and handling connections from external clients as well as administrators are the responsibilities of these controllers. According to the current hierarchical design, Cloud Controller works at the root level, Cluster Controllers are the intermediate nodes and Node Controllers operate at the leaf level. However, the hierarchical design pattern might invoke performance bottleneck with the increase in service workload and system size from the network management perspective.

Windows Azure Fabric [25] is designed based on an architecture similar to an interconnected structure composed of servers, VMs and load balancers known as nodes and power units, Ethernet as well as serial communications known as edges. Applications created and deployed by the developers, hosted in the Azure Cloud are monitored, maintained and provisioned by a centralized service named Azure Fabric Controller (AFC). Management of all the software and hardware components in a Azure powered datacenter is the responsibility of AFC. These components include servers, hardware based load balancers, switches, routers, power-on automation devices etc. Multiple replicas of AFC are used for fail safe purposes and those are constantly synchronized so that information stored in all of the AFCs are consistent and integral. This redundancy design tecnique performs well for fail safe strategies in case of small size datacenters but is inherently unable to scale in large datacenter con-

figuration because, with the number of replication degree rising, it will be infeasible to maintain consistency among replicas of AFC.

GoGrid [42] Cloud uses centralized load balancers for managing application service traffic across servers. Round Robin algorithm and Least Connect algorithm are used for routing application service requests. The load balancer can also identify server crashes, which is tackled by rerouting future requests for the failed server to other available ones. However, the centralized architecture of the load balancer is vulnerable to single point failure and bottleneck in case of large number of provisioning requests.

Unlike other cloud platforms which provide a direct access to VMs to the developers, Google App Engine [41] offers a scalable but closed platform where applications can be run. Therefore, access to the underlying operating system is restricted from the developers in Google App Engine. The platform manages load balancing strategies, service provisioning and auto scaling automatically under the hood. At this moment, very little or no documentation has been found about inner details regarding architecture of this platform.

In [39], a service discovery and load balancing scheme for cloud provisioning has been proposed which uses DHT based structured P2P scheme named Chord. As DHT implementation does not support multi dimensional query lookup inherently, it utilizes MX-CIF region tree for indexing queries via control points which are distributed all over the nodes. As the scheme uses hashing techniques, it guarantees uniform load balancing leading to suffering from high response time in query lookup, update and publish due to hashing and control point managing overheads.

In principle, these aforementioned policies, most of which are closed source and proprietary as well, utilized single or hierarchal policies for resource discovery in provisioning. Hence, those are unable to offer a scalable and fail safe approach to obtain information for provisioning in cloud.

3.3 Proposed System Architecture of Resource Discovery

The focus of this section is to provide comprehensive details on proposed resource discovery scheme for provisioning in the Cloud. First the system and application models considered are described. Then the proposed P2P model that delivers reliable and scalable resource provisioning in cloud environment is presented based on these models. There are three key aspects of the scheme which will be discussed in the following sections. These are data model, query routing and node management. Finally, this proposed resource discovery architecture which is utilized in resource provisioning will be discussed.

3.3.1 Data Model

In the proposed scheme, provisioning status is represented as attribute-value pair which is similar to tuple of relational database management system. More specifically, each field is a tuple of this form: <*type, attribute, value*>. The model features four data types and these are *int, char, float* and *string*. Data queries are conjunction of predicates which are tuple of the form <*type, attribute, operator, value*>. The following predicates supported in the scheme are: $<, >, <=, =>, ==, != .$

For example, a physical machine has been operating on 65% CPU workload. This can be represented in the following manner, $< int, node - id, 1 > < float, cpu - usage, 0.65 >$

A query looking for a node having 30% CPU and 50% memory can be represented like this following, $< float, cpu-usage, 0.30 > \&\& < float, memory-usage, 0.50 >$. More example can be similar to this, *Cloud Service Type = web hosting && Host CPU*

Utilization $< 50\%$ && *Instance OS Type = WinSrv2003* && *Host Processor Cores* > 1 && *Host Processors Speed* $> 1.5GHz$ && *Host Cloud Location = Europe*

Cloud Service Type = web hosting && *Host CPU Utilization* $= 30\%$ && *Instance OS Type = WinSrv2003* && *Host Processor Cores* $= 2$ && *Host Processors Speed* $= 1.5$ *GHz* && *Host Cloud Location = Europe*

3.3.2 Query Routing

The nodes in the datacenter are partitioned into attribute hubs. This partition is logical only and that means a physical node can be part of multiple attribute hubs. Each hub is accountable for a specific data attribute in overall schema. For example, an attribute hub can be responsible for *node-id* attributes. Another attribute hub is responsible for *memory-usage* attribute. Thus, hubs can be imagined as orthogonal dimensions of a multi dimensional Cartesian space. The decision regarding which dimension to route through is determined by the first node in the neighborhood. The rest of the routing is one-dimensional and is based on the values of that particular attribute of the data item.

Let A denote the set of attributes in overall schema, A_q denotes the set of attributes existing in a provisioning query, A_d denotes the set of attributes in a provisioning information record and H_a denotes the attribute hub for attribute a. In the datacenter, PMs within an attribute hub are arranged in a circular overlay where each PM is responsible for a continuous range of that particular attribute. Thus a node is responsible for all the queries where the particular attribute is a member of A_q and query attribute range predicate is true for the range of that PM. The PM also stores all the data records where the particular attribute is a member of A_d. In addition to having a link to the predecessor and successor within its own hub, each

node must also maintain a link to each of the other hubs.

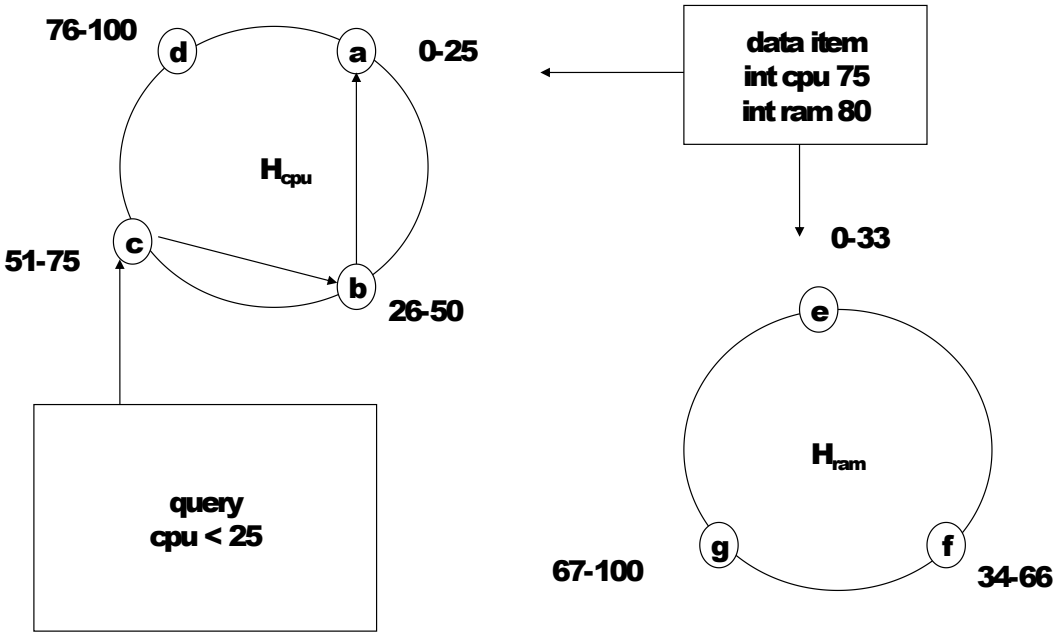

Figure 3.1: Data and Query Routing

Queries are sent to exactly one of the hubs that manages attributes matched with the query attributes. A query Q is routed to the attribute hub H_a where a is any attribute member of A_q. Inside that particular hub, the query is routed to all the nodes that could potentially have matching values. In order to ensure that the queries find all the matching data-records, during insertion, a data-record D is sent to all H_b where b is a member of A_d. This is done as the set of queries matching D can arrive in any of these attribute hubs. Inside the hub, the data record is routed to the responsible node for the value of the record.

Fig. 3.1 demonstrates the routing of queries and data-records. It denotes two attribute hubs namely H_{cpu} and H_{ram} which correspond to current CPU and RAM

usage of a physical node. The minimum and maximum values for the *cpu* and *ram* attributes are 0 and 100 respectively. The figure denotes that the ranges are being distributed to nodes. A provisioning data which contains the information of CPU usage of 75% and memory usage of 80% is sent to both H_{cpu} and H_{ram} and it is stored at both of the nodes c and g. The query which looks for data where CPU usage is below 25% enters H_{cpu} at node c and is routed at nodes b and a (destination).

