Virtual Machine Placement in Cloud Environment

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS by Research
in
Computer Science and Engineering

by

Dharmesh Kakadia
201107616
dharmesh.kakadia@research.iiit.ac.in

Search and Information Extraction Lab
International Institute of Information Technology
Hyderabad - 500 032, INDIA
July 2014

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Virtual Machine Placement in Cloud Environment" by Dharmesh Kakadia, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. Vasudeva Varma

To My Parents, Sisters

and

My Teachers

Abstract

Cloud computing has allowed the consumption of services over internet with subscription based model. Based on the level of abstraction of services, there are different models of cloud computing like Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). This model of service consumption is extremely suitable for many workloads and cloud computing has become phenomenally successful technology. Cloud computing allows users to remove the upfront infrastructure cost and pay-for-what-they-use. Cloud providers multiplexes resource requests from multiple users through use of virtualization. It becomes essential for cloud service providers to operate very efficiently in multiplexing at the scale to remain profitable.

The growth of cloud computing has lead to massive data centers with millions of servers. The resource management at this scale is pressing issue. Scheduling is responsible for arbitrating resources and is at the heart of resource management. The issue of efficiency at this scale and emerging model of consumption of cloud services require new approaches and techniques to be applied to the age old problem of scheduling. Virtual machine is the basic unit of scheduling in data centers. In this thesis, we deal with problem of virtual machine scheduling over physical machines. We aim to understand and solve the various aspect of scheduling in cloud environments. Specifically, we leverage various fine grained monitoring information in making better scheduling decisions. We used learning based approach of scheduling in widely different environments.

There is increasing concern over energy consumption by cloud data centers and cloud operators are focusing on energy savings through effective utilization of resources. At the same time, maintaining SLAs is also very important for them for the performance of applications running. We propose algorithms which try to minimize the energy consumption in the data center duly maintaining the SLA guarantees. The algorithms try to utilize least number of physical machines in the data center by dynamically rebalancing the physical machines based on their resource utilization. The algorithms also perform an optimal consolidation of virtual machines on a physical machine, minimizing SLA violations.

In large virtual data centers, performance of applications is highly dependent on the communication bandwidth available among virtual machines. Traditional scheduling algorithms either are not network-aware or are not practical for large data centers. We address the problem of identifying the virtual machine clusters based on the network traffic and intelligently placing them to improve the application performance and optimize the network usage in large data center. We describe a greedy consolidation algorithm to ensure that the number of migrations are small and the placement decisions even for large data centers. We observed significant improvements in the applications performances and reduced number of migrations on real world data center traces.

The wide spread use of mobile devices and cloud services has led to the growth of mobile cloud. Even tough mobile devices are becoming increasingly more powerful, the resource utilization of rich mobile applications can overwhelm resources on these small and battery constrained devices. In this context, ubiquitous connectivity of mobile devices also opens up the possibility of leveraging cloud

resources. Automatically running mobile application on cloud instead of mobile device can greatly aid to battery life of the device. We describe a cloud aware scheduler for application offloading from mobile devices to clouds. We used learning based algorithm for predicting the gain attainable using performance monitoring and high level features. We evaluated prototype of our system on various workloads and under various conditions.

Big data processing is one of the most popular use case of cloud. Also, the need for multi-cloud frameworks is growing as cloud computing is maturing and organizations are leveraging cloud services from on-premise private cloud as well as from number of cloud providers. We developed a big data multi cloud framework called MultiStack, which supports multiple data processing frameworks running across multiple clouds. We describe architecture and implementation of MultiStack. We also describe how it can support various scheduling policies including quota-based, locality-based, deadline-aware and cost based scheduling.

Broadly we focused on using information/data from systems and use learning based approaches to optimize the system performance through better scheduling. We studied scheduling problem in the following settings,

- Energy Efficiency without compromising SLAs
- Network awareness
- Extremely heterogenous resource capabilities between mobile device and cloud
- Multi-cloud BigData processing

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Cloud Computing

Cloud Computing has become one of the most successful technology of and has commoditized the access to computing. *Cloud* refers to services running on large cluster. The services provided can be range from operating system abstractions to softwares. *Cloud Computing* has emerged as very successful model of consumption of such services over network. The National Institute of Standards and Technology (NIST) defines cloud computing [45] as,

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"

Although remote access to resources has been around from early internet days, what makes cloud services different is the access model and management of these services. NIST lists following as essential characteristics of cloud computing,

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

The further classification of cloud computing can be done based on the services provided and the deployment model. The service abstraction provided by the cloud provider separates what is managed by cloud provider from what has to be managed by the user.

- Infrastructure as a Service (IaaS) - This model delivers the abstraction of virtualized compute, storage and network resources to the user. This model is easier to adopt for enterprises as it requires minimal changes to existing application to move to cloud.

- Platform as a Service (PaaS) - This is the model where cloud provider manages development and deployment environment, configurable by users. The PaaS model is targeted towards developers.

- Software as a Service (SaaS) - This is the one of the easiest model for end-users and many of the times users don't realize the existence of cloud services in this form.

In this thesis we are mainly focused on provider of Infrastructure as a Service (IaaS). IaaS provides access to virtual resources in an on-demand, pay-per-use basis. Thus IaaS gives completes control over operating systems, deployed softwares, firewall etc. IaaS has allowed affordable elastic recourse access and has become widely successful.

The deployment models specify where these services are hosted. There are four popular models of deployment,

- Private Cloud - Private cloud refers to the cloud services hosted on-premise behind the firewall. It gives complete control over data, but maintenance and cost can become issues. This model suits users with sensitive data, such as financial or healthcare.

- Public Cloud - Public cloud is the model where cloud services are hosted by the provider and accessible via API. The large scale deployment gives advantage of economics of scale and provides maintains free cloud access.

- Hybrid Cloud - This model is leverages the benefits of both public and private cloud. Here part of the services run on-premise and part on public cloud based on various parameters.

1.2 Virtualization and Data center

Virtualization allows running multiple operating systems and applications on a single physical machine. This is done by adding an additional layer of software between virtualized resource and actual hardware. Virtualization software known as hypervisor makes available virtual copies of the resources of the host machine, known as virtual machine (VM) and is responsible for mapping them to the real hardware. Popek and Goldberg defines a virtual machine as *an efficient, isolated duplicate of a real machine*[53]. This makes better usage of hardware and allows consolidation of multiple virtual machines (VMs) on to a single host.

Although Virtual Machine have been popular since the introduction of mainframes [58] [26], the adoption of cloud computing has spawn a hyper-growth in virtual machine usage. One of the prime reasons for wide spread adoption of virtualization is to improve resource utilization by consolidating multiple resources, multiplexing. Data centers are warehouse scale computers and cluster operating systems gives illusion of a single large computer [31]. Although to cloud client it appears that cloud services are being provided by a single entity, the cloud services are implemented by a set of data center containing pool of hardware devices. The efficient management of these resources are essential for data center operators and is even more relevant for cloud service providers and this is where scheduling plays a very important role.

1.3 Scheduling and Cloud Computing

The word scheduling is believed to be originated from a latin word *schedula* around 14th Century, which then meant *papyrus strip*, slip of paper with writing on it. In 15th century, it started to be used as mean *timetable* and from there was adopted to mean scheduler that we currently use in computer science. According to wikipedia, scheduling, is the process of deciding how to allocate resources to a set of processes. The resource arbitration is at the heart of the modern computers.

