A PROACTIVE PEER TO PEER RESOURCE PROVISIONING SCHEME IN CLOUD COMPUTING ENVIRONMENT

MD. RAYHANUR RAHMAN
BIT 0101

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

BACHELOR OF INFORMATION TECHNOLOGY
(SOFTWARE ENGINEERING)

Institute of Information Technology
University of Dhaka
DHAKA, BANGLADESH

A PROACTIVE PEER TO PEER RESOURCE PROVISIONING SCHEME IN CLOUD COMPUTING ENVIRONMENT

MD. RAYHANUR RAHMAN

Approved:

Signature *Date*

Supervisor: Dr. Kazi Muheymin-Us-Sakib

Committee Member: Dr. Kazi Muheymin-Us-Sakib

Committee Member: Dr. Md. Shariful Islam

Committee Member: Dr. Su Bong Kim

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

Abstract

In cloud computing scenario, resource wastage is undesired from financial and computational point of view. Besides under-utilization which is the key reason behind resource wastage, over-utilization is not expected either as it leads to slow response time of server applications. At present, conventional cloud infrastructures are inherently built to execute in centralized manners which poses serious single point failure threat. Thus there is always a possibility of slow response time due to mis-utilization of resources. Unavailability in cloud computing services is imminent due to dependency on a central server.

In this research effort, these critical issues are investigated thoroughly and a new approach for a proactive resource provisioning is proposed. Not only this new scheme adopts a decentralized and flat architecture where every computing node works as peers but also proposes a dynamic provisioning scheme designed with ELECTRE Multiple Criteria Decision Analysis method. Peer to peer based scheme ensures scalability as there is no central master node in the system. ELECTRE method facilitates managing resource provisioning with dynamic reconfigurable criteria on demand.

Simulation results prove the architecture offers better scalability than traditional approaches because, with the increase of nodes in the data center, the turnaround time in the proposed system is significantly less than that of centralized systems. It demonstrates that the system offers cloud service provider better opportunity to configure the provisioning mechanism which proactively checks resource wastage and Service Level Agreement violation. The proposed system invokes significantly less number of migration request compared to traditional approaches which proves that the system can manage its resources better than typical cloud infrastructures.

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.

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.

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.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

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

1.1 The Problem

Resource provisioning refers to the initialization, monitoring and controlling cloud computing resources. However, it is not a straightforward task to perform in cloud computing. There are some major aspects of pertinent to cloud computing such as initial static launch of VM (virtual machine), VM allocation to tasks, migration and replication of VM, monitoring and profiling performance etc. Not only that, horizontal and vertical scaling up and down of VMs, nodes and memory, SLO optimization, forecasting potential resource demand, cost minimization by utilizing maximum resources are important too. All of these aspects are quite interrelated to some extent.

However, extensive research efforts regarding resource and task allocation in static environment has brought about various optimization techniques. Meanwhile, it is becoming exponentially difficult to manage resources in dynamic real time environment.

Hence, adaptive resource provisioning strategies is urgent these days.

In response to the challenges, researchers have put a lot of efforts in this field. As a result, there are several dynamic provisioning and task specific provisioning algorithms in practical use (for example, Eucalyptus, OpenStack, CloudStack, Amazon EC2) which are based on single point resource monitoring and task allocation methods. These frameworks perform well in small to medium dimensioned cloud computing but struggles in performance and reliability section in case of large scale cloud and data center specific operations. Meanwhile, current enterprise and consumer market segment are constantly being attracted by the cloud services. So, maintaining service level agreements and availability of services, not only it is necessary to anticipate the resource demand early but also behave pro-actively in case of service failures.

1.2 Research Question

Considering above issues, the objective of this research is to develop an adaptive resource provisioning scheme. In this research, this following research question will be answered, How can an adaptive resource provisioning scheme in cloud computing environment be developed so that CSP can maximize their profit and end users can enjoy quality services.

More specifically,

1. how can decision making regarding resource provisioning be developed in decentralized approach while traditional cloud computing environments work in centralized approach.

2. in case of resource provisioning, what criteria will play major role.

3. how wastage of resources and potential SLO violations can be checked in a

proactive manner.

1.3 Contribution of this Research

In answering above research questions, this research contribution to a decentralized resource provisioning scheme is summarized as follows.

First, unlike traditional approaches, the control is distributed to all the computing nodes functioning in the data center. This enables each and every nodes to work with one another similar to peer to peer model where each peer takes cares of itself. This technique answers the first question stated above.

Second, as there are several dominating criteria in resource provisioning and none of those can be ignored in favour of another one. Hence, ELECTRE Multiple Criteria Decision Analysis method is proposed. This approach provides opportunity for tweaking and tuning provisioning scheme tailored to real life situations that can maximize the profit of CSPs. This scheme resolves the second and third questions stated above.

1.4 Thesis Organization

The rest of this thesis is organized as follows. Chapter two emphasizes background and motivation for this research. Chapter three highlights notable progress in this research field. Chapter four discusses the proposed system architecture. Chapter five demonstrates the result analysis. Finally, chapter six provides research achievements, limitations and potential room for improvements which illuminates the future directions from this research.

Chapter 2

Background Study and Motivation

2.1 Introduction

In this chapter a closer look has been taken on target environment-the cloud. As this research endeavor focuses on decentralization of resource provisioning, decision making algorithms especially multiple criteria decision analysis methods, a basic overview of cloud followed by an under the hood discussion regarding background of resource provisioning in cloud computing, decentralization of cloud computing and multiple criteria decision analysis is highlighted here.

2.2 An Overview of Cloud Computing

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

The remarkable achievement of cloud computing is that it has successfully con-

verted hardware/IT infrastructure, storages, development/application stacks/environments and software/applications from products to services. From technical point of view, cloud computing might be called sister of grid computing but these commercial aspects made cloud computing more attractive and revolutionary [1],

- Satisfying demand for computational resources on the fly. That results in elimination of pre-planing 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 principal, cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. The data center hardware and software is what we will call a Cloud. The primary services are categorized into IaaS, PaaS and SaaS.

Infrastructure as a Service (IaaS)

IaaS refers to Infrastructure as a Service [2]. 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 resources not as tangible products but 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 support large scale VM deployment to application environment. Other resources in IaaS clouds include images in a virtual machine 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 machines 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 an application stack which includes operating system, programming language execution environment, database, and web server. 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 user does 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 [3]. 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.

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.

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.

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 program which executes some operations and/or access data on the behalf of a user, group usually referred to as client. 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 tier 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 principal, 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 measure 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 maintains a certain level of performance. In fact, there are dedicated contracts, called Service Level Agreements (SLAs, called as Service Level Objectives or SLO too) 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 rate and request type 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 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 task in cloud computing. Planning of a resource provisioning mainly involves two

steps: workload characterization and system modeling. First one will be derived from the incoming request patterns while the second one will be derived from the application demand. In particular, system modeling is a process of associating server operation with 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 be done in a reactive manner which involves resources are updated after a workload change is detected. In both cases, the main goal is to minimize the deviation from desired QoS. In addition, resource provisioning hugely depends on workload pattern. It is usually more complicated when the workload pattern is very dynamic as, in this situation, no static allocation technique is useful.

Deployment

Back to 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 dedicated places are called data center. These days, different model of 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. In some particular cases, dedicated hosting cannot provide sufficient resource 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 resources more compared to dedicated hosting, it complicates the management situation too 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 set of logical resources that can be used by applications in the same manner as using physical ones. There are three basic functionalities are 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 created as well 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 equal 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 share policy, disk space and

network allocation.