3.3.3 Node Management

As datacenter environment is highly dynamic, physical machines in the datacenter might join and leave the datacenter network frequently. This is why the proposed scheme has to handle both the node join and leave scenarios without abrupting the datacenter operation. This section describes the detailed protocol used by nodes during join and departure.

Each node in the datacenter needs to construct and maintain the following set of links: a) successor and predecessor links within the attribute hub and b) one cross-hub link per hub for connecting to other hubs. The crosshub link implies that each node knows about at least one representative for every hub in the system. In order to recover during node departures, nodes keep a small number (instead of one) of successor/predecessor and cross-hub links.

Like most other distributed overlays, an incoming node needs information about at least one (or at most a few) node(s) that are already part of the routing system. This information can be obtained via a match-making server or any other out-of-band means. The incoming node then queries an existing node and obtains state about the hubs along with a list of representatives for each hub in the system. Then, it randomly chooses a hub to join and contacts a member m of that hub. The incoming

node then installs itself as a predecessor of m, takes charge of half of m's range of values and becomes a part of the hub circle.

To start with, the new node then copies the routing state of its successor m, including its links to nodes in the other hubs. At this point, it starts random-walks on each of the other hubs to obtain new cross-hub neighbors distinct from its successor's. It is noteworthy that these processes are not essential for correctness and only affect the efficiency of the routing protocol.

When nodes depart, the successor/predecessor links and the inter-hub links must be repaired. To repair successor/predecessor links, each node maintains a short list of contiguous nodes further clockwise on the ring than its immediate successor. When a node's successor departs, that node is responsible for finding the next node along the ring and creating a new successor link.

Finally, to repair a broken cross-hub link, a node considers the following three choices: a) it uses a backup cross-hub link for that hub to generate a new cross-hub neighbor, or b) if such a backup is not available, it queries its successor and predecessor for their links to the desired hub, or c) in the worst case, the node contacts the bootstrap server to query the address of a node participating in the desired hub.

3.3.4 Provisioning Resource Discovery

Resource provisioning information usually contains current state of physical and virtual machines, their resource usage, Service Level Objective parameters and violation information. In a datacenter, each of the physical machine has their own set of provisioning information. These information are necessary for any provisioning decision making, for example, virtual machine migration. Such decision making process involves obtaining knowledge about other physical machines in the datacenter. This is

why, provisioning information of all the nodes in the datacenter should be published so that they can be discovered whenever needed.

Provisioning information can be considered as data item D in the proposed scheme. These items are stored in the physical machine which matches the attribute hub and range of the attribute in the given data item. Meanwhile, for routing queries, it is routed to the first value appearing in the range and then the contiguity of range values is used to pass the query along the circle.

As cloud environment is highly dynamic, provisioning information of each physical machine changes with the passage of time. Hence, those information are needed to be frequently updated while phased out information should be invalidated. In order to do so, each physical machine publishes its provisioning information with a timestamp and epoch count in each epoch. On the other hand, each physical machine will delete the outdated provisioning information stored inside those. Each query will also have a timestamp and epoch number in order to prevent returning query matched data which are outdated.

3.4 Simulation Setup and Experimental Result Analysis

In this section, the focus will be put on how the proposed resource discovery scheme performs in comparison with the centralized and DHT based scheme. First, the simulation environment of the datacenter will be highlighted. Then simulation goal will be presented followed by the discussion on obtained results.

3.4.1 Simulation Setup

In the previous work[38], a P2P resource provisioning scheme for cloud using MAUT methods [44] was proposed where unstructured P2P scheme was utilized. In this work, the same simulated environment was used, which utilized the proposed resource discovery scheme instead of unstructured P2P network. The simulation platform was built using Common Language Runtime and C# programming language. The program focused on emulating a datacenter where there were numerous PMs forming a P2P structure. Each PM contained a fixed number of computational resources such as CPU, Memory, IO bandwidth which were required for executing several VMs inside the host. VMs were also characterized by their change of resource demand in CPU, memory and IO bandwidth with respect to time. These changes in the resource demands were generated following Gaussian, Poisson, uniform and burst statistical distributions [45].

As the production level datacenters deploy about 1 million physical machines in the datacenter, in this simulated environment, the VM counts are kept around 1×10^5 to emulate the scenario of a large datacenter[46, 47, 48]. However, for the procurement of the result, around 1×10^3 VMs are considered sufficient to understand the characteristics of the plotted graphs as well as the performance comparisons among the schemes under observation.

In order to do provisioning jobs, the nodes in the simulated environment published their provisioning status and query on provisioning information that were already published. Each node in the datacenter published their *id*, allocated virtual machine *ids*, total cpu, memory, IO, bandwidth and resource usage statistics of each of the virtual machine running inside the host. On the other hand, each of the node periodically updated their provisioning information in each epoch. They also made queries on provisioning information using the proposed resource discovery scheme. The queries

included both multi dimensional value and range matching such as *equal, between, greater than, less* than predicates as well, .

The proposed resource discovery scheme was then utilized in the aforementioned simulated environment. A single global resource arbiter based centralized resource discovery scheme was also implemented in order to compare to the proposed P2P based scheme. Finally, DHT based Cloud Peer, proposed in [39], has also been implemented in the proposed environment and then utilized to obtain resources needed for the provisioning. In the centralized scheme, all the nodes in the datacenter communicated to a single resource arbiter for querying information from neighborhood for making provisioning decision. An example for such queries can be *find the nodes where cpu usage is below* 10% *and memory usage is below* 50%. The response time of such queries was compared for centralized, proposed and DHT based approach. In addition, resource publish or mapping time was observed for those P2P approaches as well.

3.4.2 Experimental Result Analysis

The experiment first focused on the comparison among the response time of the queries invoked by the resource discovery schemes against total number of virtual machines in the datacenter. Then response time was observed against the epoch time in the case of a fixed number of virtual machines. After that, resource mapping time was observed for the decentralized resource discovery schemes followed by the data distribution among the nodes. Time is considered as discrete steps in this simulation context.

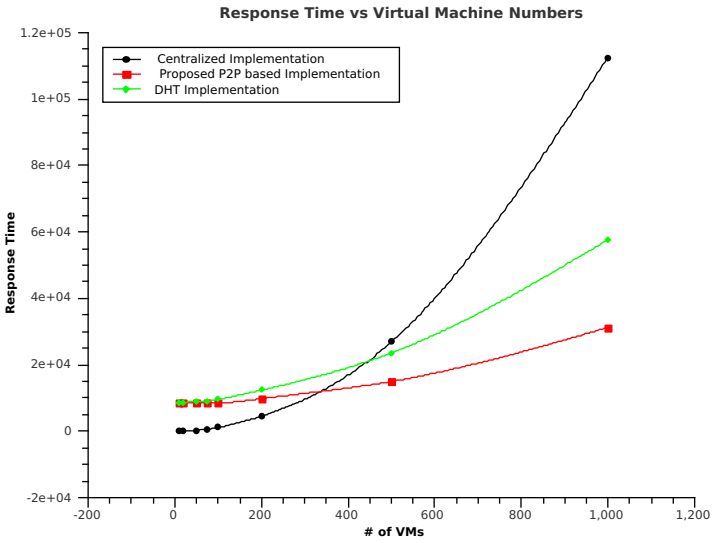

Figure 3.2: Response Time vs. Number of VMs

Response Time vs. Number of VMs: Figure 3.2 demonstrates the comparison of query response time among the proposed scheme, the centralized scheme and the DHT based scheme. From the figure, both of the decentralized resource discovery schemes show almost a linear growth in response time of the queries with respect to the increase of VM size. On the other hand, The centralized scheme showed an exponential growth in respect of VM size which is much higher than both of the decentralized schemes. As the number of VMs increased, the lone resource arbiter simply could not cater to all provisioning request at the same time. However in the proposed decentralized scheme, such bottleneck situation never occurred. This is why the proposed system generates 44.24% less response time on an average. However, the proposed method also generates 45.81% less response time as well in comparison with proposed DHT based implementation in the case of maximum number of VMs. The justification of this improvement from proposed implementation is: as DHT implementation requires determining the address around the overlay of each resource information via hierarchical region tree organization and hashing, a time penalty is

invoked because of computational processing invoked by centralized nature of the overlay for each information addressing.