Scheduling is very important for cloud provider to achieve efficient resource pooling and elasticity. Scheduling problem becomes very relevant in cloud computing scenarios where the cloud service providers have to operate at very much efficient to be competitive and take advantage at scale. The wide acceptance of cloud computing means data centers with many more machines and the usage model is much different than traditional clusters, for example hour boundaries, auction based prices etc. Thus scheduling in cloud data center is more challenging than traditional cluster schedulers. Also, these data center run many different kinds of applications with varying expectations from infrastructure. Resource usage patterns in traditional data centers are have less variance in than the unpredictability faced by cloud data centers. The way operating system scheduler tries to optimize the utilization of resources on a single machine, similarly cloud schedulers tries to optimize the utilization of data center as a whole. It is clear that in such environment, the role of a scheduler becomes very important in achieving high utilization without effecting application performance. As noted by researchers in [16],

> Scheduling, it turns out, comes down to "deciding how to spend money"

1.4 Problem Definition and Scope

Scheduling problem in cloud environment is to allocate virtual machine requests to physical machines (PM). It also deals with the problem of reassigning VMs to PMs, if required by migrating them to other PMs. In mathematical notation, the problem of scheduling is to find a function that maps a set of VMs to a set of PMs.

$$Map\langle VM, PM\rangle = schedule\left(Set\langle VM\rangle,\ Set\langle PM\rangle,\ constraints\right) \tag{1.1}$$

where Set represent a collection of objects and $constraints$ represents challenges that scheduling function must address. In this thesis, we will be pursuing the question of how to come up with function $schedule$ with different constraints. Our focus in thesis is scheduling decisions and not on the optimizing orchestration or migration. Optimizing the orchestration and migration of virtual machines are related, but separate problems. They are required to carry out the actions based on decisions made by the scheduler and can be treated as a black-box. We took this approach and used state of the art migration approaches and reported the effect on our results when relevant.

Our focus in this thesis is on scheduling decisions for various cloud scenarios. Learning based methods try to optimize scheduling objective in various environments. We present scheduling algorithms for SLA-aware consolidation saving energy, network-aware, mobile cloud and multi cloud scheduling. We identify and solve scheduling problem in different settings. Specific contributions of this thesis are

- Dynamic energy and SLA aware virtual machine scheduling algorithm
- Network aware virtual machine placement algorithm

- Scheduling algorithm for offloading decisions of mobile cloud
- MutliStack - a multi cloud big data system including cost-aware scheduling

The overhead of the scheduler is also an important consideration and becomes even more critical at scale. We make sure that the scheduler is scalable and has minimal overhead for all the cases.

1.4.1 Energy and SLA awareness

The success of cloud computing is motivated by the ability to deliver reliable services while operating at very large scale. While the use of commodity hardware at scale allows operators to amortize initial investments, the operational cost (OPEX) of cloud-scale data center infrastructure is a major issue. The energy consumption accounts for a major part of the operational cost of a data center and has environment consequences. The energy consumption of the datacenter for supporting massive cloud services can be more than entire cities. The scheduling decisions for VM to PM mapping to minimize the number of PMs running can save significant amount of energy. This is traditionally seen as a risk to compromise with Service Level Agreements (SLAs), which are performance guarantees by the provider. We present consolidation algorithm for VM consolidation which honors the SLAs.

We use vector based model and present VM allocation methods based on similarity of required and available resources and dissimilarity of resource requirements of VMs. Each machine is represented as a resource vector consisting of cpu, memory, disk and network components. The cosine similarity between these vectors are used to find allocation which minimizes SLA violation. We adjust number of machines dynamically to suit to the workload requirement, migrating VMs from low utilization machine and put them into low power state. The scale up and scale down operations are triggered by high and low utilization patterns. We validated our approach through extensive simulation.

1.4.2 Network awareness

The network can have significant performance impact for applications in cloud. Network performance has been cited as one of the prime concerns for many workloads particularly for HPC processes. Traffic localization between VM has a huge impact for both cloud providers and consumers. Cloud providers face the issue of network scaling as the data center sizes are becoming much larger and traditional network algorithm are showing age. Modern workloads have notably different resource requirements, due to emergence of many new paradigms in computing. The modern scientific and scale-out workloads depend heavily on network performance.

The network aware scheduler can improve the application performance and solve network scaling problems at the same time. Here, the goal of the scheduler's is to improve traffic localization by looking at the traffic exchange patterns between VMs using a heuristics based algorithm for consolidating VMs. We present algorithms to identify VMs with high network traffic among them (VMCluster) and their placement, while minimizing number of migrations by predicting the gain from the migration.We evaluated these algorithms on traces from three different data centers.

1.4.3 Leveraging Heterogeneity

The heterogeneity of the resources pose a significant challenge for the scheduler. We investigate two cases of heterogeneity, mobile cloud and multi cloud.

Mobile cloud represents extreme scenario of resource heterogeneity case. The mobile devices have limited resources but have access to infinite on-demand resources of clouds, due to always connected nature of mobile. This is an interesting opportunity for mobile scheduler as it can run applications efficiently between mobile and cloud. We present learning based algorithm for offloading decisions from mobile device to cloud. We used various application level features combined with resource usage features for estimate the benefits of running application on cloud vs mobile. We evaluated our algorithms for various workloads under varying network conditions.

The availability of resources from multiple public cloud providers along with the growing adoption of private cloud raise the opportunity of leveraging these resources in a dynamic manner. Multi cloud is a paradigm to consume cloud resources from multiple providers. Seamless use of these heterogenous resources requires scheduler to be intelligent to hide variations of clouds and provide a unified interface for the all the resources. The scheduling problem in such a multi cloud environment have additional complexity. We present a cost based algorithm for optimal cluster sizing and coupled with policy based scheduling achieve autoscaling for data processing clusters.

1.5 Organization of the thesis

In this chapter we identified properties present in current cluster schedulers and outlined problems which require attention of researchers. We provided a survey of existing cluster schedulers and highlighted their design points. Our objective was not to prove that one scheduler is better than other but to highlight the choices made by current schedulers and its implications.

Chapter 2 discusses importance of energy awareness for data center schedulers. We present a dynamic and SLA aware scheduler for data centers. We present vector based resource utilization model. Based on this, our allocation algorithm and consolidation algorithm is discussed which is followed by rigorous evaluation based on simulation and report energy conservation and SLA violations.

Chapter 3 shows the requirement of network awareness in scheduling decisions. We present algorithm for identification of virtual machine clusters based on network traffic and a heuristics based solution for re-placement of this virtual machines. We compare our algorithm with other approaches and present in-depth performance analysis based on real world traces from three data centers.

Chapter 4 presents a case of scheduling in mobile cloud scenario. We focus on offloading decision of mobile scheduler. This scenario presents an interesting setting where nodes in the distributed system (mobile device and cloud) have extreme challenges in terms of heterogeneity of resources and communication latency/bandwidth. We present a gain based model for potential benefits of running a process on cloud vs mobile device. We used continuous learning model to predict this gain based on previous interactions of process with user and utilized resources. Based on this model, we present a modified android scheduler and evaluation for variety of applications and network scenarios.

In chapter 5, we present a multi cloud system and show how to autoscale big data frameworks on it. We conclude the thesis in chapter 6 providing directions to take this research forward.