Migration tasks denote transferring an existent VM from one physical host to another. During the 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 called teleportation too.

Provision of adequate resources for VMs is critical for high-performance data center. On the other hand, it is very important for the application hosted inside the VM 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 this situation.

Suppose, there are two server 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 physical resources for the server machine. Hence, two VMs (VM A and VM B) are created with proper configuration. In this way, both applications are served adequately and the total resource utilization of the physical machine is now increased simply by augmenting the number of running servers.

Let a scenario where workload of both application changes. In VM A, it increases while in VM B, it decreases. As the configuration of these two VM is 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 application 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 cloud computing rises exponentially in the industry and end user level, smart resource provisioning and decentralization of control are the most critical challenges in cloud computing these days.

2.4 Motivation

Cloud Provisioning [4] is the process of deployment and management of applications on Cloud infrastructures. It consists of three key steps: (i) Virtual Machine Provisioning, which involves instantiation of one or more Virtual Machines (VMs) that match the specific hardware characteristics and software requirements of an application. Most Cloud providers offer a set of general purpose VM classes with generic software and resource configurations. For example Amazon EC2 supports 11 types of VMs, each one with different options of processors, memory, and I/O performance; (ii) Resource Provisioning, which is the mapping and scheduling of VMs onto physical Cloud servers within a cloud [5, 6, 7]. It refers to the initialization, monitoring and controlling cloud computing resources. However, it is not a straightforward task to perform in cloud computing. There are some major aspects of pertinent to cloud computing, namely initial static launch of VM (virtual machine), VM allocation to tasks, migration and replication of VM, monitoring and profiling performance, horizontal and vertical scaling up and down of VMs, nodes and memory, SLO optimization, forecasting potential resource demand, cost minimization by utilizing maximum resources etc. All of these aspects are quite interrelated to some extent. Currently, most IaaS providers do not provide any control over resource provisioning to application providers [8]. In other words, mapping of VMs to physical servers is completely hidden from application providers; and (iii) Application Provisioning, which is the deployment of specialized applications (such as ERP system, mail services, web servers

and storages) within VMs and mapping of end-user's requests to application instances VM Migrations.

Moreover, popular cloud infrastructures used in data center such as Eucalyptus, Openstack, CloudStack, Amazon EC2 are all based on centralized control. That means, in a datacenter facilitated with these mentioned stacks either consists of only one node controllers with numerous slave nodes or hierarchical clustering of node controllers. However, this scheme invites 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 too. In order to avoid this problem introduce decentralization and parallelism is urgent in cloud computing.

2.5 Decision Making

Decision making is the study of identifying and choosing alternatives based on the values and preferences of the decision maker [9]. 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 [10]:

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

2.5.1 Single Criteria vs. Multiple criteria Decision Making

It is very important to make distinction between 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 [11]. 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 be used for the solution regarding linear programming, nonlinear programming, discrete optimization etc.

2.5.2 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 below. 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

Table 2.1: Multiple Criteria with weights

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

is assumed that a higher score value means a better performance since any goal of minimization can be easily 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. 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 with 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 alternative.

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 lost on another [12].

Meanwhile, the concept of outranking was proposed by Roy [13]. 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 it's 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 compromise alternative should be found by further considerations or methods.

2.5.3 Multi-attribute Utility Theory (MAUT) Methods

In most of the approaches based on the Multi-Attribute Utility Theory (MAUT), the weights associated with the criteria can properly reflect the relative importance of the criteria only if the scores a_{ij} are from a common, dimensionless scale [14]. 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 be also formalized as applying utility function.

Simple Multi-attribute Ranking Techniques (SMART)

However, Simple Multi-Attribute Ranking Techniques (SMART) [15] 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 associated with it.

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

2.5.4 Outranking Methods

The principal outranking methods assume data availability broadly similar to that required for the MAUT methods [16, 17]. 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. [18] 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.[18]. 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 $\phi^-(A_j)$ 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 [17]. 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,

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

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 outrank 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 also can 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 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.

2.5.5 Comparison

From the above discussion regarding Multiple Criteria based decision making, it can be seen that in order to apply MAUT methods, the relative weight of criteria should be in a same scale which is near infeasible in cloud computing scope. Eliminating MAUT methods from the options, there are only outranking methods left.Though, PROMETHEE outranks decisions based on head to head comparison which tends to be biased from a comparison of two options differing in huge margin . Clearly, ELECTRE provides flexibility in associating weights. Concordance and discordance relation reject any biasness which ensures the outcome decision is more acceptable. Hence, the proposed system uses ELECTRE method during resource provisioning.

2.6 Distributed Hash Table and Consistent Hashing

A Distributed Hash Table (DHT) [19] 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 with a given key. Maintenance of this 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 length 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 [20] 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 system, 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. Contrast this with a traditional hash table in which addition or removal of one bucket causes nearly the entire keyspace to be remapped.

The proposed system architecture is inspired by the advantages of DHT and circular hashing. Despite, introducing DHT based peer to peer cloud is very difficult due to multiple query based search[21], this system is the stepping stone of such possible implementations.

2.7 Summery

In this section, cloud computing environment id discussed here in a nutshell. Not only that, but also the under the hood detail about decision making and distributed hash table are also highlighted. In principal, these highlighted topics will be really insightful to understand the proposed decentralized resource provisioning in cloud computing environment.

Chapter 3

Literature Review

3.1 Introduction

The problems in resource provisioning have been studied extensively from the perspective of cloud service provider's view. All studies are eventually focused on maintaining high availability of resources on demand, achieving technical and economical flexibility regarding resource utilization, acquisition as well as release. Though, current research effort shows significant improvement in these sector, scalability and flexibility in resource configuration has not been studied much. Thus, these issues are still open research question. This chapter provides a walkthrough of research endeavors focusing typical centralized resource provisioning schemes. Later, few research efforts on decentralized provisioning will be highlighted.

3.2 Resource Provisioning in Traditional Approach

Traditional approaches performs centralized provisioning and scaling system through a single decision maker that has knowledge of all VM executing in all hosts and privilege of making decisions based on some statistical models in order to perform some certain objectives. For example, in case of setting up Eucalyptus Enterprise Cloud Environment throughout multiple PCs, cloud administrators have to define one particular node as master and rest of the other PCs as slave nodes. In this occasion, all tasks will be managed, distributed and monitored by the master node while the slave nodes perform just the operations specified by master node according to the task specifications. Much efforts has been given to the optimization techniques

regarding resource provisioning strategies in static cloud computing environments [22, 23, 24]. However, the key focus is to optimal resource allocation strategies. Here, most notable research effort in resource provisioning is discussed. It is noteworthy that here only resource provisioning from distributed system perspective is studied thoroughly. The mathematical and statistical properties of resource allocation and management are considered as black box methods as they are out of scope pertinent to this research.

3.2.1 Statistical Analysis

Zhang et al. [25] proposed a resource allocation scheme based on statistical facts named Statistical Load Balancing. It combines resource allocation and consumption prediction. It performs two critical responsibilities. The first one is to perform online statistical analysis of VM load performance and resource demand forecast. The second one is to use a load balancing algorithm which chooses a proper host from the resource pool based on the resource usage forecast and historical load information.

The proposed system is composed of VM monitoring cluster, a master node and shared storages. The overall resource provisioning scheme consists of VM monitoring cluster acting as a resource pool when VMs run, shared storages storing VMs as disk images and master nodes monitoring resource usage of cloud pool, collecting data of historical load information and resources allocation and controlling the operation of virtual machines.