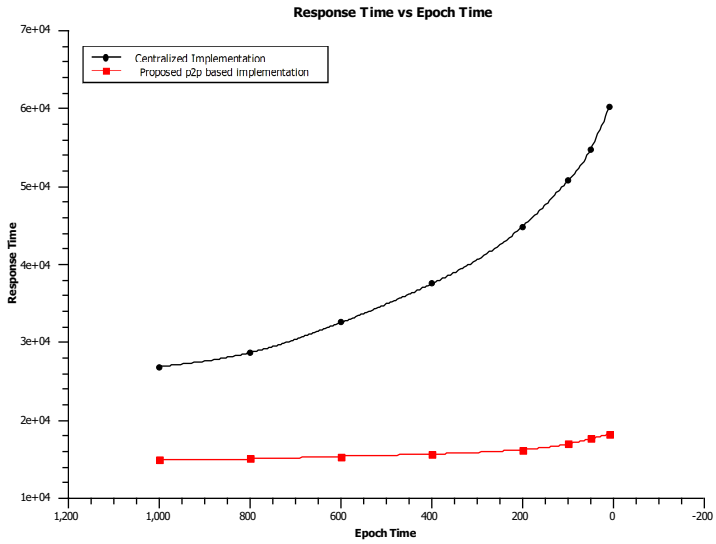

Figure 3.3: Response Time vs. Epoch for a Fixed Number of VMs

Response Time vs. Epoch: Figure 3.3 refers to the outcome of the experiment where resource discovery response time of both the centralized and the proposed scheme were compared in the context of epoch time. Epoch time refers to the time interval between two subsequent updates of provisioning information of the same physical machine. With the increase of epoch time frequency, the resource discovery scheme has to cope with more numbers of queries in a fixed amount of time. As the figure displays, the centralized scheme reveals exponential growth in query response time while the proposed scheme shows a linear growth in that occasion. Thus, it is obvious that response time in centralized discovery scheme is much higher than that of the proposed scheme with the decrease of epoch time window. The more frequent provisioning information were updated/queried, the more centralized scheme suffered bottleneck situation in serving all the requests because, the single resource arbiter had to deal with all the increased amount of discovery requests but its processing

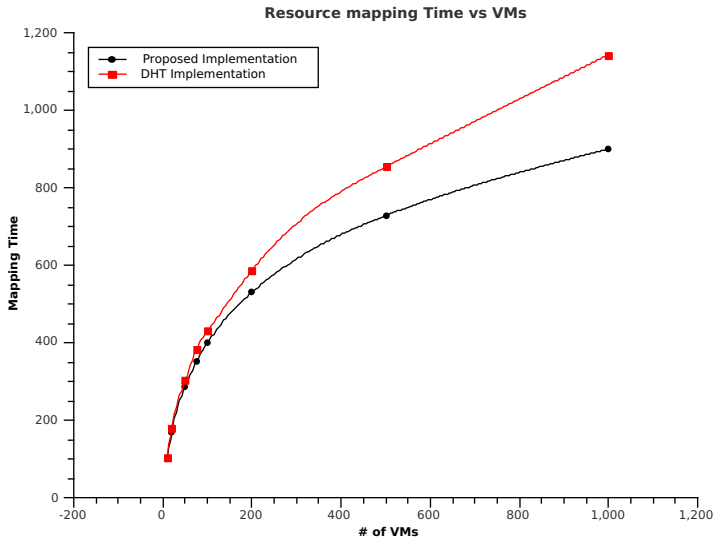

Figure 3.4: Resource Mapping Delay Time vs. VM

power and I/O bandwidth were fixed. The proposed scheme did not suffer from such lagging situation because, the increased amount of resources were distributed all over the attribute hub space in the overlay network and hence, none of them had to face the bottlenecked situation.

Resource Mapping Delay Time vs. VM: Figure 3.4 shows the measurement of average resource mapping delay for each provisioning data with respect to increase in the virtual machine size in the datacenter. The results shows that at higher number of virtual machines, the resource mapping or publishing task experienced increased delay logarithmically, proving the scalability of the proposed scheme. This happens due to the fact that the mapping tasks had to await longer period of time before getting matched against an update query. However, mapping delay is slightly higher, about 14.75% for the DHT based implementation than the proposed implementation due to time penalty obtained from hashing computation of data items to the node location mapping.

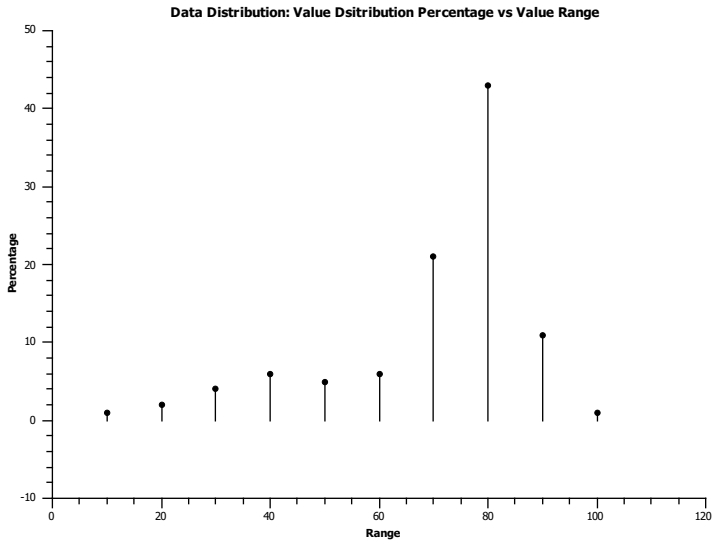

Figure 3.5: Distribution Percentage of CPU Attribute vs. Value Range

Distribution Percentage of CPU Attribute vs. Value Range: Figure 3.5 shows the provisioning resource distribution over the physical machines in the datacenter. In the simulated datacenter environment, most of the physical machines CPU usage were kept around 80% and thus most of the provisioning resource contained CPU usage data attribute between the range of 0.7 and 0.9. Consequently, the physical machines which were assigned to the range of $[0.7, 0.9]$ in CPU usage attribute hub, held most of the resources or pointers to those resources. On the other hand, DHT based implementation ensured uniform distribution of resources around the overlay network because hashing ensures that resources will be distributed uniformly and randomly over the nodes.

From the above experiments, the proposed resource discovery scheme shows significantly less query response time. It also reveals that, with the increase of virtual machines in the datacenter, the resource publishing time does not rise significantly. Although large number of resources are queried successfully by the proposed scheme, the resource distribution of the scheme is slightly uneven. Thus, it can be claimed that the proposed resource discovery scheme offers both decentralized and scalable

approach for provisioning in cloud with paying the penalty of nonuniform data distribution across the nodes.

3.5 Summary

Resource discovery is fundamental to the operation of datacenter because provisioning in cloud relies upon the information obtained from this scheme. Traditional resource discovery approaches for provisioning in cloud are centralized in nature. Those approaches are vulnerable from the aspect of scalability and single point failure issues. DHT based P2P approaches have also been addressed but those do not inherently support multi dimension range search needed for resource discovery queries. To tackle this problem, a decentralized resource discovery scheme has been proposed which is based on structured P2P network. The discovery scheme does not depend upon any global or hierarchal resource arbiter but allows publishing and querying provisioning data stored in a decentralized manner inside all the physical machines in the datacenter. The discovery scheme addresses P2P approach for routing data tuples and publishing them via storing those inside nodes, searching appropriate provisioning data stored in the nodes from the datacenter and fail-safe policies in case of physical machine failure. From the simulation, the proposed system has shown lower response time in resource query and mapping causing slightly nonuniform resource distribution. However, effectiveness of a resource discovery scheme can only be understood when it is utilized in resource provisioning. In the next chapter, focus is put on decentralized resource provisioning on the basis of resource discovery method proposed in this chapter.

Chapter 4

A Peer to Peer Resource Provisioning Scheme for Cloud Computing Environment Using Multi Attribute Utility Theory

4.1 Introduction

From technical point of view, virtualization holds the key to deploying dynamic number of applications in cloud computing environment. In datacenters, physical machines (PM) host virtual machines (VMs) depending on its capacity. VMs are configured with proper application environments in order to host the applications. VMs are also loosely coupled with PMs as those can be reconfigured or migrated from one PM to another. Configuration and utilization of these computational resources in the datacenter with dynamic adjustment is called resource provisioning [37, 26]. It also scales applications horizontally and vertically according to their resource needs and consumer provided constraints. This process is critical to the effectiveness of cloud computing because it manages all deployed applications intelligently to keep consumers satisfied through fulfilling their SLO.

Traditionally, a resource provisioning scheme consists of two phases [49], (i) initial static provisioning involving deploying applications in VMs as well as mapping VMs to PMs and (ii) runtime provisioning including reconfiguration and migration of VMs across the datacenters in time variant workload situations. Phase (i) is performed at initial startup of datacenter as well as maintenance purpose whereas, phase (ii) is performed continuously in the runtime. In phase (i), the issue of mapping between static number of VMs and PMs can be resolved with combinatorial optimization techniques such as Vector Bin Packing [50, 51]. On the other hand phase (ii) is

the most critical phase of resource provisioning because, datacenter linger on large portion of its operation span here. As research has been done extensively regarding the phase (i) [49], this chapter solely focuses on phase (ii).