Chapter 2

Dynamic Energy and SLA aware Scheduling of Virtual Machines in Cloud Data Centers

With the advancement of Cloud Computing over the past few years, there has been a massive shift from traditional data centers to cloud enabled data centers. The enterprises with cloud data centers are focusing their attention on energy savings through effective utilization of resources. We propose algorithms which try to minimize the energy consumption in the data center duly maintaining the SLA guarantees. The algorithms try to utilize least number of physical machines in the data center by dynamically rebalancing the physical machines based on their resource utilization. The algorithms also perform an optimal consolidation of virtual machines on a physical machine, minimizing SLA violations. In extensive simulation, our algorithms achieve savings of about 21% in terms of energy consumption and in terms of maintaining the SLAs, it performs 60% better than Single Threshold algorithm.

In this chapter, we describe scheduling algorithms that,

- Consolidate the virtual machines effectively based on the resource usage of the virtual machines.
- Utilize the physical machines to the maximum extent and put low utilized physical machines to standby mode, by intelligently migrating the load on to other physical machines.
- Maintaining the SLA guarantees while effectively saving the power consumed by the data center.

2.1 Introduction

Virtualization is the technology which enables cloud computing by providing intelligent abstraction that hides the complexities of underlying software and hardware. Using this technology, multiple operating system instances called Virtual Machines (VMs) [58] can be executed on a single physical machine without interfering each other. Each virtual machine is installed with its own operating system and acts as an independent machine running its own applications. The abstraction provided by this technology takes care of security, isolation of computation and data across the virtual machines without the knowledge of the user. This gave rise to cloud computing which commercializes the benefits of consolidation of virtual machines by exposing them as utility [17]. The rise of cloud computing has relieved many of the enterprises from a massive effort of managing their own data centers by renting computation resources on-demand from any of the cloud providers. There are many Infrastructure as a Service (IaaS) providers like Amazon, Rackspace, GoGrid etc., who provide computing power in *pay-as-you-go*

model. These providers provide a simple interface for managing virtual machine instances on the cloud through web services. Hence we see more applications being deployed on the cloud framework each day.

The cloud providers consolidate the resource requirements of various customers on to virtual machines across the data center. This inherently does not mean that these data centers are energy efficient due to consolidation. The cloud data center administrators have to follow required policies and scheduling algorithms so as to make their data centers energy efficient. The focus on *Green Cloud Computing* has been increasing day by day due to shortage of energy resources. The *U.S. Environmental Protection Agency* (EPA) data center report [23] mentions that the energy consumed by data centers has doubled in the period of 2000 and 2006 and estimates another two fold increase over the next few years if the servers are not used in an improved operational scenario. The *Server and Energy Efficiency* Report [10] states that more than 15% of the servers are run without being used actively on a daily basis. The *Green Peace International* survey [25] reports that the amount of electricity used by Cloud data centers (Figure 2.1) could be more than the total electricity consumed by a big country like India, in the year 2007. This shows that there is a need to utilize the resources very effectively and in turn save energy.

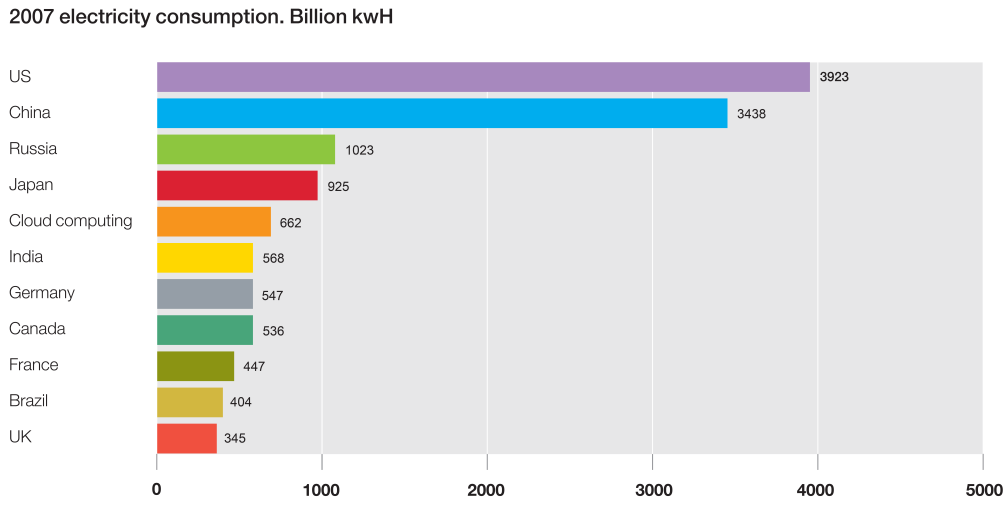

Figure 2.1 Electricity consumption statistics of various countries in the year 2007. Source: Green Peace International [25].

We focus on conserving the energy by effective scheduling and provisioning of virtual machines without compromising on Service-Level Agreement (SLA) guarantees. We present scale-up and scale-down algorithms, which try to consolidate the virtual machines intelligently and reduce the overall energy usage of the data center.

2.2 Related Work

Scheduling has always been a challenging research problem in the field of computer science. Many scheduling algorithms have been proposed each having its own pros and cons.

2.2.1 Round Robin, Greedy and Power Save

Eucalyptus is one of the leading and widely used open source software packages to set up private cloud infrastructure. Round Robin, Greedy and Power Save algorithms are the virtual machine scheduling algorithms provided along with it. *Round Robin* algorithm follows the basic mechanism of allocating the incoming virtual machine requests on to physical machines in a circular fashion. It is simple and starvation-free scheduling algorithm which is used in most of the private cloud infrastructures. The *Greedy* algorithm will allocate the virtual machine to the first physical machine which has enough resources to satisfy the resources requested by it. In *Power Save* algorithm, physical machines are put to sleep when they are not running any virtual machines and are re-awakened when new resources are requested. First, the algorithm tries to allocate virtual machines on the physical machines that are running, followed by machines that are asleep.

These algorithms have limited or no support for making scheduling decisions based on the resource usage statistics. Moreover these algorithms do not take into account of SLA violations, energy consumed etc., which are very important factors in real cloud environments.

2.2.2 Dynamic Round Robin

Ching-Chi Lin et. al in [42] presented an improved version of Round Robin algorithm used in Eucalyptus. According to Dynamic Round Robin algorithm, if a virtual machine has finished its execution and there are still other virtual machines running on the same physical machine, this physical machine will not accept any new virtual machine requests. Such physical machines are referred to as being in 'retirement' state, meaning that after the execution of the remaining virtual machines, this physical machine could be shutdown. And if a physical machine is in the 'retirement' state for a sufficiently long period of time, the currently running virtual machines are forced to migrate on to other physical machines and shutdown after the migration operation is finished. This waiting time threshold is denoted as 'retirement threshold'. So, a physical machine which is in the retirement state beyond this threshold will be forced to migrate its virtual machines and shutdown.

Even this algorithm has limited support for making scheduling decisions based on the resource usage statistics and does not take into account of SLA violations, energy consumed etc.

2.2.3 Single Threshold

In [14], the authors propose Single Threshold algorithm which sorts all the VMs in decreasing order of their current utilization and allocates each VM to a physical machine that provides the least increase of power consumption due to this allocation. The algorithm does optimization of the current allocation of VMs by choosing the VMs to migrate based on CPU utilization threshold of a particular physical machine called 'Single Threshold'. The idea is to place VMs while keeping the total utilization of CPU of the physical machine below this threshold. The reason for limiting CPU usage below the threshold is to avoid SLA violation under a circumstance where there is a sudden increase in CPU utilization of a VM, which could be compensated with the reserve. Single Threshold algorithm works better in terms of energy conservation when compared to Dynamic Round Robin Algorithm discussed in 2.2.2. This algorithm is fairly improved one which takes into consideration of power consumption and CPU usage of physical machines.