The resource usage of the pool is monitored by the master node in order to keep track of resource provisioning situations. The master node collects the performance data of the hosts and VMs. These data is used to analyze and predict the true resource needs of the VMs and provide references for the allocation of pool resources. As VMs

can run on any host in the resource pool. Therefore when the master node receives a request to start the VM, it needs to allocate system resources in the resource pool and chooses the appropriate host to run the it. The basis of determining host is to maintain load balancing between hosts and reduce resource conflict and contention.

From this system architecture, it is obvious that master node bears the key responsibilities of provisioning. That implies, all the slave nodes are dependent on a single master node. However, In order to enhance the system's scalability, hierarchical scaling has been proposed by this system. The hierarchy is a full 32-ary tree. Where each parent controls its 32 slave nodes. Thus, the scalability issue has been resolved partially. Still the 32 slave nodes are centrally dependent upon corresponding master node.

3.2.2 Policy

Kephart et al. [26] from IBM research center proposed a policy prototype for resource provisioning strategies based on artificial intelligence perspective [27, 28, 29]. The primary focus of this system is to consider the cloud ecosystem as an autonomous computing system and the proposed provisioning scheme is solely focused on developing the system as self-management capable rational agent.

The framework is targeted for data center scenarios which consists of Application Environments (AE) serving the consumers. AEs are logically separate from each other and have a dedicated pool of resources. Each AE is managed by an Application Manager (AM) while there is a Policy Repository (PR) which is responsible for implementing various policies. Each AM continually adjusts resource usage in accordance with its policies. When demand changes significantly, some AMs may not be able to adequately implement their policies with available resources. In this sort of

cases, they can appeal to the Resource Arbiter for additional resources. The Resource Arbiter handles such resource requests. Under some circumstances it might even remove resources from one AM and provide that to another AM if it maximizes the overall utility or reach the goal. The resource allocation strategies are based on three key perspectives of modern AI strategies.. These are goal, utility and action based allocation strategies. The hybrid policies combined of these three are also possible. Theoretically, this framework is successful as self-manageable resource provisioning agent. However, the system is inherently not scalable due to the dependency of global resource arbiter.

3.2.3 Reinforcement Learning

In the literature, usage of Reinforcement Learning (RL) in this sector has been extensively researched [30, 31, 32]. Theoretically, RL methods provide application environment better opportunity to be adaptive enough to cope with the dynamic nature of cloud computing environment [33]. From the system architecture point of view, all of the proposed frameworks are more or less identical with variations in the methods regarding how RL techniques are used.

For example, Gerald Tesauro et al. [30] proposed a resource allocation scheme based on hybrid RL. In principle, RL can automatically learn high quality management policies without an explicit performance or traffic model and with little or no built in system specific knowledge. The proposed system consists of mainly a Resource Arbiter (RA) and Application Managers (AM). AM performs the responsibility of optimization of resource usage in accordance with SLA, meanwhile RA performs the duty of marshaling resource needs demanded by application environments.

In this model, the optimization goal of each application is defined by a local

performance-based objective function, which is called expected business value. The goal of RA is to allocate applications among servers in order to maximize the sum of expected business value over all applications implying that all local value functions share a common scale. A fixed five-second time intervals is used to make decisions regarding resource allocation issues. Each Application Manager reports to the Arbiter with an estimated business value as a function of the number of allocated servers from the aspect of application's current state. It is the proposed assumption that business value of an application is defined in monetary units by net expected revenue according to the SLA payments or penalties of an application as a function of one or more performance metrics such as operational cost, availability and service consistency. It is noteworthy that the estimated business value continually changes as the application's states and load levels fluctuate, so that they have to be continually recomputed.

Having received the current business value from each application, then RA computes the globally optimal allocation maximizing total expected business value (i.e. total SLA revenue) summed over all the applications. Moreover it distribute a list of assigned servers to each application, which are then used in a dedicated fashion until the next allocation decision. The decision making process is controlled by a hybrid RL mechanism proposed in this system named SARSA 0,

- observing the system's current state s_t at time t
- performing some legal action a_t in state s_t
- receiving a reward r_t (a numerical value that the user would like to maximize) followed by an observed transition to a new state s_{t+1}

Although, the system behaves adaptively in the ever changing scenario and train itself up to the tasks, yet again it lacks scalability issues due to the dependency upon a single global arbiter.

3.2.4 Utility Function

Hien Nguyen Van et al [34] proposed a centralized autonomic resource management system for service hosting platform. Significant achievement of this system is the separation of resource provisioning and dynamic VM placement. It focuses on fulfilling mainly three issues. The first issue is to automate the dynamic provisioning and placement of VMs considering not only SLA but also resource exploitation costs. The second one is to support heterogeneous applications and workloads. The third one is to support arbitrary application topology.

The proposed system depends on two level architecture. There is a distinct separation of concern between application specific responsibilities and global decision making responsibilities. The model consists of Application Environments (AE), Local Decision Module (LDM) and Global Decision Module (GDM). AE encapsulate the application hosted in the cloud systems. The performance goal of a particular AE is tailored according to the SLA restriction of the corresponding application.

A LDM is associated with each AE. It evaluates the opportunity of allocating more VMs or releasing existing VMs to/from the AE on the basis of the current workload using service-level metrics such as response time, number of requests per second etc. coming from application-specific monitoring probes. The main job of the LDM is to compute a utility function which gives a measurement of application satisfaction with a specific resource allocation (CPU, RAM) given its current workload and SLA goal.

Meanwhile, GDM interacts with LDMs. It is the decision making entity which continuously performs inside an autonomic controlling loop. It arbitrates resource requirements coming from every AE and treats each LDM as a black-box without being aware of the nature of the application and the way the LDM computes its utility function. It receives the utility functions from every LDM and system-level performance metrics (e.g. CPU load) from virtual and physical servers as input. The

output of the GDM consists of management actions directed to the server hypervisor and notifications sent to LDMs. Management actions include the initializing, suspending, resuming, stopping and reconfiguring VMs as well as VM migration and packing. Upon receiving notifications from GDM, LDM understands that a new VM with specific resource capacity has been allocated or an existing VM configuration has been changed for example, its class, resource capacity or priority in job execution queue.

This architecture successfully implements separation of concern regarding resource provisioning and VM placement. LDM handles the issues of resource provisioning while GDM arbitrates VM across the environment. However, only one node in the ecosystem acts as GDM, hence, scalability is not ensured when the number of nodes in the data center rises exponentially.

3.3 Decentralized Resource Provisioning

From the previous section, it is comprehensive that all of the proposed architectures consist local agents and global agents. Local agents compute decision makings based on utility values or statistical analysis of each VM performance whereas global agents computes and applies configuration on the behalf of the whole data center. Moreover, previous research efforts are solely focused on maximization of physical hosts own utility value by means of statistical analysis, combinatorial optimization or RL based techniques. However, this architecture is quite effective and efficient for small data center scenario however, it fails to ensure scalability, flexibility due to its centralized control nature. This situation raises the need to explore the opportunities to introduce decentralized decision making process. The advantages of decentralized decision making have been researched extensively. In this section, noteworthy steps

forwards are discussed.