Most of the provisioning schemes found in the literature feature statistical analysis [52], utility computing [53], reinforcement learning [54] and combinatorial optimization techniques [50]. Although those techniques work well in fulfilling SLO, with the increase of datacenter workload, centralized architecture becomes impractical and inefficient [55] due to the scalability issues. Moreover computing a global or sub-optimal configuration on the runtime of a large datacenter poses a serious impediment to achieve SLO. Thus scalability issues and single point failure threats stand in the way of effective provisioning.

This chapter presents a decentralized resource provisioning scheme based on multi attribute range query resource discovery method which utilizes structured P2P [56, 57] configuration of PMs in the datacenter. In P2P architecture, there is no dedicated client or server nodes in the network resulting in making the provisioning scheme inherently decentralized and scalable. For the sake of avoiding network and computational bottlenecks, the system considers local awareness from the neighborhood peers instead of maintaining global awareness. The system will obtain provisioning information neighborhood peers via multi attribute range query resource discovery scheme proposed in the previous chapter. In addition, each node will make its own provisioning decision regarding VM allocation and migration. Such decision is affected by numerous criteria such as computational resource availability, network bandwidth, migration cost and SLO constraints. This is why Multi Attribute Utility Theory (MAUT) [58, 7] methods are used to produce the best decision in such trade-off situations. Thus decentralized organization of nodes with each node doing its own job ensures that the system is scalable and resilient to single point failure.

For the simulation, the proposed provisioning scheme using the proposed resource discovery scheme and underlying datacenter environment has been developed in C# programming language. In addition, a centralized provisioning scheme has also been implemented which uses a single decision maker and computes provisioning decision based on first fit and first fit decreasing bin packing schemes [59] referred as FF and FFD in this chapter. Those were then tested to observe their SLO violation, migration and CPU utilization for a large number of VMs. The outcome concluded with the proposed provisioning scheme generating a significantly less number of SLO violations and migrations which is about 60.27% and 83.58% respectively paying the penalty of slightly lower CPU utilization. Such outcome proves the proposed system provisions resources better than centralized architecture.

4.2 Related Work

In the literature several resource provisioning schemes have been proposed. Most of those approaches perform centralized provisioning and scaling system through a single decision maker. In this section, first centralized and then decentralized schemes will be discussed.

In [53], an autonomous computing agent for datacenter is proposed. It transforms physical machines to a set of rational agents to manage their own behavior without human intervention. Although the system behaves adaptively without any explicitly defined decision model, it lacks scalability due to the dependency upon a single Global Resource Arbiter (GRA).

Reinforcement learning (RL) based provisioning scheme has been proposed in [54]. Instead of having any predefined model, it uses decompositional RL method to allocate resource dynamically. However, centralized architecture of this framework

and utility calculation of all the nodes in the datacenter make the provisioning scheme unable to scale with number of VMs increasing and avoid single point failure threat.

Zhang et al. proposed a resource management policy to perform load balancing based on VM performance and resource demand forecasts [52]. By utilizing statistical analysis, the system manages to decrease resource wastage in both peak and off peak hour by a significant margin. However, the system is prone to miscalculate the forecast and suffer from single point failure due to dependency on a GRA.

A decentralized decision making based resource provisioning scheme is proposed by Chieu et al. aiming to eliminate the threat of centralized provisioning schemes [49]. A distributed lightweight resource management system called Distributed Capacity Agent Manager (DCAM) is used in this framework. However, this decision making framework uses a central database for storing all PM and VM information that makes the system inherently centralized and thus difficult to scale.

Onat Yazir et al. proposed a decentralized resource provisioning scheme using PROMETHEE II, ELECTRE III and PAMSSEM II outranking methods [60]. This framework considers each PM as an independent decision making module. Provisioning information of other PMs are supplied from a global information source. Scalability is in question as that source proves to be yet an another centralized implementation.

In [39], a distributed hash table based cloud framework named cloud-peer is proposed. It deals with how real life P2P architectures can handle cloud monitoring, routing architecture, service discovery, load balancing and message passing schemes. However, the work focused little on VM allocation and migration policies in a decentralized manner.

In addition, various optimization techniques have been implemented to minimize server sprawling and SLO violation. Constraint Satisfaction Problem and NP Hard

optimization such as bin packing and integer programming are utilized to host a fixed number of VMs into minimum number of PMs so that it issues less migration [50, 61, 51]. Those are similar to the previously discussed systems because of the presence of a Global Resource Arbiter. Although being effective in small to medium sized datacenter, performance of these schemes in extremely large datacenters is still a major concern.

In principle, these systems recalculate the whole datacenter configurations periodically through centralized process. Consequently suffering from system lag and outdated outputs are imminent regardless of the goodness of the optimization techniques. Moreover, scaling and resilience to single point failure threat for large datacenters is also questionable.

4.3 Overview of the Proposed Resource Provisioning

In order to cope with a large number of VMs deployed in the datacenter, scalability issue and single point failure threat must be addressed. Hence the proposed system features no Global Resource Arbiter(GRA). Each node will make the decision regarding its own provisioning such as application profiling, resource utilization, allocation and migration. Moreover each node needs to know about similar provisioning information of other nodes. As a result, all the nodes need a source for obtaining such knowledge. However if the system has a global information provider which will maintain global awareness in the datacenter, the system will be exposed to single point failure and higher data retrieval latency. To tackle such threats, nodes in the proposed system are organized based on structured P2P architecture proposed in the previous chapter. Such architecture is used in order to form a large datacenter setup

where centralized configuration is undesired for its inability to tackle scalability and single point fault issues.

Each node will pull out information from the neighborhood but not necessarily from all the nodes in the datacenter. As maintaining global awareness in the datacenter is nearly infeasible due to huge computational overheads, the proposed provisioning scheme uses local awareness. Each node will pull information from its neighbor with a constant node distance with the value of $\log_d N$ where N is the total number of nodes and d is the average degree of a node [62]. This overlay structure and interconnectivity of the nodes are formed according to the multi attribute range query policy described in the previous chapter. Upon this structure, each node can pull out important provisioning information from other nodes using multi dimensional range queries facilitated by aforementioned proposed resource discovery method in the previous chapter. Finally, the system proposes a policy that uses Multi Attribute Utility Theory (MAUT) for migration of VMs that minimizes the undesired re-migration of VMs as well as provides opportunity of tweaking and tuning migration criteria.

4.4 Proposed System Architecture for Decentralized Resource Provisioning

The system architecture of the proposed resource provisioning is composed of three major modules. These are Application Agent (AA), Node Agent (NA) and a Job Pool (JP). AA is an entity that is tightly coupled with each application that is submitted to the datacenter. As the resource requirements in real time application is always subject to change, responsibility of an AA is to demand latest computational resource from its host. In the proposed system, each application is considered to be deployed in a VM and no more than one VM will host the same application. Therefore, VMs are