2.2.4 Dynamic Voltage Scaling

Dynamic Voltage Scaling (DVS) is a power management technique where under-volting (decreasing the voltage) is done to conserve power and over-volting (increasing the voltage) is done to increase computing performance. This technique of DVS has been employed in [35, 41] to design power-aware scheduling algorithms that minimize the power consumption. Hsu et al. [32] apply a variation of DVS called Dynamic Voltage Frequency Scaling (DVFS) by operating servers at various CPU voltage and frequency levels to reduce overall power consumption.

2.2.5 Dynamic Cluster Reconfiguration

In [52, 22, 44], the authors proposed systems that dynamically turn cluster nodes on - to be able to handle the load on the system efficiently and off - to save power under lower load. The key component of these algorithms is that the algorithm dynamically takes intelligent re-configuration decisions of the cluster based on the load imposed on the system. Our work is mainly inspired by these algorithms which scale the cluster up and down as per the requirement and save power. We tried to employ the same kind of principle in a virtualized data center environment.

In [49], Pérez et al. try to achieve a dynamic reconfiguration using a mathematical formalism, with the use of storage groups for data-based clusters. A considerable amount of research has been done in the fields of load balancing and cluster reconfiguration, with prime focus on harvesting the cycles of idle machines [50, 21, 34]. Our work is based on load balancing and VM migration decisions with prime focus on reducing the total number of running physical machines.

2.3 Proposed Algorithm

Data centers are known to be expensive to operate and they consume huge amounts of electric power [17]. Google's server utilization and energy consumption study [13] reports that the energy efficiency peaks at full utilization and significantly drops as the utilization level decreases (Figure 2.2). Hence, the power consumption at zero utilization is still considerably high (around 50%). Essentially, even an idle server consumes about half its maximum power. Our algorithms try to maintain high utilization of physical machines in the data center so as to utilize energy and resources optimally. In this section, our approach to handle the scheduling decisions of VMs in the data center is presented.

Initially, we assume that all the physical machines in the data center are put to standby mode except for few. We start with only one physical machine that is up and running and only awaken the physical machines as and when required, as directed by our algorithms discussed ahead. When a new request to allocate a VM is received by the data center, the request is directed to *Allocation Algorithm*. The *Allocation Algorithm* takes the decision of allocating the VM on a particular physical machine.

2.3.1 Allocation Algorithm

The Allocation Algorithm presented in the Figure 1, accepts the VM request and tries to fit on to one of the currently running physical machines. The algorithm tries to fit the virtual machine based on the resource usage of the target physical machine. The resource usage of the target physical machine is represented by its *Resource Vector*. Firstly, we discuss *Resource Vector* which forms the base for our algorithms.

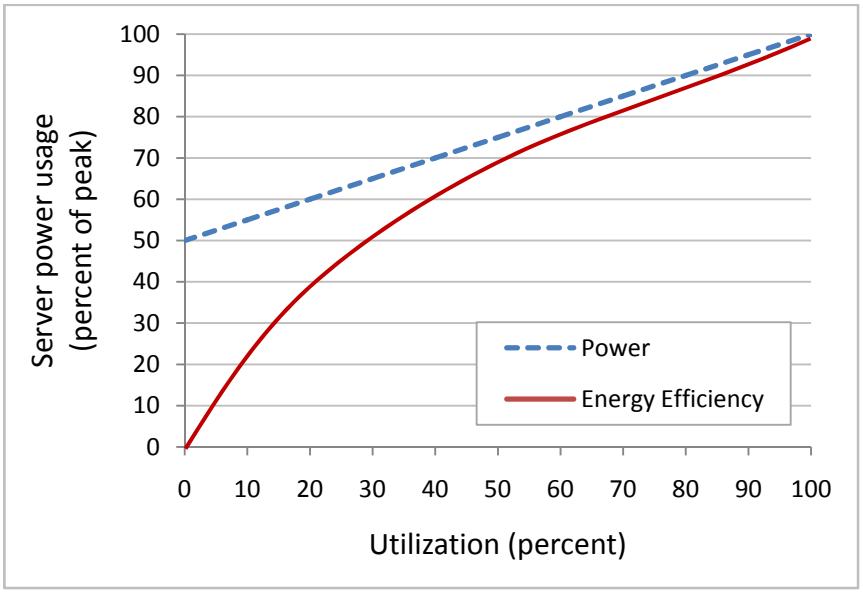

Figure 2.2 Server power usage and energy efficiency at varying utilization levels, from idle to peak performance. Even an energy-efficient server still consumes about half its full power when doing virtually no work. Source: [13].

2.3.1.1 Resource Vector

A virtual machine uses the computing resources based on the applications running on it. Based on the resource usage, a virtual machine can be broadly categorized as CPU-intensive if it uses high CPU, or memory-intensive if it accounts for more of memory IO and similarly disk-intensive or network-intensive. But, just identifying this information about a virtual machine does not give its exact resource usage pattern. To calculate a better resource usage pattern of a virtual machine, we need to take into account of all the resources used by it, at once. So, we define the resource usage pattern of a virtual machine as a vector with four components each denoting CPU, memory, disk and network resources.

$$ResourceVector(RV) =< E_{cpu}, E_{mem}, E_{disk}, E_{bw} > \tag{2.1}$$

where E_x represents the percentage of corresponding resource used i.e. percentage of total CPU, memory, disk and network resources used respectively on that physical machine or Mathematically,

$$E_x = \frac{x\ used\ by\ VM}{max\ x\ capacity\ of\ PMHostingVM} \tag{2.2}$$

Since we denote E_x as percentage of resource used on the physical machine, we represent its value from 0 to 1.

Example: Resource Vector (RV)

Resource Vector 1 = $< 0.70, 0.10, 0.05, 0.05 >$ denotes a CPU-intensive vector.

Resource Vector 2 = $< 0.70, 0.50, 0.05, 0.05 >$ denotes a CPU and memory intensive vector.

2.3.1.2 Construction of Resource Vector

The resources used by a virtual machine are logged at regular intervals at the hypervisor level. Resource Vector (RV) of virtual machine is represented as RV_{vm}. E_x in RV_{vm} of the virtual machine is calculated by averaging its corresponding resource usage (say E_{cpu}) over a period of time Δ (previous Δ time units in the history). For example, E_{cpu} at any time τ is the average percentage utilization of CPU by the virtual machine between $\tau - \Delta$ and τ.

Handling Resource Vector in Heterogeneous Environment: *Resource Vector* of a VM (RV_{vm}) is the vector representation of percentage of resources utilized by the VM on a physical machine. But since the data center could be heterogeneous, this RV_{vm} may not be uniform across different physical machines because of diverse resource capacities. To handle such heterogeneous data center environments, the resource vector could be modified as $RV_{vm}(PM)$, denoting resource vector of a VM on a particular PM.