3.3.1 IBalloon

Proposed by Jia Rao et al. [35], this mechanism is based on distributed learning approach targeting the VM performance improvement through self-adaptation to the dynamic scenario of the data centers. According to this approach, cloud users manage individual VM capacity and ask for resources based on application demand. The host agent which manages all the VMs evaluates the aggregated request and provide feedback to individual VMs. Based on those feedback, VM learns eventually to manage its own policy. Each VM uses RL approach to manage its own capacity. The learning agent operates on each VM while it is running. In order to manage capacity measurement, the system synthesize application profiles and resource utilization. Via feedback, it punishes if there is SLA violation and provide incentives to release unnecessary resources.

In principal, IBalloon comprises of three key components. These are Host Agent (HA), App Agent (AA) and a Decision Maker (DM). HA is responsible for allocating proper hardware resource to each VM and providing feedback. AA maintain SLA profile and application performance in run time. DM hosts a learning agent for each VM for autonomic capacity management. There are two vital assumptions regarding self-adaptive capacity management. Firstly, data obtained from a VM in execution is the primary source of information regarding obtained from a VM in execution is the primary source of information regarding capacity management decisions. Secondly, VMs entirely depend on feedback which are based on previous capacity management or revised policy imposed by the learning agent.

Data obtained while VMs are running mainly includes current utilization of CPU,

memory, IO resources. The feedback signal is designed in such a way so that it punishes a resource allocation action explicitly if it causes SLA violation. Hence, it encourages to free up required resources. This feedback signal is actually a real value called reward. The value of reward is set to negative (usually -1) when there is contention for resources among the VMs and otherwise, 0 if there is no contention. Thus, some of the competing VMs backs up via releasing resources. This RL process is governed by Markov Decision Process. Given predefined environment states S, a set of actions A, MDP is governed by transition probability distribution and reward functions. At each time step, the agent transits into next state and receives an immediate reward.

Never the less, this system is scalable as there is no global decision maker and adaptive as it reacts according to the provided feedback, the system is inherently confined to the limitation of transitional function and choice of best possible actions to choose. As MDP [36] approach is temporal, this system is a discrete intelligent agent while real time cloud environment is continuous. Moreover, this scheme is more reactive than proactive because, this system focuses on punishing on undesired action rather than minimizing the probability of occurring negative events.

3.3.2 Distributed Provisioning and Scalable Management System (DPSMS)

Trieu C. Chieu et al. [37] proposed a scalable architecture of distributed provisioning and scalable management system. Decentralize decision making is the key feature of this research. The system includes front end load balancer, a pool of physical hosts, provisioning and scaling management system and a service monitoring system with dynamic scaling algorithm. The front end load balancer distributes computa-

tional jobs to suitable VMs from resource pool. Resource pool consists of multiple PMs each hosting numerous VMs. Each PM continuously runs a capacity and utility agent (CUA). This agent will continuously try to maximize its own utility along with Distributed Capacity Agent Manager (DCAM) which is a light weight agent management system responsible for managing and communicating with the participating agents and directing the resource adjustment actions to the target systems.

Each VM sends capacity and resource information to CUA and maintains a system profile which reflects the characteristics of the hosting physical machine of the VM. From this profile, it generates a capacity index (CI). Its range is from 0 to 1. 0 means the host is idle while 1 means the host is fully utilized. Thus, each physical hosts indicates its current status and based on the CI, it decides whether it accepts a new VM or migrate its own VM to another host. Each creation and migration of VM triggers recalculation of CI of each hosts. Physical hosts notifies the current value of CI if it changes. The DCAM maintains a central database of CI from each physical host. Thus each physical host maximizes its own utility and that converges to a scalable resource provisioning in data center scenario. Still, DCAM uses a central database for storing CI that makes the system inherently centralized and thus difficult to scale up.

3.3.3 Multiple Criteria Decision Analysis (MCDA) Method

Onat Yazir et al [38] proposed a decentralized resource provisioning scheme based on multiple criteria decision analysis using PROMETHEE [18] method. Flexible configuration and scalability is the questions that are addressed in this research. The system includes a distributed network of Node Assistants (NA). Each physical

machines (PM) is tightly coupled with one NA. It is assumed that each NA is capable of accommodating VM and delegating the VM to other PM when necessary. Not only that, it also maintains a global awareness of the resource availability and task management through an oracle. Simply, each NA continuously monitoring local usage pattern of PMs and if any sign of over-usage or under-usage of resources is noticed, NA will find out the problematic VM and choose a potential suitable node to migrate the VM. This selection process of most suitable node to migrate the VM is done by PROMETHEE multiple criteria decision analysis method. Although this proposed system is inherently scalable as there is no master node, however the scalability is in question as the oracle proves to be yet another centralized implementation. Despite the cons, the proposed system discussed in the next chapter gets the inspiration from this research effort.

3.4 Summery

From this chapter, it is easily understood that much research effort is focused on maximizing the utilization of resources in terms of computational power, SLA conditions and costs. However, none of these endeavor comes forward much to resolve the particular issues rises when the data center is scaled exponentially. Thus it is much needed to resolve this open research question regarding resource provisioning in cloud computing environment.

Chapter 4

System Architecture

4.1 Introduction

This chapter demonstrates step by step details of the proposed system. In the first section a higher level detail is presented. In the subsequent section, the under the hood details are discussed extensively.

4.2 A Bird's Eye View

The system is solely focused on ensuring the scalability and robustness of the data center environment in cloud computing. Hence, the architecture is based on eliminating the scalability issues which are usually the byproduct of using global optimal configuration and centralized resource arbiter. As the expansion of the data center capacity is ever imminent in real life scenario, the dependency upon a lone resource arbiter might pose a serious threat of single point failure.

For example, if a data center has more than thousands of nodes and a single resource arbiter, no matter how powerful computational resources are granted, the arbiter will just struggle in the long run. Suppose, a data center has N number of nodes with a single resource arbiter. Consequently the resource arbiter will have to take care of deployed tasks, proper utilization of the resources, collecting and providing feedback to the nodes, handling VM migration and so forth. Moreover, horizontal scaling of the data center will complicate the scenario further. Although it is unlikely to happen, during the operational state the resource arbiter might fail due to physical damage or power outage. Excessive amount of jobs which is beyond the

capacity of resource arbiter culminates in slower response time and thrashing in the worst case. As a result, the whole data center will become nonfunctional and lead to unacceptable throughput, SLA violation, financial loss and irreparable consumer dissatisfaction.

So the prime target is to achieve these following goals,

- to eliminate the single point dependency on global resource arbiter
- to find out a viable alternative to centralized resource provisioning scheme
- to bring about efficiency and flexibility in provisioning scheme

The solution to the first issue is quite straightforward. The proposed system frees the global resource arbiter from the duty of managing all the nodes. Instead, the responsibility of maintaining each node is delegated to itself. This means, the node will maintain the profiling information of the applications currently running inside it, look after its own resource utilization issues and handle migration when necessary.

Nevertheless, delegating each nodes its own responsibility does not eventually eliminates the need of a resource arbiter. As each node needs to know about various information of other nodes in the data center, they need a source for obtaining such knowledge. So, the system proposes the concept of a global node information provider which will maintain global awareness in the data center.

Still, if the whole system depends on just a lone information source, then the data center will be vulnerable to the threat of single point failure. Hence, a decentralized network of information providers replaces the central entity. Eventually, this achieves the second goal as well.

Last but not the least, the system proposes a new intelligent policy for migration of Virtual Machines (VM) that minimizes the undesired re-migration of VMs as well as provides opportunity of tweaking and tuning migration criteria.

4.3 The System Architecture

The system architecture is composed of four major subsystem. These are Application Agent (AA), Node Agent, Datacenter Agent (DA) and a Job Pool (JP).