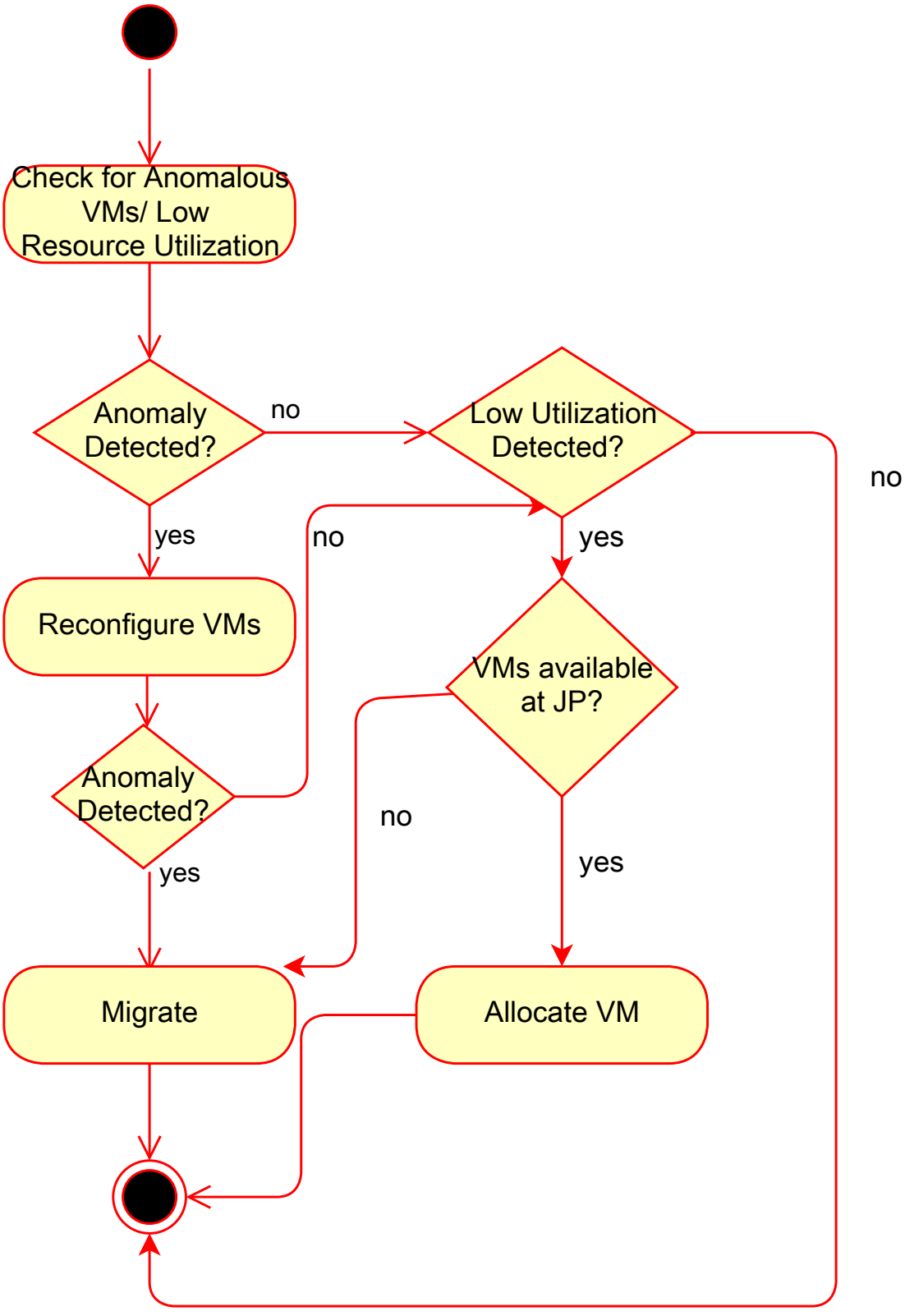

Figure 4.1: NA Activity Flow in Each Loop

considered as application or task unit and assigned to PMs which have the ample resource to host and run those.

Every PM in the datacenter hosts exactly one NA. Each NA monitors the resource usage of the VMs hosted by the same PM. It also performs allocation of VMs which has just reached the datacenter. When the corresponding PM is incapable of hosting VMs, NA reconfigure or migrate those VMs.

The system also maintains a pool named Job Pool (JP) where unallocated VMs are stored. When an application is submitted to the datacenter for execution, it is mapped to a VM and stored in this pool. During the whole life cycle of a VM, it maintains certain status for its corresponding phase. Inside the JP, status of each VM is *unassigned*. NAs will look for unassigned VMs here and allocate it to a suitable PM.

Once a VM is assigned to a PM, the status will be changed to *assigned*. When the PM begins the execution of the VM, the status becomes *allocated*. During the migration process, its status is called *migrating*. Finally, once the VM finishes its defined task, its status becomes *terminated*.

Apart from executing VMs, each NA also performs several actions in order to maintain the resource utilization of its host to a desired level which is demonstrated in Figure 4.1. These following tasks are executed sequentially in a loop. Each NA maintains its utilization index which is bounded by an upper and a lower value. Utilization index value denotes how much system resource is being used by the host. In each loop, NA checks whether its utilization value is below the lower bound threshold. If so, it will look for an unassigned VM in the JP and if it finds out a VM which can be accommodated by the host PM, NA will assign that VM to itself. Otherwise, it will look for a suitable PM in the neighborhood which can host it.

Moreover, NA will continuously monitor the performance of the hosted VMs. If NA detects one or several VMs behaving anomalously (this scenario can be characterized as resource usage beyond the upper bound of utilization index), it will first reconfigure these problematic VMs. Reconfiguration simply means allocating more computational resources or withdrawing some. However, if the reconfiguration does not work or seems to be impossible due to system resource constraints, NA will stop the VM and invoke a migration. This is identical to the scenario of looking for a suitable PM of an unassigned VM for allocation. In both cases, NA will issue an inquiry message to the neighborhood for a PM suitable enough to host that problematic VM. In an structured P2P network, it might be noticed that a certain region is more overloaded than the rest of the network. Any node in that region might not find a suitable option for migration inside that neighborhood. However, such scenario is highly unlikely as P2P network is dynamic and its overlay network configuration is subject to change periodically.

Upon acquiring information from peers, the NA will compute the best suitable PM for this particular VM using MAUT methods. The computation can be based on various criteria. However the most important criteria is the availability of the resources demanded by the VM. Other criteria will be discussed later. If such a PM is found, NA sends a resource lock request to the target node before delegating the task.

The NA which monitors the receiving PM, will check for the resource whether it is still available when it is accepting the lock request. If resource availability is found, the lock request will be accepted. In addition, the receiving NA will maintain a timeout value until which it maintains the lock. If timeout expires, it will cancel the lock and release the resources. NA also keeps a timeout value until which, it will await the reply from neighborhood. If no such reply is received within that time

window, the VM status will be changed to unassigned and sent back to the pool for processing later.

4.5 Provisioning Decision Making Model

The overall process regarding allocation of an unassigned VM or migrating a problematic VM to other suitable node can be simplified into two steps. First it needs to determine which VM is the root of anomalous behavior and the second one is to decide in which PM the VM will be migrated.

The computation regarding first step is not simple because of several critical issues. These are maximization of resource utilization and minimization of migration cost, SLO violation and further probability to migrate the VM. The most important criteria of choosing a suitable PM are mentioned here. The (+) emblem denotes that the criterion impacts the decision positively via maximization. For example the PM that has higher available computation resource is a better choice. (-) emblem signifies exactly the opposite. These criteria are:

- **CPU** (+): Availability of CPU resource of that PM
- **Memory** (+): Availability of primary memory of that PM
- **I/O bandwidth** (+): Availability of read/write bandwidth of that PM
- **Peer Distance** (-): Node distance between the current PM and potential suitable PM

The issues regarding the second step are finding the nodes having available resources, maximizing the resource utilization and minimizing the possibility of re-migration. For choosing an anomalous VM, critical factors are:

- **Migration Cost** (-): Cost of migrating a VM to other PM. This cost depends on type of migration such as live or offline, distance, network bandwidth, application type etc.

- **Priority (+)**: The priority of the application executing inside the VM. The priority depends on the type of the application. For real time application, the priority is higher than the batch processing applications.

- **SLO Violation Penalty** (-): Application performance will be degraded during the migration because of context switching followed by SLO violation. CSPs might have to pay monetary penalty for such violations.

- **Cumulative Migration Factor** (-): One single VM cannot be allowed to cause consequent migration requests. The more a VM invokes migration, the less it will be given preference for future migrations.

As the criticality of these criteria depends upon the nature of application, workload and datacenter configuration, the problem can be solved with Multi Attribute Utility Theory (MAUT) Methods. According to MAUT methods, there are several alternatives and criteria such as neighborhood PMs that can host migration-worthy VMs and VMs that are causing undesired behaviors in this proposed system scenario. MAUT methods will be fed the alternatives and criteria with weights. These weights will signify how important the contribution of a single criteria in choosing the alternative. In this proposed system three MAUT methods are used. Those are two Simple Multi Attribute Ranking Techniques (SMART) and Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS). According to MAUT methods, there are m criteria with weights and n alternatives. a_{ij} denotes the score of j^{th} alternative on i^{th} criteria. Among these MAUT methods, SMART is the simplest.

The ranking value x_j of j^{th} alternative is obtained simply as the weighted mean of the score associated with it.

$$x_j = \frac{\sum_{i=1}^{m} w_i a_{ij}}{\sum_{i=1}^{m} w_i}; j = 1, ..., n \tag{4.1}$$

$$x_j = \prod_{i=1}^{m} a_{ij}^{\frac{w_i}{w}}; j = 1, ..., n \tag{4.2}$$

The method SMART Arithmetic is based on equation 4.1 and SMART Geometric is based on equation 4.2. TOPSIS is the third MAUT method used in this proposed system which focuses on the best decision closest to the ideal solution and furthest from the negative ideal solution [63].

In principle, the proposed system is based on structured multi attribute range query P2P network with making decision locally via multi dimensional range query resource discovery method. This policy ensures scalability no matter how many VMs are deployed. P2P architecture ensures no single point failure threat is imminent because there is no Resource Arbiter that can shut down the whole datacenter if it fails. Flexibility is also achieved as VM allocation and migration decisions can be taken with arbitrary number of reconfigurable criteria fed into MAUT methods.