Example RV in heterogeneous environment: Resource Vector RV of a VM on a physical machine PM_1 is given as follows:

$$RV_{vm}(PM_1) =< E_{cpu}, E_{mem}, E_{disk}, E_{bw} > \tag{2.3}$$

similar to Equation 2.1, where

$$E_{cpu} = \frac{CPU\ used\ by\ VM}{max\ CPU\ capacity\ of\ PM_1} \tag{2.4}$$

Similarly, the rest of the components of $RV_{vm}(PM_1)$, which are E_{mem}, E_{disk}, E_{bw} can be calculated for the PM_1.

So, given the resource vector of a VM on a physical machine say, PM_1 i.e., $RV_{vm}(PM_1)$, we can calculate its resource vector corresponding to another physical machine say, PM_2 denoted by $RV_{vm}(PM_2)$. The calculation is straight forward as the information about resource capacities of both the physical machines is available to the system.

Next, the Allocation algorithm tries to allocate the new VM request on to the physical machine on which it fits the best. To check whether a VM perfectly fits on a running physical machine, we follow *Cosine Similarity model*.

2.3.1.3 Cosine Similarity Model

Cosine similarity gives the measure of the angle between two vectors. If the angle between two vectors is small, then they are said to possess similar alignment. The cosine of two vectors lies between -1 and 1. If the vectors point in the same direction, the cosine between them is 1 and the value decreases and falls to -1 with an increase in angle between them.

Using Euclidean dot product, the cosine of two vectors is defined as

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta \tag{2.5}$$

And the similarity is shown as follows,

$$\textbf{similarity} = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \tag{2.6}$$

The Allocation Algorithm uses this similarity model and tries to allocate the incoming virtual machine request on to a physical machine based on the **similarity** measure between $RV_{vm}(PM)$ of incoming VM and RV of physical machine (denoted by RV_{PM} which will be discussed later).

This idea of similarity is used to allocate dissimilar VMs on a physical machine. By similar/dissimilar VMs, we are referring to the similarity/dissimilarity in resource usage patterns of the VMs. For example, if VM1 is CPU-intensive, we would not want VM2 which is also CPU-intensive, to be allocated on same physical machine since there may be a race condition for CPU resource. By allocating dissimilar VMs on the physical machine, following benefits could be achieved.

1. Race condition for the resources between the VMs could be minimized.

2. The physical machine would be trying to use all the resources, increasing its overall utilization.

Reason for choosing Cosine Similarity model: The reason for choosing Cosine Similarity model over the other similarity models is that, it is simpler and takes into consideration of similarity measure of each component of the vector. And this perfectly suits our requirement of comparing usage patterns of different resources at a time.

Before moving forward, we shall discuss about Resource Vector of a physical machine, RV_{PM}. RV_{PM} is the percentage of resources used on the physical machine. It is similar to $RV_{vm}(PM)$ denoting the percentage of resources used on the physical machine i.e., the usage accounted due to sum of the resources consumed by all the virtual machines running on that particular physical machine. RV_{PM} can be shown as follows,

$$RV_{PM} =< E_{cpu\_used}, E_{mem\_used}, E_{disk\_used}, E_{bw\_used} > \tag{2.7}$$

where E_{x_used} (x is $cpu, mem, disk, bw$) represents the percentage of corresponding resource used i.e. percentage of total CPU, memory, disk and network resources used respectively on the physical machine PM. Similarly, PM_{free} is resource vector which represents the free resources available on physical machine.

2.3.1.4 Calculation of Similarity

As discussed, Allocation Algorithm uses the cosine similarity measure to find a physical machine that is most suitable for the incoming VM request. To use the cosine similarity model, we need to know the RV of the incoming VM. But, since the incoming VM's resource usage may not be known ahead of it's allocation, we make an initial assumption to take a default $RV_{vm}(PM)$. The default $RV_{vm}(PM)$ is assumed to be $< 0.25, 0.25, 0.25, 0.25 >$. Once the VM is allocated and run for a time period of Δ, its exact $RV_{vm}(PM)$ could be found by the mechanism discussed in 2.3.1.2.

To avoid race condition for resources between the VMs, we need to allocate VMs of dissimilar nature. We propose two different methods of calculating similarity measure which are based on Cosine Similarity.

Method 1 - Based on dissimilarity: In this method, we calculate the cosine similarity between RV of the incoming VM and RV_{PM}. And, we select a running physical machine which gives *least* cosine **similarity** measure with the incoming VM. The *least* cosine similarity value implies that the incoming VM is mostly dissimilar to the physical machine in terms of resource usage patterns.

By equation 2.6 we arrive at following formula,

$$\textbf{similarity} = \frac{RV_{vm}(PM) \cdot RV_{PM}}{\|RV_{vm}(PM)\|\|RV_{PM}\|} \tag{2.8}$$

Method 2 - Based on similarity: In this method, we calculate the cosine similarity between RV of the incoming VM and PM_{free}.

$$\textbf{similarity} = \frac{RV_{vm}(PM) \cdot PM_{free}}{\|RV_{vm}(PM)\|\|PM_{free}\|} \tag{2.9}$$

where PM_{free} represents free resource vector of PM, defined by following equation. Individual components of the vector (E_{cpu_free}, E_{mem_free} E_{disk_free} and E_{bw_free}) represents percentage of free resources on the physical machine PM.

$$PM_{free} =< E_{cpu\_free}, E_{mem\_free}, E_{disk\_free}, E_{bw\_free} > \tag{2.10}$$

We select a running physical machine which gives *maximum* cosine **similarity** measure with the incoming VM. The *maximum* cosine similarity value implies that the incoming VM's resource requirements are most compatible with the free resources of physical machine.

The **similarity** value lies between 0 and 1 since we are not dealing with negative physical resource values.

Difference between Method 1 and 2: The similarity methods discussed above help in consolidating VMs on a physical machine without a race condition for resources. There is a subtle difference between the proposed methods. Method 1 tries to allocate VMs which are dissimilar in resource usage patterns. This method helps in achieving the consolidation of VMs with diverse resource usage patterns. While, Method 2 tries to allocate a VM which could properly consume the underutilized resources of the physical machine. This method inherently makes sure that race condition for resources is avoided and at the same time improves the utilization of the physical machine.

Before discussing further algorithms, we present the utilization model for a physical machine upon which the following algorithms are based on.

2.3.1.5 Utilization model

Our work considers multiple resources viz. CPU, memory, disk and network of a physical machine. It is difficult to incorporate utilizations of each of the resources individually into the algorithms. Hence, we come up with a unified model that tries to represent the utilization of all these resources into a single measure, U. The unified utilization measure, U is considered to be a weighted linear combination of utilizations of individual resources. It is given as follows,

$$U = \alpha \times E_{cpu} + \beta \times E_{mem} + \gamma \times E_{disk} + \delta \times E_{bw} \tag{2.11}$$

where, $\alpha, \beta, \gamma, \delta \in [0, 1]$ can be weighed accordingly by the the administrator as per the requirements. And,

$$\alpha + \beta + \gamma + \delta = 1 \tag{2.12}$$

So we try to measure the utilization of any physical machine or virtual machine through a single parameter, U. This unified single parameter, U is introduced for simplicity reasons reducing the complexity of resource modeling while taking multiple parameters into consideration.

The Allocation Algorithm not only tries to consolidate dissimilar VMs but also makes sure that the physical machine is not overloaded after the allocation of VM. Hence, first the similarity measure between the VM and the physical machine is calculated. If the algorithm finds that the similarity measure is good enough to accommodate the VM on the physical machine we proceed to next step. In the next step, the algorithm calculates the estimated U after the VM allocation on the target physical machine (the machine which is suggested by similarity measure) ahead of its actual allocation. The VM allocation is considered only if U after allocation, i.e., the estimated utilization of the physical machine after the allocation of VM on it, is less than the value ($U_{up} - \textit{buffer}$). $buffer$ is a parameter controlling the reserved capacity to handle sudden increase in resource usage.