AA is an entity that is tightly coupled with each application that is submitted to the data center. As the resource requirement 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 considered as application or task unit and assigned to Physical Machines (PM) (or physical host or simply host or node) which has the ample resource to host and run those.

Every PM in the data center 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 data center. When the corresponding PM is incapable of hosting one or more VMs, NA reconfigure and migrate those problematic VMs.

DA will act as the global information provider. Whenever a NA needs information of its neighboring nodes, DA will provide that information. This information is simple. For example, answer of a simple query such as which is the first node that is running at $x\%$ resource utilization rate or which is the set of nodes that are currently idle etc. It is mentionable that DA is implemented as circular hash based decentralized network of multiple nodes which will be discussed later in this chapter.

Last but not the least, the system maintains a pool where unallocated VMs are stored. When an application is submitted to the data center 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* VM here and allocate it to a suitable PM.

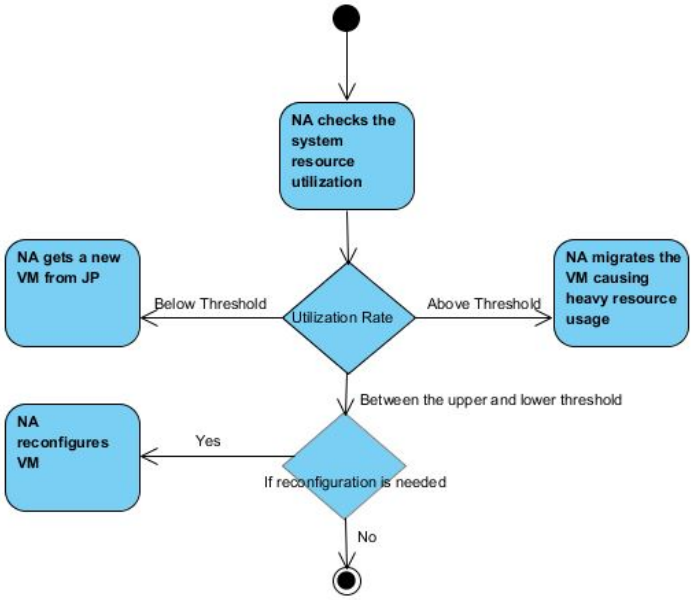

Figure 4.1: Overall responsibility of a NA

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 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 performs several actions in order to maintain the resource utilization of its host to a desired level. These following tasks are executed sequentially and consistently 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 if 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 accommodate it. Moreover, NA will continuously monitor the performance of VMs hosted by the same PM too. If NA detects one or several

VM behaving anomalously, it will first try to 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 possible due to system resource constraints, NA will stop the VM and invoke a migration request to the DA. Interestingly, 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 DA for a neighboring PM suitable enough to host that problematic VM.

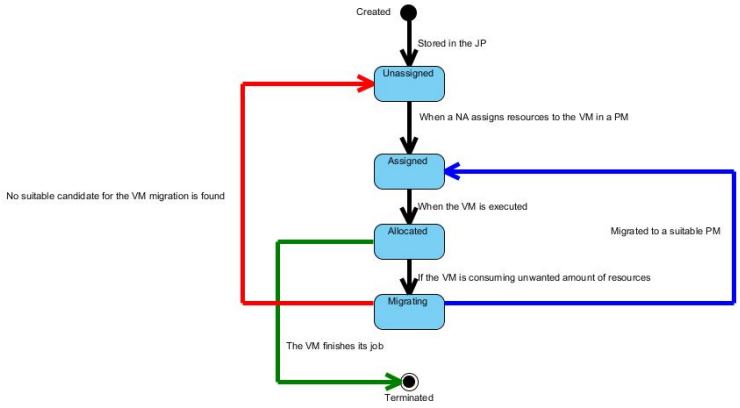

Figure 4.2: VM State Machine Diagram

Then NA will be provided necessary information of neighboring PMs from DA. Then it will compute the best suitable PM for this particular VM. The computation can be based on various criteria. But single 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 to allocate its resources before actually delegating the task. It also changes the status of the VM to *migrating*.

Meanwhile, 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

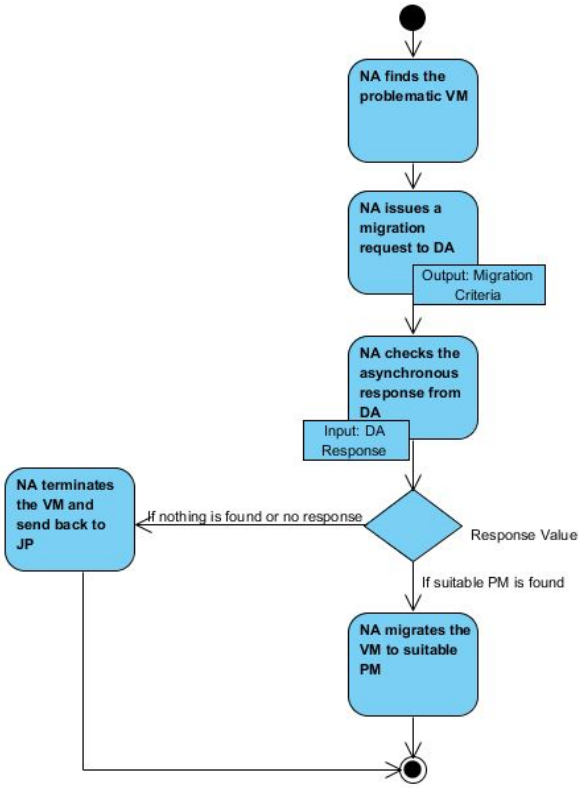

Figure 4.3: Migration Process

maintain a timeout value until which it maintains the lock. If it does not get the VM in that time window, it will cancel the lock and release the resources.

NA keeps a timeout value until which, it will await the reply from DA. 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.

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 one is to determine which VM is the root of anomalous behaviour and the second one is to decide in which PM the VM will be migrated.

The computation regarding first step is not one dimensional. The key issues needed to be considered here are maximization of resource utilization and minimization of migration cost, Service Level Agreements (SLA) violation and further proba-

bility to migrate the VM (from other node). Since the criticality of these issues varies upon the nature of application, workload and data center configuration, hence the problem can be modeled as multiple criteria decision problem. Yet again, the key issue regarding the second step is not only to find the nodes having available computational resources but also maximizing the resource utilization and minimization of the potential chance of re-migration. These factors can be considered as criteria too. In order to find out the best decision, these criteria can be fed into ELECTRE [13] MCDA algorithm with weights that denote how critical each criteria is.

Finally, DA is built as a decentralized network of nodes. It has been already mentioned that the key task of DA is to provide information of neighboring nodes. It is simply understandable that the probability of new VM allocation or migration request will get higher with the increase of number of VMs or PMs in the data center. Though, the DA has much less to do because it is relieved from the duty of provisioning each and every node (i.e. the task of provisioning is now done by NA), it will struggle to cater to the demands when the data center is large. Hence the choice of introducing decentralized DA is justified regarding this context.

DA consists of numerous nodes which collectively form a decentralized network. The relationship between DA nodes and NAs is defined via consistent hashing. A DA node is allowed to accept request from certain number of NAs. Hence, if there is M number of NAs in the data center and N is the maximum allowed number of NAs per DA node, the total number of DA nodes in the system will be $\lceil \frac{M}{N} \rceil$. The relationship information is a simple key value pair between NA and DA node. Each NA has a unique identifier and it will be mapped to the corresponding DA node according to the consistent hashing method. As the number of PMs in the system is not static, the total number of DA nodes is dynamic too. Thus the use of circular hashing is justified as it imposes an approximate cost of $\frac{K}{B}$ number of rehashes where

K is the number of keys (in this context, NAs), B is the number of buckets (in this context, DA nodes). Meanwhile typical hashing function requires almost all keys to be rehashed. This mechanism facilitates the system to dynamically change the value of M and N without affecting the performance.