4.6 Simulation Setup and Experimental Result Analysis

In this section, the focus will be put on how the proposed resource provisioning scheme performs in comparison with the centralized scheme. First, the simulation environment of the datacenter will be highlighted. Then simulation goal will be presented followed by the discussion on obtained results.

4.6.1 Simulation Setup

The simulation platform was built using C# programming language. It focused on emulating a datacenter including numerous PMs forming a P2P structure. The overlay network was built according to the simulation setup described in the previous chapter. As each PM contained a fixed amount of CPU, Memory and IO bandwidth resources required for executing VMs, it can host a limited number of VMs without violating SLO. VMs were also characterized by their change of resource demand in CPU, memory and IO bandwidth with respect to time. These changes in the resource demands were generated following Gaussian, Poisson, uniform and burst statistical distributions [45]. As the production level datacenters deploy about 1 million physical machines in the datacenter, in this simulated environment, the VM counts are kept around 1×10^5 to emulate the scenario of a large datacenter[46, 47, 48]. However, for the procurement of the result, around 3×10^4 VMs are considered sufficient to understand the characteristics of the plotted graphs as well as the performance comparisons among the schemes under observation.

When the simulation started, VMs demanded computational resources according to predefined resource demand based on statistical distributions. The passage of time in the datacenter was simulated as steps. Each step corresponded to a unit of time. Thus the flow of simulation was executed step by step where in each step, PMs executed the VMs according to their resource demand and then performed reconfiguration and migration processes if necessary. VMs also demanded resources in stepwise manner. This step based simulation facilitated measuring the state of every PMs precisely the same point in each simulation run. If an asynchronous parallel simulation run were used, datacenter configuration with the same input would have culminated in different outcomes.

Regarding the migration of VMs to suitable physical machines which have the enough computational resource to host it, all the criteria were fed into three MAUT methods specified in the previous section. However, for choosing an anomalous VM, two out of four identified criteria were fed into the methods. These criteria were Priority and Cumulative Migration factor. All the criteria were assigned equal weights. In addition, the situation when no suitable PM was found for a VM that needed to be migrated, simulation environment sent the VM to JP and reallocate it to a new PM from JP.

In order to compare the proposed system with the centralize scheme, simulation of centralized scheme has to be defined. Hence, datacenter incorporated a GRA. When migration was needed, each PM requested the RA for the migration. RA used two schemes named First Fit (FF) and First Fit Decreasing (FFD) in order to migrate VMs. FF scheme denotes migrating the VM into the very first available PM which can host that VM while FFD scheme means migrating the VM into the very first PM sorted in descending order on the basis of criteria. It is noteworthy that allocating a static number of VMs in as less PMs as possible is a Vector Bin Packing problem. This problem is a NP-Hard problem and applying greedy as well as approximation scheme FF and FFD ensures $(11/9)OP + 1$ solution where OP denotes the optimal solution [64]. Hence, the outputs of FF and FFD schemes can be considered as the output of any VM migration algorithm based on centralized provisioning acquiring global information.

The simulation targeted the number of total SLO violations, migrations and server sprawling in the datacenter. In the simulation, when a VM did not manage to get desired computational resource in each step of application lifetime, it was assumed that a SLO violation had occurred. In addition, let v as number of virtual machines were allocated to p number of physical machines. However with the passage of time,

for avoiding SLO violations those v number of VMs used some arbitrary additional number of PMs besides p number of PMs. This scenario is called server sprawling. Higher number of server sprawling denotes relatively lower CPU utilization.

4.6.2 Experimental Result Analysis

There were three types of comparisons done in this experiment. In the first and second case, SLO violation and migration count were compared for both the centralized with FF and FFD methods as well as proposed system with SMART Arithmetic, SMART Geometric and TOPSIS method. In the third test, server sprawling was focused for both centralized and proposed systems similarly to the first and second test.

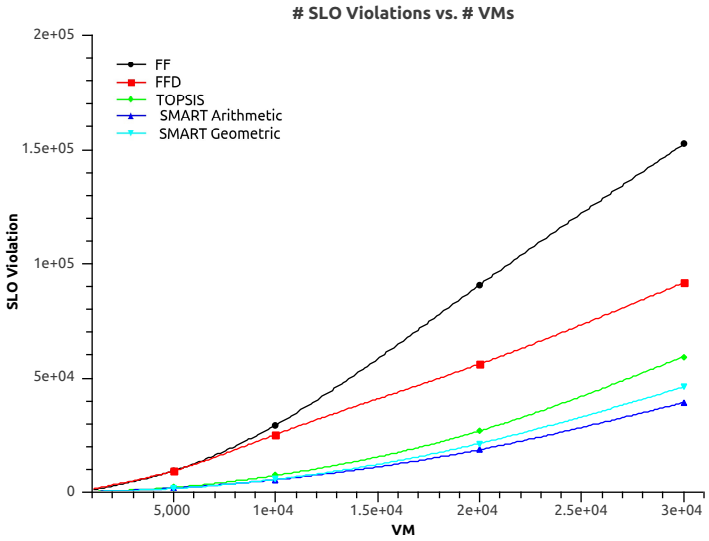

Figure 4.2: SLO Violation vs. VM

SLO Violation vs. VM: The first experimental result shown in Figure 4.2 demonstrates how the proposed scheme performs to tackle SLO violation in comparison with centralized schemes. Both FF and FFD schemes generate a higher number of SLO violations compared with SMART Arithmetic, SMART Geometric and TOPSIS. As

number of VMs increased, the lone RA simply could not cater to all provisioning request. This is why the proposed system generates 60.27% less number of SLO violations on average. Between two centralized schemes FFD shows better result than that of FF as, with the increase of VMs, SLO violation count increases linearly and exponentially for FFD and FF respectively. However, among the decentralized schemes, SMART Arithmetic performs the best while SMART Geometric and TOPSIS show similar outputs with TOPSIS falling behind for very high number of VMs.

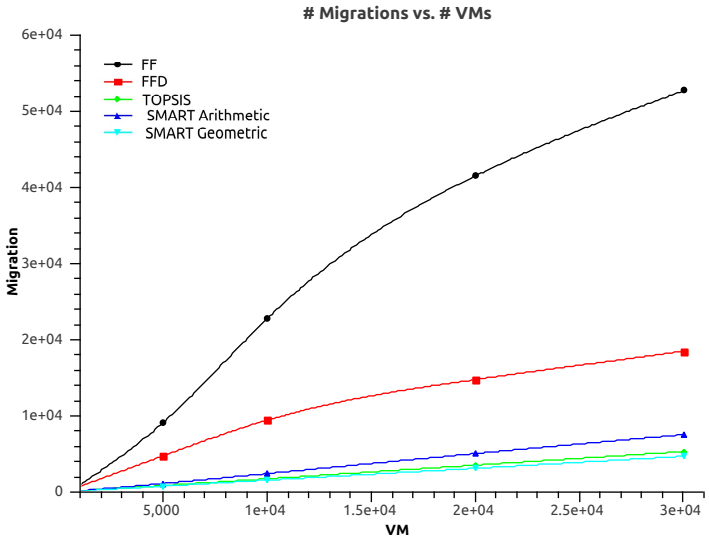

Figure 4.3: Migration vs. VM

Migration vs. VM Similar behavior has been obtained from the second experiment where number of migrations is observed. The proposed decentralized schemes generate significantly less number of migration request than the centralized ones. Overall, the proposed system generates 83.58% less migrations on average as displayed in Figure 4.3.. Compared to the first experiment, centralized schemes perform worse. The hindrance created by the single GRA is mainly responsible for this. Moreover, the proposed system migrates VM in such manner so that the probability of re-migration of already migrated VMs is kept low. FF performs worse than FFD be-

cause it allocates VMs in the first available PM increasing the chance of re-migration. SMART Arithmetic, SMART Geometric and TOPSIS perform almost similar while SMART Arithmetic and SMART Geometric present the most and least migration affinity respectively.

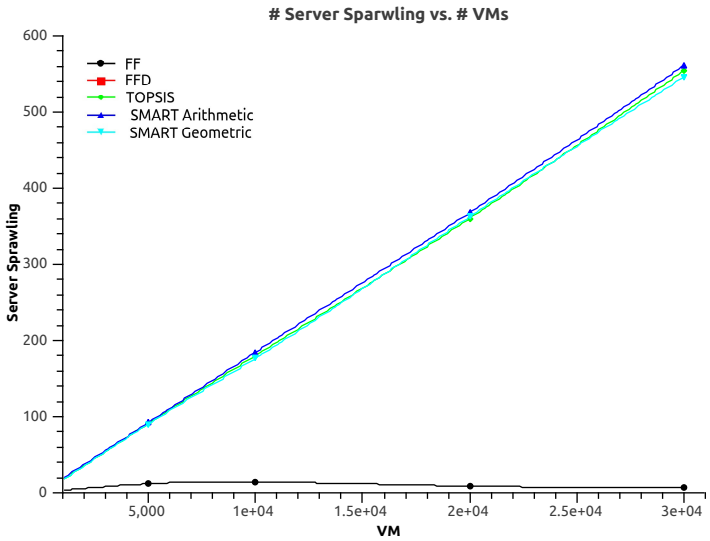

Figure 4.4: Sprawling vs. VM

Sprawling vs. VM The third experiment, as shown in Figure 4.4 demonstrates server sprawling is literally zero in FF and FFD scheme while two SMART and TOPSIS schemes show high rate of server sprawling. This happens in decentralized schemes when no suitable PMs is found in the neighborhood region in P2P network. In that scenario, a new PM was used to host that VM. Hence server sprawling occurs in the proposed scheme.