If U after allocation, is greater than the value ($U_{up} - \textit{buffer}$), we do not consider that physical machine as the allocation may overload it. Instead we take the physical machine which is next best in terms of similarity measure and find its U after allocation. The physical machine is accepted if U after allocation is less than the value ($U_{up} - \textit{buffer}$), else we repeat the same procedure by taking the physical machine with next best similarity measure. The details of this value ($U_{up} - \textit{buffer}$) is discussed clearly later.

Algorithm 1 Allocation Algorithm (VMs to be allocated)

for all VM $\in$ VMs to be allocated **do**
  for all PM $\in$ Running PMs **do**
    {physical machine is represented as PM}
    $similarity_{PM} = calculateSimilarity(RV_{vm}(PM), RV_{PM})$
    {similarity is calculated using any of the two methods discussed}
    add $similarity_{PM}$ to $queue$
  end for

  sort $queue$ in ascending values of $similarity_{PM}$ {if Method 1 is used} *or*
  sort $queue$ in descending values of $similarity_{PM}$ {if Method 2 is used}

  for all $similarity_{PM}$ in $queue$ **do**
    $target_{PM}$ = PM corresponding to $similarity_{PM}$
    if U after allocation on $target$ PM $<$ ($U_{up} - \textit{buffer}$) **then**
      $allocate$(VM, $target$ PM)
      {VM is allocated on $target$ PM}
      return SUCCESS
    end if
  end for

  return FAILURE {VM can't be allocated on any of the running machines}
end for

If the algorithm fails to find any running physical machine which satisfies both the above conditions, then it awakens one of the standby physical machines and allocates the VM on it. The calculation of

estimated U after allocation is straight-forward since we have enough information about the resource vectors of VMs and physical machines.

After the allocation, the resource usage of each physical machines is monitored at regular intervals. And if the utilization of a physical machine reaches an administrator specified threshold (*Scale-up Threshold*, U_{up}), we follow the following Scale-up Algorithm.

2.3.2 Scale-up Algorithm

If the utilization, U of any physical machine is observed to be greater than U_{up} for a consistent time period T, the Scale-up Algorithm is triggered. The Scale-up Algorithm presented in Algorithm 2, then tries to bring down U of the physical machine by migrating the VM with ***highest utilization*** on that particular physical machine to another physical machine. By choosing VM with highest utilization, we try to free up maximum resources from the PM. Firstly, the Scale-up algorithm hands over the VM with high utilization on that overloaded physical machine to Allocation Algorithm for suitable migration. Then, the Allocation Algorithm tries to consolidate that particular VM on any of the other already running physical machines, duly taking into consideration that the migration does not overload the target physical machine as well. If the Allocation Algorithm succeeds in finding a physical machine to allocate the VM, the migration of the VM is instantiated on to the target physical machine. But, if the Allocation Algorithm fails to find a suitable physical machine, then one of the standby physical machines is awakened and migration of the VM is instantiated on to it. By doing this we bring down the U of the physical machine below U_{up}.

Addition of standby physical machines to the running physical machines happens only when required, to handle the rise in resource requirement. This makes sure that the physical machines are used very optimally, conserving a lot of energy.

Algorithm 2 Scale-up Algorithm

1: **if** $U > U_{up}$ **then**
   {if U of a PM is greater than U_{up}}
2: VM = VM with $max\ U$ on that PM
3: Allocation Algorithm(VM)
4: **end if**
5: **if** Allocation Algorithm fails to allocate VM **then**
6: $target$ PM = add a standby machine to running machine
7: $allocate$(VM, $target$ PM)
8: **end if**

Similarly, if the utilization of a physical machine goes down below an administrator specified threshold (*Scale-down Threshold*, U_{down}), we follow the following Scale-down Algorithm.

2.3.3 Scale-down Algorithm

If the utilization, U of any physical machine is observed to be lower than U_{down} for a consistent time period T, the Scale-down Algorithm is triggered. This suggests that the physical machine is under-utilized. So, the Scale-down Algorithm presented in Figure 3, tries to migrate VMs on that particular under-utilized physical machine to other running physical machines and put it to on standby mode. The VMs on the physical machine are handed over to Allocation Algorithm one after the other for allocation

on any other running physical machines, duly taking into consideration that the target physical machines are not overloaded. If the Allocation Algorithm succeeds in finding suitable physical machine where it can consolidate these VMs, the migration of such VMs is initiated on to the target physical machine. The physical machine is then put in standby mode after all the migration operations are performed. But, if the Allocation Algorithm fails to find a suitable physical machine, the VMs are allowed to run on the same physical machine.

Algorithm 3 Scale-down Algorithm

1: **if** $U < U_{down}$ **then**
   {if U of a PM is less than U_{down}}
2: Allocation Algorithm(VMs on PM)
3: **end if**

Reason for using Threshold: Scale-up and Scale-down algorithms are triggered when it is observed that U of a physical machine is above or below U_{up}, U_{down} thresholds for a consistent period of time respectively. These thresholds make sure that the physical machines are neither over-loaded nor under-utilized. Preventing the utilization of physical machine above U_{up} helps in reserving sufficient resources for any sudden surge in utilizations of any of the VMs. This reserve compute resources greatly helps in avoiding any SLA violations. Similarly, usage of U_{down} helps in conserving energy by putting an under-utilized machine to standby mode. To trigger these algorithms, it is a necessary condition that the utilization activity on physical machine should persist consistently for a certain period of time. Sometimes there could be a sudden surge in utilization of a VM and may just persist for small duration. By imposing this condition we could avoid unnecessary migration of VMs during such conditions.

Since these thresholds are percentage utilizations of physical machines, the algorithms work unchanged for heterogeneous data centers.

Reason for using buffer: Before a VM is allocated on a physical machine using Allocation Algorithm, its utilization U after the VM's allocation is calculated upfront. And the allocation of that VM is considered only if U after the allocation is less than (U_{up} - *buffer*). This *buffer* value is considered to make sure that the utilization does not reach U_{up} immediately after allocation, which avoids scale-up operation.

Selection of standby physical machines while scaling up: During the scale-up operation, a standby physical machine may be re-awakened to accommodate VMs. The machine which is least recently used is picked up while selecting a standby physical machine. This makes sure that all the machines in the data center are uniformly used and avoids hot-spots.

Difference between Scale-up and Scale-down Threshold: U_{up} and U_{down} are set by the administrator of the data center as per the requirements. Difference in these values should be made sufficiently large so that the data center does not experience a jitter effect of scaling up and down very frequently.

2.4 Evaluation and Results

The cloud data center architecture is simulated and the results are generated over it. The simulator is written in Java.

2.4.1 Simulation Model

Our simulator simulates the cloud data center from a granularity level of physical machines, virtual machines running on it, to applications running on each virtual machine. Each physical machine could be designed with its own resource specification. Each virtual machine could be assigned to any physical machine dynamically with requested amount of resources. One or many applications could be running on each virtual machine with its own resource requirement dynamics. The simulator has the provision to incorporate scheduling algorithms which guide the allocation of resources in the data center. The simulator takes care of the amount of energy consumed using the model discussed in Google's server utilization and energy consumption study [13]. The simulator is designed with the following SLA model.