The way DA nodes handle the requests issued from NAs is simple too. When a DA node receives a request from NA, it will first look for the nodes (only those that are assigned to it) that satisfy the conditions. Not only that, it also communicates with other DA nodes for that information too. When the DA node receives all the information from its peer DA nodes, it will return the list of PMs with necessary information to that particular NA.

4.4 Summery

In principal, this system is a group of intelligent autonomous agents which can collectively computes better decision regarding provisioning and maximizing the performance of application executing in the data center. Controls are distributed among all the peers and thus, it is ensured that the system will not come to a halt unless all the computing entity go down.

Chapter 5

Experiments and Results

5.1 Introduction

This chapter first highlights the simulation procedures as well as measurement criteria and then demonstrates the outcomes of the experiments.

5.2 The Experiment

For this experiment, a simulation program is developed using Common Language Runtime. The simulation program simulates the changes in VM resource requirements according to statistical distributions. The simulation is done in a step wise manner where each step represents a constant time interval. Every VM declares new resource usages at each step. In this experiment, the resource requirement is composed of only CPU and memory demand.

In the simulation process, every application will be assigned to exactly one VM and then the VM will be assigned to a suitable PM. The application are represented as two vectors of CPU and memory demand. The length of the vector denotes the lifetime span of that application i.e. steps is needed to finish the application lifetime. For example an application's CPU vector contains these following integers; 10, 20, 25, 24, 23, 30, 5, 1. That means its lifetime is of 8 steps and it demands 10 unit computational resources in the first step. The CPU and memory demand are generated according to the normal and exponential distribution with random mean and variances. Besides generating data as statistical distribution, consistent demand with sudden infrequent peak is generated too. This type of demand simulates very

low probability of migration needs. Thus, each application will have two vectors of CPU and memory demands as well as mean and variances of them. In the simulation program, each application will be mapped to a VM with CPU and memory capacity equaling to standard means of corresponding demand. By this process a PM will accommodate as many VMs as it can host. That means in ideal case, CPU of a PM will be greater than the sum of CPU demands of all accommodated VMs. The case is also identical for memory requirement as well.

In each step, NA will monitor if the demand of VMs can be met by PM. If VM demands outnumber its specified mean requirement, the NA will look if the PM still has enough available computational resources. If so, then the NA will provide the VM additional resources or, it will just look for a peer PM which can accommodate the VM.

When a NA send the request for information regarding the most suitable peer for VM migration to its corresponding DA node, it will await the reply for a constant number of interval. And for each request from NA, the DA node will take a constant number of steps to process it. It is quite easy to comprehend, if more than one NA will request a single DA node, other NA will certainly have to wait for a certain amount of time for the reply. So, if a single DA is used, then obviously the throughput of the application will increase with respect to the number of total PMs in the data center.

in this simulation program, the throughput vs. the number of VMs in the data center will be the key focus. The comparison will be done for (a) centralized conventional resource provisioning scheme where the single resource arbiter will do all the resource provisioning management; (b) the proposed system with a single NM and (c) the proposed system with decentralized circular hashing based network of DA.

The centralized system is implemented in such a way so that the single global resource arbiter will assign all the unallocated VMs in the system as well as handle

the migration request coming from each PM. In principal this means the arbiter is working as much as tasks collectively completed by all the NA in the proposed system model.

Apart from analyzing throughput, experiment of performance analysis among proposed ELECTRE method will be tested against First Fit (FF) and First Fit Decreasing (FFD) method regarding the choice of best possible node that will be chosen to allocate an unassigned VM or migrate a problematic VM. The target is to observe how each method perform in the terms of number of migration with the increase of VMs. Obviously the less the migration numbers are, the more efficient the method is. The input of ELECTRE method is CPU and Memory demand with equal weight. Meanwhile, the FF and FFD methods are pretty much self-explanatory. During the migration phase, when a NA is looking for a node to offload a problematic VM, FF method will select the first node from the possible candidates and FFD method will select the first node after sorting the nodes by their resource availability. This experiment includes migration vs number of VM and migration vs. number of iterations.

5.3 The Results

First experiment is run on for (a) very low pseudorandom migration request, (b) random resource demand based on normal distribution and (c) random resource demand based on exponential distribution. The horizontal axis denotes how many VMs are submitted in the data center while the vertical axis denotes how many time step is needed to finish all the submitted tasks in the data center.

Regarding case (a), when the migration request probability is set to very low for the sake of experimental purposes, it has been seen that centralized system is keeping pace with the proposed system with single DA and multiple DAs. If closely observed,

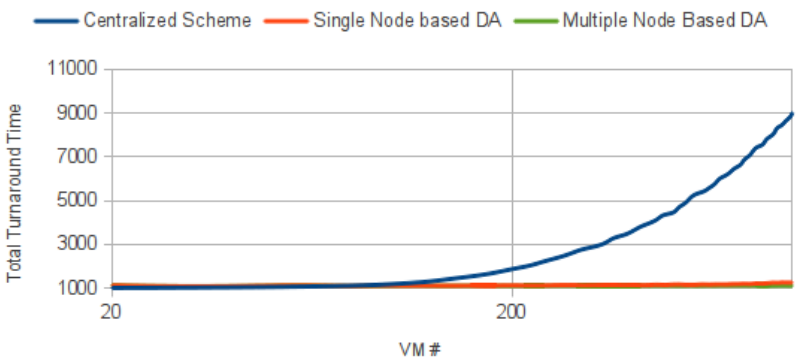

Figure 5.1: Turnaround Time vs. VM at Infrequent Short Burst

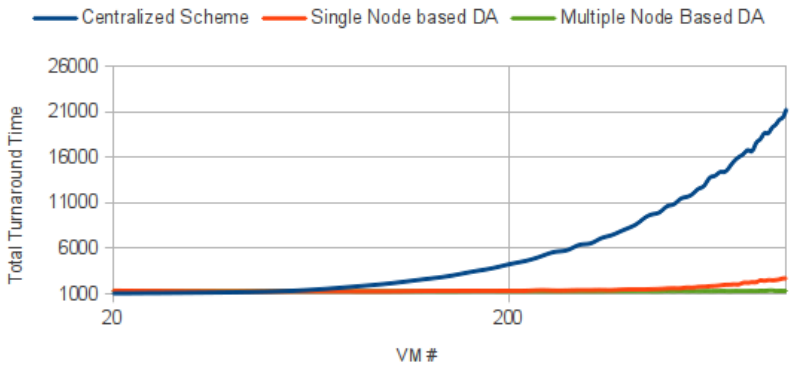

Figure 5.2: Turnaround Time vs. VM at Normal Distribution

it can also be found that, initially centralized system's throughput is better than that of proposed decentralized system. The reason is simple. Initially, when the total number of tasks in the datacenter is quite low and as long as migration rate is lower than the capacity of the centralized server, it should perform better. However, as the number of tasks rises, the rate of incoming migration request from PMs quickly outpaces the capacity of the global resource arbiter. Hence, the decentralized system proves to be more efficient in the long run.