From the three experiments, the proposed system violated SLO linearly with respect to increase in total number of VMs. That proves scalability of the system. Besides, migrations invoked by the proposed system are far less than centralized schemes which have GRA vulnerable to single point dependency. MAUT methods used in provisioning decision allows VM allocation and migration in flexible manner.

In order to achieve this, the scheme has to sacrifice a margin of utilization highlighted by high rate of server sprawling.

In principle, the experiment highlights that proposed scheme issues significantly less number of SLO violations and migrations considering the penalty of using slightly more resources.

4.7 Summary

Resource provisioning refers to the management of computational resources such as VM allocation, task allocation, migration and so forth. Business model operated by cloud service providers depends heavily on the effectiveness of this provisioning as provisioning directly puts impacts upon meeting the SLO of consumers. Traditional provisioning schemes used in cloud environments are centralized in nature. These schemes lead to scalability and single point failure issues. To tackle this problem, a decentralized resource provisioning scheme has been proposed which is based on structured multi attribute range query P2P network. Provisioning information from peer nodes are achieved via proposed resource discovery method presented in the previous chapter which supports multi dimensional range queries. The provisioning scheme does not depend upon any global provisioning decision maker but delegates each node its own provisioning responsibility. It also uses MAUT methods for allocating VMs into suitable PMs and VM migrations. From the simulation, the proposed system has shown less number of SLO violations and migrations causing slightly lower resource utilization. In the next chapter, retrospect on the proposed resource provisioning scheme based on decentralized P2P resource discovery method will be discussed along with future direction on how the current work can be enriched.

Chapter 5

Discussion and Conclusion

5.1 Introduction

Developing provisioning techniques that integrate application services in a scalable and fail-safe fashion is critical to the success of Cloud computing platforms. Architecting provisioning techniques based on peer-to-peer network models is significant; since peer-to-peer networks are highly scalable, they can gracefully adapt to the dynamic system expansion (join) or contraction (leave, failure), and are not susceptible to a single point of failure. In this thesis, a structured P2P based decentralized resource provisioning with underlying resource discovery scheme is presented. In this chapter, first discussion regarding the proposed resource discovery and provisioning model will be presented followed by the future direction and room for improvements of this research.

5.2 A Peer to Peer Resource Discovery Scheme in Cloud Using Multi Attribute Range Query

Resource discovery is a key process for provisioning in Cloud as fundamental provisioning information such as the status of the nodes, resource usage, virtual machine allocation, deployed applications and SLO updates. Provisioning jobs such as virtual machine allocation and migration require knowledge about neighbor nodes are procured through this discovery method. At present, major cloud stacks perform this process in centralized fashion which leads to suffering from scalability issues.

To tackle scalability and single point vulnerabilities, this thesis proposes decentralized P2P based discovery scheme . It proposes how inherent inability of answering multi dimensional queries in structured DHT based P2P based network is addressed by proposing an attribute hub based network overlay with data indexing, routing and storing inside nodes. Resource publishing and querying along with node join and departure fail safe policies are also presented here.

In the simulation, the proposed resource discovery scheme is put to comparison with centralized and DHT based approach of resource discovery. From the obtained result, it is observed that the proposed system has shown significantly lower response time in resource query response time which is about 44.24% on average and 45.81% in the case of maixmum number of VMs respectively in comparison with centralized and DHT based implementation. However, the proposed method invokes uneven data distribution which leads to the fact that the datacenter configuration needs more secondary memory to cope with uneven data distribution around the overlay network. However, quicker response time guarantees ensuring SLA as promised by the cloud service provider which is much more cost effective in comparison with added secondary memory cost [65, 66, 67].

5.3 A Peer to Peer Resource Provisioning Scheme for Cloud Computing Environment Using Multi Attribute Utility Theory

The management and decision making regarding datacenter operation such as VM and task allocation, VM migration, physical node and application service management is referred to as resource provisioning. From its definition, understandably, effectiveness

of cloud based services with huge user base depends on how underlying provisioning can handle large workload having direct impact on SLO. Current market leaders in IaaS providers such as OpenStack, Eucalyptus, Amazon EC2 features master-slave architecture oriented resource discovery and provisioning [5, 2, 3]. Thus SLO is always in threat of violation caused by the potential of sudden service outage due to scalability issues.

To address this problem, a decentralized resource provisioning scheme has been proposed which is based on structured multi attribute range query P2P overlay network. Provisioning information from peer nodes are achieved via proposed resource discovery method which supports multi dimensional range queries. The provisioning scheme does not depend upon any global provisioning decision maker but delegates each node its own provisioning responsibility. It also uses MAUT methods for allocating VMs into suitable PMs and VM migrations.

In the simulated environment, the proposed decentralized provisioning scheme has been compared to global resource arbiter based centralized provisioning approach. Procured result demonstrates that the proposed system has shown less number of SLO violation and migration causing slightly lower resource utilization which is about 60.27% and 83.58%. Although slightly higher rate of server sprawling has also been noticed, the provisioning model succeeds in terms of meeting the SLO demand and saving the CSP from paying monetary penalties [65].

5.4 Future Direction

Addressing the uneven data distribution in the proposed discovery method and high rate of server sprawling in the provisioning model are two major concerns that can pave the way for future research. In order to improve the correlation of simulation and

real life scenario, network latency, peer co-ordination delay, query patter and routing delay from a production level environment could be observed and characterized into precise statistical workload model. Moreover, only limited numbers of criteria are considered in the simulation however, resource utilization, service level agreement, migration penalty, network bandwidth issues are critical enough to be included in each and every decisions at provisioning schemes.

In addition, MapReduce framework can be used for efficient parallel workflow management of resource identifying queries. The broader vision of this research endeavor is to develop a real life open source cloud component, for example, Openstack Cloud where every entity will be a peer and completely free of centralized provisioning schemes. Other multidimensional data indexing and routing techniques that can achieve close to logarithmic bounds on messages and routing state, balance query (discovery, load-balancing, coordination) and processing load, preserve data locality and minimize the metadata should also be explored.

Another important algorithmic and programming challenge in building robust P2P cloud services is to guarantee consistent routing, look-up and information consistency under concurrent leave, failure, and join operations by application services. To address those issues, robust fault-tolerance strategies should also be based on distributed replication of attribute/query subspaces to achieve a high level of robustness and performance guarantees. Moreover, decentralized P2P based storage management and replication framework can ensure further reliability and fault tolerant capability of the Cloud.

Bibliography

[1] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, *et al.*, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4–1, 2009.

[2] "Eucalyptus systems." http://www.eucalyptus.com/. last accessed on April 20, 2014.

[3] "Open source software for building private and public clouds." https://www.openstack.org/. last accessed on April 20, 2014.

[4] "Apache cloudstack." http://cloudstack.apache.org/. last accessed on April 20, 2014.

[5] "Amazon cloudwatch service." `http://aws.amazon.com/cloudwatch/`. last accessed on April 20, 2014.

[6] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: supporting scalable multi-attribute range queries," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 353–366, 2004.

[7] J. S. Dyer, "Mautmultiattribute utility theory," in *Multiple criteria decision analysis: state of the art surveys*, pp. 265–292, Springer, 2005.

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Tech. Rep. UCB/EECS*, vol. 28, 2009.

[9] A. Amies, H. Sluiman, Q. Tong, and G. Liu, *Developing and Hosting Applications on the Cloud*. IBM Press, 2012.

[10] M. Hamdaqa, T. Livogiannis, and L. Tahvildari, "A reference model for developing cloud applications," in *In Proceedings of the 1st International Conference on Cloud Computing and Services Science*, vol. 11, pp. 98–103, 2011.

[11] R. Harris, *Introduction to decision making*. http://www.virtualsalt.com/crebook5.htm, 2012.

[12] D. Baker, D. Bridges, R. Hunter, G. Johnson, J. Krupa, J. Murphy, and K. Sorenson, *Guidebook to decision-making methods*. Website http://www.dss.dpem.tuc.gr/pdf/Decision

[13] G. Nemhauser, A. Rinnooy Kan, and M. Todd, *Handbooks in operations research and management science, vol. 1: optimization*, vol. 8. North-Holland, 1989.