SLA Model: An SLA violation is considered at the process scheduling level of hypervisor, whenever any requested resource could not be met to any virtual machine. In simpler terms, during the scheduling of VMs on a physical machine by the hypervisor (scheduling the VMs is a kind of process scheduling in the operating system), a violation of SLA is considered, whenever requested resources such as the amount of CPU, memory, disk or network could not be supplied to any virtual machine.

2.4.2 Experimental Set-up and Dataset

The simulation is performed on a machine with Intel core 2 Duo, 2.4 GHz processor with 2 GB of memory and 500 GB of hard disk which runs Ubuntu 10.04 LTS (Lucid Lynx).

Table 2.1 Simulation and Algorithm Parameters

Parameter	**Value**
Scale-up Threshold, U_{up}	[0.25, 1.0]
Scale-down Threshold, U_{down}	[0.0 to 0.4]
buffer	[0.05 to 0.5]
Similarity Threshold	[0, 1]
Similarity Method	Method 1 or 2
Number of physical machines	100
Specifications of physical machines	Heterogeneous
Time period for which resource usage of VM is logged for exact RV_{vm} calculation, Δ	5 minutes

Rigorous simulation is carried out with various distinctive workloads, based on the real life data center usage, as the input dataset to the simulator. The simulator and algorithm parameters are specified in Table 2.1. To verify the efficacy of our algorithms, we compared them to Single Threshold algorithm and the results are recorded as follows.

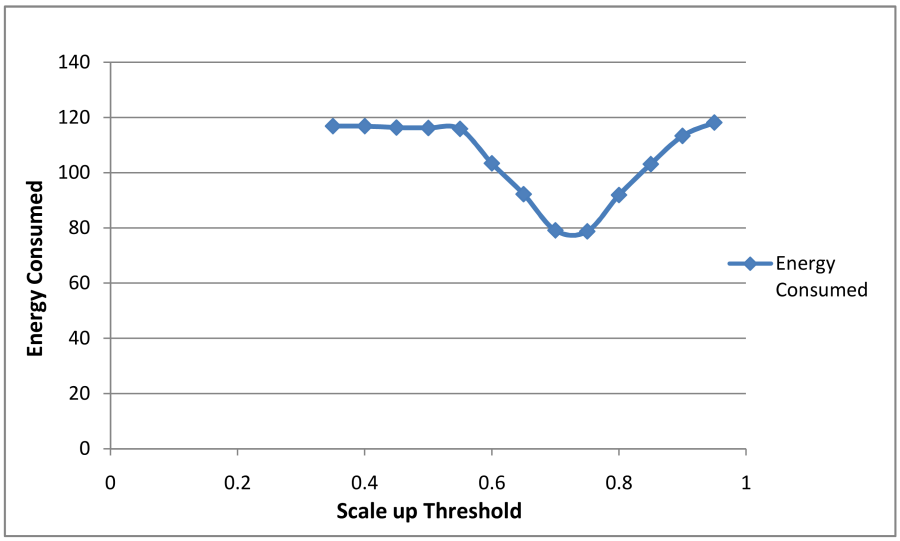

Figure 2.3 The graph demonstrates the effect of Scale up Threshold on energy consumption (in kWh). We see a sudden drop of energy consumption when U_{up} is around 0.70 to 0.80.

2.4.3 Energy Savings

2.4.3.1 Effect of Scale up Threshold

Experiments are carried out on our algorithms to find out the effect on energy consumption for various values of U_{up} and the output is plotted in Figure 2.3. The curve shows a dip when U_{up} is around 0.70 to 0.80 indicating a sudden drop in the energy consumed.

The curve says that the U_{up} should not be too high or too low and its optimal value is around 0.70 to 0.80. If U_{up} is low, Scale-up algorithm tries to run more physical machines to accommodate VMs. And when U_{up} is too high, we see more number of VMs getting consolidated in the machine and few surges in the usage of VMs could lead to running new physical machines. Hence, we see a gradual increase in the energy consumption after 0.80.

2.4.3.2 Effect of scaling down

Figure 2.4 demonstrates the use of having a threshold to put machines to sleep and its effect on energy conservation.

The graph shows that the energy consumed by our algorithms with scale down algorithm enabled, is much lower than the algorithm without scale down procedure. Scaling down of machines when there is not enough load on them could directly save up to 50% of energy as demonstrated in the figure. Higher the value of U_{down}, more the physical machines that are scaled down. At the same time, U_{down} should not be too high, which could result in a jitter effect of scaling up and down, due to a low difference between U_{up} and U_{down}, which was discussed earlier.

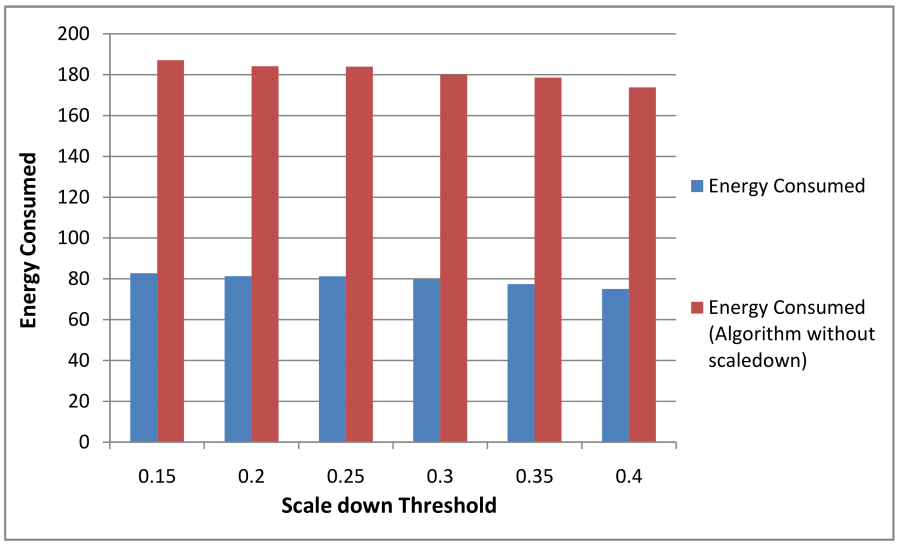

Figure 2.4 The graph demonstrates the effect of Scale down Threshold on energy consumption (in kWh). Algorithm with scale down procedure enabled, performs better in terms of energy conservation.

2.4.4 SLA violations

2.4.4.1 Effect of Similarity Threshold

In Figure 2.5 we try to compare our results with Method 1 and Method 2 similarity measures. The **similarity** value lies between 0 and 1 since we are not dealing with negative resource values. Method 1 works well with zero violations for lower values of *Similarity Threshold* as expected, i.e. for values less than 0.6, since dissimilar VMs are perfectly consolidated with lower threshold values. We observe that Method 2 works even better in terms of SLA violations which shows no SLA violations for any *Similarity Threshold*. This is because Method 2 takes into consideration of available resources in the first place, even before performing consolidation. This proves as an advantage in case of Method 2 and hence Method 2 is better than Method 1 in terms of consolidation.

2.4.4.2 Effect of Scale up Threshold

In Figure 2.6 we try to compare the effect of U_{up} on the number of SLA violations. A very highly stochastic workload is imposed to the simulator to test this experiment. We see that the SLAs are not violated for lower U_{up} values. But as U_{up} increases, more VMs get consolidated on a single physical machine. And when there is a sudden surge in usage of few of the VMs on this machine, there is not enough free resources to handle the immediate requirement, which leads to SLA violations.