However, in a real time data center scenario, although initial static assessment of VM allocation to PM is efficient as well as tailored as per the SLA and cost assessment, need of migration scenarios will arise frequently with the rise of number of VMs in the data center. This situation is simulated in case (b) and (c) where the computational

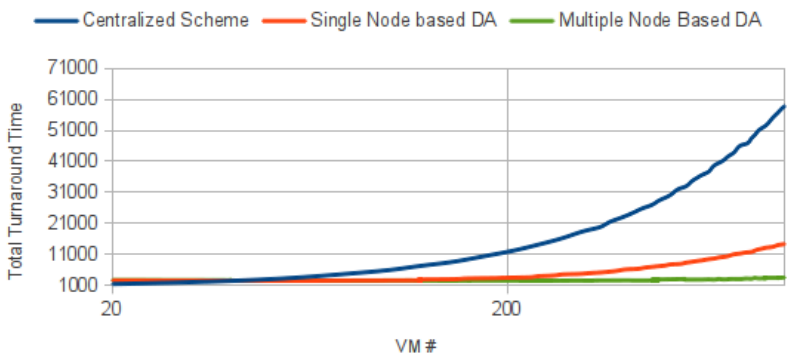

Figure 5.3: Turnaround Time vs. VM at Exponential Distribution

resources are of normal and exponential distribution. This distribution will generate higher probability of migration request. In this extreme situations, the centralize system performs poorly against the proposed system.

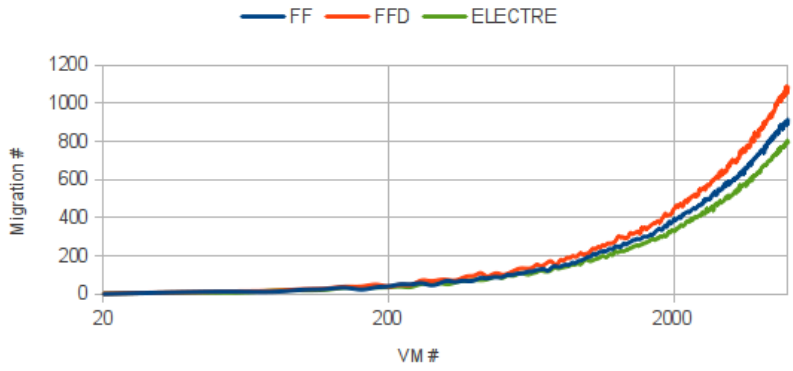

Figure 5.4: Migration vs. VM at Infrequent Short Burst

The simulation reveals another key information regarding the performance of single node DA and multiple node DA. In normal distribution based migration invoking scenario, the single node based DA is performing slightly less than multiple node based DA. However, in the scenario of exponential distribution, single node based DA lags behind with the increase of VM number. From this two scenarios, most prominent factor is multiple node based DA is suffering from sequential bottleneck problem. This is the reason why that explains the fact that when the migration load is not significantly high, the single node DA matches the performance with multiple

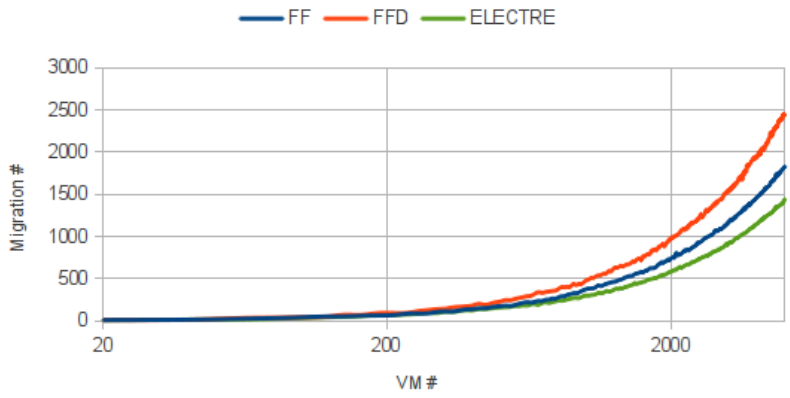

Figure 5.5: Migration vs. VM at Normal Distribution

node based DA. The sequential bottleneck problem occurs when a DA node receives a migration request and start searches for suitable options but it has to communicate with other neighboring DA nodes too. Thus, until the other DA nodes returns their result, this particular DA node has to wait for their answer and thus, the bottleneck situation occurs. However, in significantly high migration rate, request processing capacity a single DA node is outpaced by the incoming migration rate from all the NA in the data center. Thus the situation becomes a single point bottleneck problem identical to the situation of a traditional centralized server. Hence, the multiple node based decentralized network of DA provides good performance.

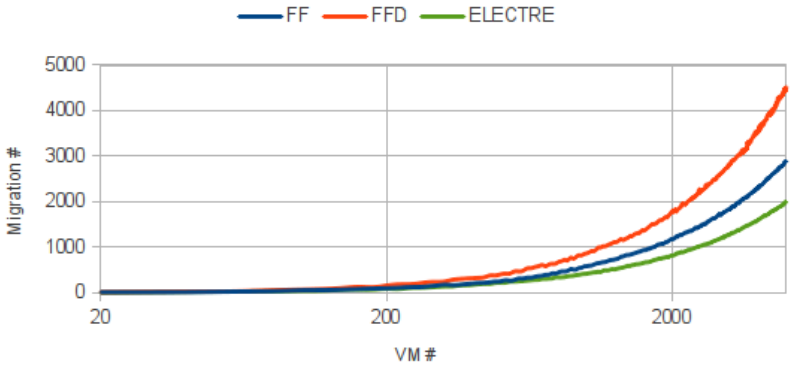

Figure 5.6: Migration vs. VM at Exponential Distribution

Meanwhile, as previously mentioned, the system architecture is primarily focused

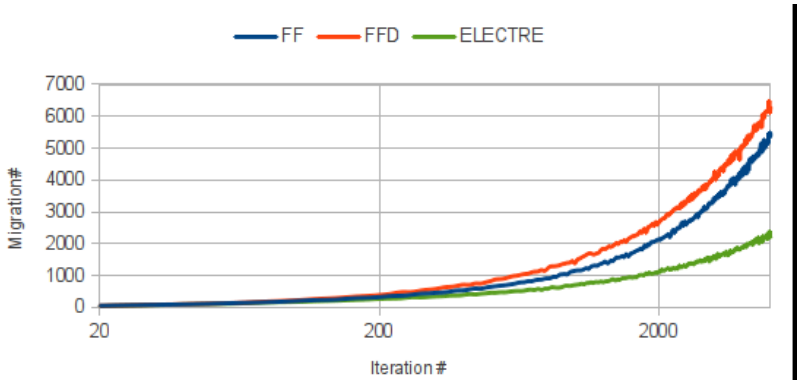

Figure 5.7: Migration vs. Iteration at Infrequent Short Burst

on increasing the efficiency of resource provisioning. Practical scenarios demand frequent need for VM reallocation, reconfiguration and replacement for both unassigned and running VMs. However, considerable efficiency can be achieved through proper migration scheme so that need for further migration is minimized .The second experiment directly reflects on this issue.