[14] R. Keeney and H. Raiffa, *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge University Press, 1993.

[15] B. Roy, "Classement et choix en présence de points de vue multiples (la méthode electre)," *RIRO*, vol. 2, no. 8, pp. 57–75, 1968.

[16] R. Schfer, "Rules for using multi-attribute utility theory for estimating a users interests, workshop on adaptivity and user modelling," in *University of Dortmund*, Cambridge University Press, 2001.

[17] F. Barron and B. Barrett, "The efficacy of smartersimple multi-attribute rating technique extended to ranking," *Acta Psychologica*, vol. 93, no. 1, pp. 23–36, 1996.

[18] J. Brans and P. Vincke, "Notea preference ranking organisation method (the promethee method for multiple criteria decision-making)," *Management science*, vol. 31, no. 6, pp. 647–656, 1985.

[19] S. Greco, *Multiple criteria decision analysis: state of the art surveys*, vol. 78. Springer, 2004.

[20] J. Brans, P. Vincke, and B. Mareschal, "How to select and how to rank projects: The promethee method," *European Journal of Operational Research*, vol. 24, no. 2, pp. 228–238, 1986.

[21] G.-H. Tzeng and J.-J. Huang, *Multiple attribute decision making: methods and applications*. CRC Press, 2011.

[22] F. Dabek, *A distributed hash table*. PhD thesis, Massachusetts Institute of Technology, 2005.

[23] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 654–663, ACM, 1997.

[24] "Amazon load balancer service." http://aws.amazon.com/elasticloadbalancing/. last accessed on April 20, 2014.

[25] "Windows azure platform." www.microsoft.com/azure/. last accessed on April 20, 2014.

[26] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in *Proc. of 10th IEEE/ACM International Conference on Grid Computing*, pp. 50–57, IEEE, 2009.

[27] R. Ranjan, A. Harwood, and R. Buyya, "Peer-to-peer-based resource discovery in global grids: a tutorial," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 2, pp. 6–33, 2008.

[28] X. Zhang, J. L. Freschl, and J. M. Schopf, "A performance study of monitoring and information services for distributed systems," in *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pp. 270–281, IEEE, 2003.

[29] S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk, *et al.*, "Cryptographic hash functions: A survey," *Centre for Computer Security Research, Department of Computer Science, University of Wollongong, Australia*, 1995.

[30] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in p2p systems," *Communications of the ACM*, vol. 46, no. 2, pp. 43–48, 2003.

[31] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 654–663, ACM, 1997.

[32] B. Preneel, "The state of cryptographic hash functions," in *Lectures on Data Security*, pp. 158–182, Springer, 1999.

[33] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim, *et al.*, "A survey and comparison of peer-to-peer overlay network schemes.," *IEEE Communications Surveys and Tutorials*, vol. 7, no. 1-4, pp. 72–93, 2005.

[34] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud computing: Principles and paradigms*, vol. 87. John Wiley & Sons, 2010.

[35] P. Ganesan, B. Yang, and H. Garcia-Molina, "One torus to rule them all: multi-dimensional queries in p2p systems," in *Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004*, pp. 19–24, ACM, 2004.

[36] H. Samet, *The design and analysis of spatial data structures*, vol. 85. Addison-Wesley Reading, MA, 1990.

[37] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 164–177, 2012.

[38] R. Rahman, A. Imran, A. U. Gias, and K. Sakib, "A peer to peer resource provisioning scheme for cloud computing environment using multi attribute utility theory," in *Innovative Computing Technology (INTECH), 2013 Third International Conference on*, pp. 132–137, IEEE, 2013.

[39] R. Ranjan and L. Zhao, "Peer-to-peer service provisioning in cloud computing environments," *The Journal of Supercomputing*, vol. 65, no. 1, pp. 154–184, 2013.

[40] J. Varia, "Cloud architecture- amazon web service," tech. rep., 2009.

[41] "Google app engine." http://code.google.com/appengine/. last accessed on April 20, 2014.

[42] "Gogrid cloud hosting (2010) (f5) load balancer. gogrid wiki." http://wiki.gogrid.com/wiki/index.php/(F5)-Load-Balancer. last accessed on April 20, 2014.

[43] "Amazon auto scaling service." http://aws.amazon.com/cloudwatch/. last accessed on April 20, 2014.

[44] J. Figueira, S. Greco, and M. Ehrgott, *Multiple criteria decision analysis: state of the art surveys*, vol. 78. Springer, 2005.

[45] A. Shingo, M. Murata, and H. Miyahara, "Analysis of network traffic and its application to design of high-speed routers," *IEICE Transactions on Information and Systems*, vol. 83, no. 5, pp. 988–995, 2000.

[46] "Report: Google uses about 900,000 servers." `http://www.datacenterknowledge.com/archives/2011/08/01/report-google-uses-about-900000-servers/`. last accessed on May 20, 2014.

[47] "Google throws open doors to its top-secret data center." `http://www.wired.com/2012/10/ff-inside-google-data-center/all/`. last accessed on May 20, 2014.

[48] "Microsoft now has one million servers less than google, but more than amazon, says ballmer." `http://www.extremetech.com/extreme/161772-microsoft-now-has-one-million-servers-less-than-google-but-more-than-a` last accessed on May 20, 2014.

[49] T. C. Chieu and H. Chan, "Dynamic resource allocation via distributed decisions in cloud environment," in *Proc. of 8th IEEE International Conference on e-Business Engineering*, pp. 125–130, IEEE, 2011.

[50] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *Proc. of 10th IEEE/IFIP Network Operations and Management Symposium*, pp. 373–381, IEEE, 2006.

[51] C. Papagianni, A. Leivadeas, S. Papavassiliou, V. Maglaris, A. Monje, *et al.*, "On the optimal allocation of virtual resources in cloud computing networks," 2012.

[52] Z. Zhang, H. Wang, L. Xiao, and L. Ruan, "A statistical based resource allocation scheme in cloud," in *Proc. of International Conference on Cloud and Service Computing*, pp. 266–273, IEEE, 2011.

[53] J. O. Kephart and W. E. Walsh, "An artificial intelligence perspective on autonomic computing policies," in *Proc. of 5th IEEE International Workshop on Policies for Distributed Systems and Networks*, pp. 3–12, IEEE, 2004.

[54] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in *Proc. of IEEE International Conference on Autonomic Computing*, pp. 65–73, IEEE, 2006.

[55] Y. O. Yazır, *Multiple Criteria Decision Analysis in Autonomous Computing: A Study on Independent and Coordinated Self-Management*. PhD thesis, University of Victoria, 2011.

[56] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Proc. of 1st International Conference on Peer-to-Peer Computing*, pp. 101–102, IEEE, 2001.

[57] M. Yang and Y. Yang, "An efficient hybrid peer-to-peer system for distributed data sharing," *IEEE Transactions on Computers*, vol. 59, no. 9, pp. 1158–1171, 2010.

[58] R. L. Keeney and H. Raiffa, *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge University Press, 1993.

[59] D. S. Johnson, *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[60] Y. O. Yazir, Y. Akbulut, R. Farahbod, A. Guitouni, S. W. Neville, S. Ganti, and Y. Coady, "Autonomous resource consolidation management in clouds using impromptu extensions," in *Proc. of 5th IEEE International Conference on Cloud Computing*, pp. 614–621, IEEE, 2012.

[61] H. Nguyen Van, F. Dang Tran, and J.-M. Menaud, "Autonomic virtual resource management for service hosting platforms," in *ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pp. 1–8, IEEE Computer Society, 2009.

[62] S. Tewari and L. Kleinrock, "Entropy and search distance in peer-to-peer networks," tech. rep., UCLA Computer Science Dept Technical Report UCLACSD-TR050049, 2005.

[63] T. Gwo-Hshiung, G. H. Tzeng, and J.-J. Huang, *Multiple attribute decision making: Methods and applications*. Chapman & Hall, 2011.

[64] M. Yue, "A simple proof of the inequality ffd (l) 11/9 opt (l) + 1, l for the ffd bin-packing algorithm," *Acta Mathematicae Applicatae Sinica*, vol. 7, no. 4, pp. 321–331, 1991.

[65] P. Patel, A. H. Ranabahu, and A. P. Sheth, "Service level agreement in cloud computing," 2009.

[66] S. A. Baset, "Cloud slas: present and future," *ACM SIGOPS Operating Systems Review*, vol. 46, no. 2, pp. 57–66, 2012.

[67] "Service level agreement." http://www.gogrid.com/legal/service-level-agreement-sla. last accessed on April 20, 2014.