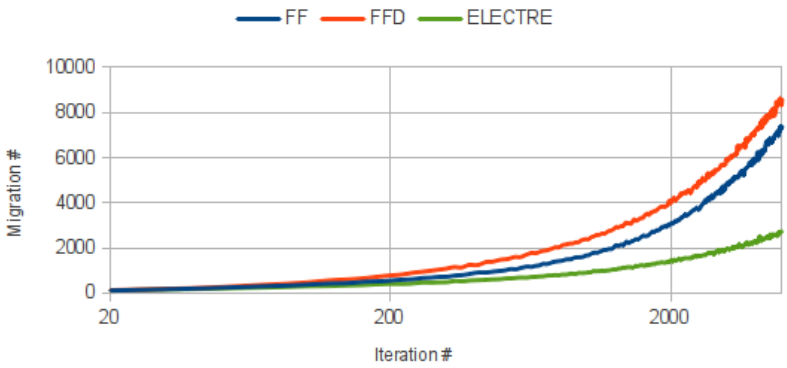

Figure 5.8: Migration vs. Iteration at Normal Distribution

The first sub-experiment highlights the efficiency of migration scheme with respect of the total number of VMs in the data center. In all probability distribution, it is clearly seen than FFD performs worst. Meanwhile, the difference between FF and ELECTRE method is not such huge but with very high number of VMs, ELECTRE method pulls out a reasonable lead over FF method.

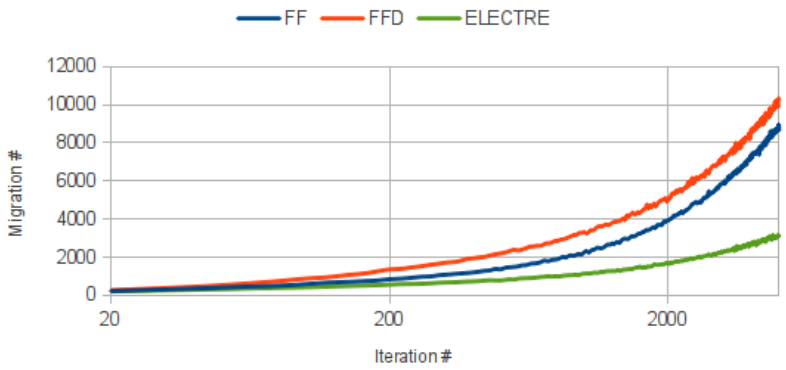

Figure 5.9: Migration vs. Iteration at Exponential Distribution

The second sub-experiment measuring migration efficiency with respect of iteration number provides more insight on the advantage of ELECTRE method over simple FF and FFD method. FFD is once again the worst performer similar to the first sub-experiment however this time, ELECTRE builds up a far wider lead margin over FF method.

In principal, the bottom line from these experiments are,

- The system proves to be more scalable than the traditional system. As the turnaround time is much less in the proposed system than that of the traditional approach.
- The system shows that the number of migration is significantly less than that of FF and FFD methods. It proves that the resource wastage is much less in the proposed system.
- The ELECTRE method enables tweaking and tuning of the critical criteria. Thus, it facilitates the cloud service providers to dynamically decides the weight of the criteria according to their needs.

5.4 Summery

The experiments prove that the proposed method is on the right track in achieving the research goal. It ushers a new window of opportunity to configure the application performance and data center utilization on the fly. Despite the promises it offers, it should be noted that this system has to make a long way to be a perfect replacement of the cloud computing infrastructures of today. This issue is discussed in the following chapter.

Chapter 6

Discussion and Conclusion

In principal the system achieves the key goals specified in the previous chapter. Although, the implementation enlightens with the notion how the system would perform in real life scenario, actual situation is far more complex than the issues considered in this research effort. For instance, in migration scenarios, only two criteria is considered but in practical implementation, resource utilization, service level agreement, migration penalty, network bandwidth issues are critical enough to be included in each and every decisions at provisioning schemes.

Moreover, the system focuses more on aftereffect or resource provisioning rather than application to VM task mapping as well as minimizing the need for migration scenarios. Hence, in the implementation, the migration signals are randomly generated which is not an ideal case in pragmatic situations.

It is obvious from the result postmortem that due to sequential bottleneck, unless the VM number is very high, the DA consisting multiple nodes is not performing well as expected theoretically. For example, if n number of node works as DA collectively, it should be theoretically n times faster than a single node based DA. However, they are supposed to work in parallel but still the system design performs poorly due to low level of parallelism. This situation can be explained by Amdahl's law. [39] According to this law,

$$S = \frac{n}{1 + \alpha(n-1)} \tag{6.1}$$

a system's parallelism (S) is bounded by the reciprocal of a constant (α) which is equal to the percentage of sequential, blocking job in the critical path of total workflow sequence.

As previously mentioned, the simulation program is built with C# language on Common Language Runtime stack. It generates pseudorandom numbers based on default implementation defined by the framework using hardware dependent method which shows poor randomness property in comparison with computational and statistical methods. This is the most prominent reason behind the zigzag effects at the curves in the simulation results.

Overcoming these mentioned limitations paves a broader opportunity to future research work. In addition, MapReduce framework can be used for efficient parallel workflow management of multiple node based DA. The broader vision of this research endeavour is to develop a Distributed Hash Table based cloud where every entity will be a peer and completely free of centralized provisioning schemes.

Bibliography

[1] 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.

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

[3] 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.

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

[5] P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 289–302, 2007.

[6] D. Kusic and N. Kandasamy, "Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems," *Cluster Computing*, vol. 10, no. 4, pp. 395–408, 2007.

[7] M. Bennani and D. Menasce, "Resource allocation for autonomic data centers using analytic performance models," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pp. 229–240, IEEE, 2005.

[8] R. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and qos in cloud computing environments," in *Parallel Processing (ICPP), 2011 International Conference on*, pp. 295–304, IEEE, 2011.

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

[10] 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

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

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

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

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

[20] 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.

[21] R. Ranjan, L. Zhao, X. Wu, and A. Liu, "Peer-to-peer cloud provisioning: Service discovery and load-balancing," *CoRR*, vol. abs/0912.1905, 2009.

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

[23] X. Wang, H. Xie, R. Wang, Z. Du, and L. Jin, "Design and implementation of adaptive resource co-allocation approaches for cloud service environments," in *Advanced Computer Theory and Engineering (ICACTE), 2010, 3rd International Conference on*, vol. 2, pp. 484–488, IEEE, 2010.

[24] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, p. 5, ACM, 2011.

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

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

[27] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," 2nd International Workshop on Policies for Distributed Systems and Networks, 2001.

[28] M. Bearden, S. Garg, and W. Lee, "Integrating goal specification in policy-based management," *Policies for Distributed Systems and Networks*, pp. 153–170, 2001.

[29] J. Strassner, "How policy empowers business-driven device management," in *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on*, pp. 214–217, IEEE, 2002.

[30] G. Tesauro, N. Jong, R. Das, and M. Bennani, "On the use of hybrid reinforcement learning for autonomic resource allocation," *Cluster Computing*, vol. 10, no. 3, pp. 287–299, 2007.

[31] G. Tesauro, "Online resource allocation using decompositional reinforcement learning," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 20, p. 886, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[32] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow," in *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, pp. 67–74, 2011.

[33] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, vol. 1. Cambridge Univ Press, 1998.

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

[35] J. Rao, X. Bu, C. Xu, and K. Wang, "A distributed self-learning approach for elastic provisioning of virtualized cloud resources," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pp. 45–54, IEEE, 2011.

[36] S. Russell, P. Norvig, J. Canny, J. Malik, and D. Edwards, *Artificial Intelligence: A Modern Approach*, vol. 2. Prentice Hall Englewood Cliffs, NJ, 1995.

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

[38] Y. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady, "Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 91–98, IEEE, 2010.

[39] K. Hwang, *Advanced computer architecture: parallelism, scalability, programmability*, vol. 199. McGraw-Hill Singapore, 1993.