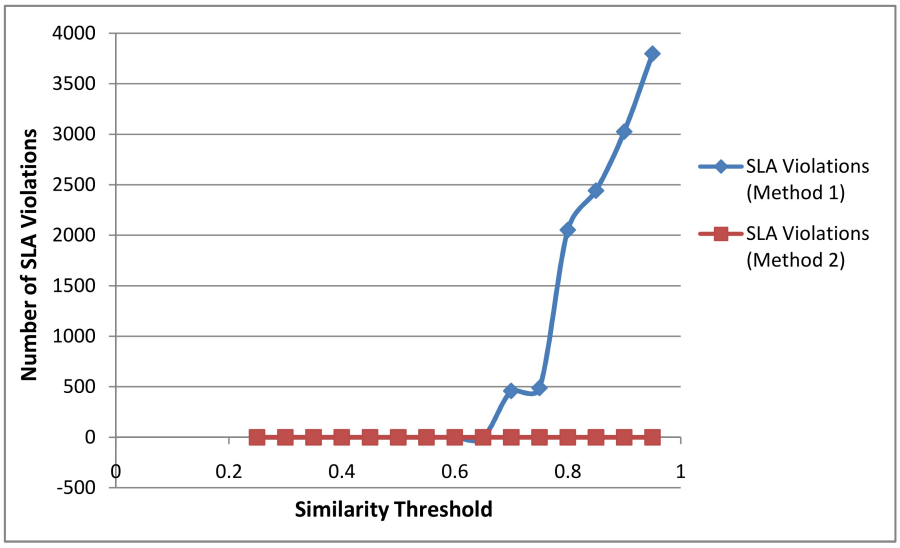

Figure 2.5 The graph demonstrates the effect of Similarity Threshold on number of SLA violations. Method 2 performs very well with zero violations for any Similarity Threshold.

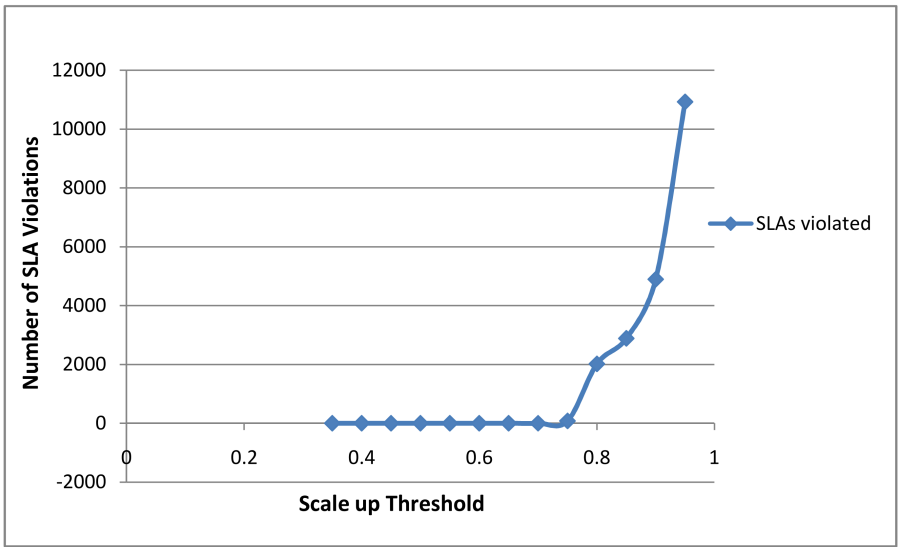

Figure 2.6 The graph demonstrates the effect of Scale up Threshold on number of SLA violations. No violations occur for lower values of *Scale up Threshold* U_{up}.

2.4.5 Effect of buffer

An additional padding called as *buffer* is provided to Allocation Algorithm shown in Figure 1, to avoid SLA violations. The Figure 2.7 shows the advantage of having *buffer* on SLA violations. We see in the curve that as *buffer* increases, the number of SLA violations drop to zero, which is as expected. The *buffer* value has to be used very economically in conjunction with U_{up} and optimal value is around

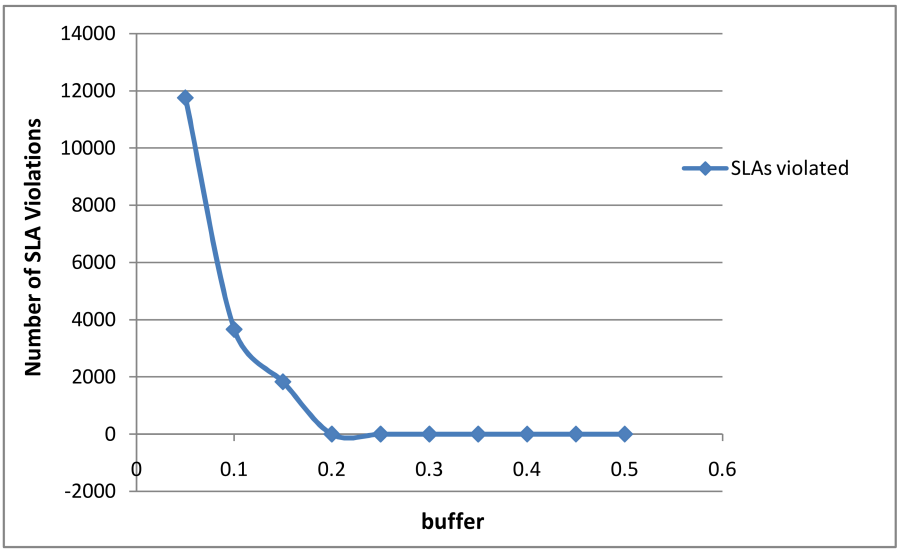

Figure 2.7 The graph demonstrates the effect of buffer on SLA violations. The number of SLA violations drop to zero with a buffer value of more than or equal to 0.2.

0.2. Increase in *buffer* creates more hindrance to consolidation, causing a steady increase in energy consumption which is shown in Figure 2.8.

2.4.6 Effectiveness of our algorithm against Single Threshold Algorithm

We have conducted several experiments with various workloads on both Single Threshold and our algorithm. We have chosen the best configuration for our algorithm, i.e., with U_{up} = 0.75, U_{down} = 0.15, *buffer* = 0.15, Method 2, *Similarity Threshold* = 0.6. And for Single Threshold algorithm a threshold of 0.75 is used. In Figure 2.9, we see a considerable amount of energy savings with our algorithm, saving up to 21% of energy. While in terms of number of SLA violations, our algorithm performs very well maintaining up to 60% more SLA guarantees.

2.5 VM-Network Co-Scheduling for energy savings

The energy consumption in data center is a key concern in sustainable growth of paradigms like cloud computing. As servers are becoming more and more energy efficient, the concerns around network power consumption are increasing. Current networks are designed for peak load and it is well known that they are not power proportional [43]. Power control in network switches is available at two levels - ports and switch. Turning off a port does not yield significant power savings as the power consumption of a switch varies less than 8% when the traffic goes from zero to full capacity. We propose an approach to dynamically decide the number of network devices required to support the current load. We explored scheduling of network devices along with virtual machine scheduling. This leads to much better energy efficiency in data center as network devices. We evaluate our approach and report findings in terms of energy saving and performance of the network.

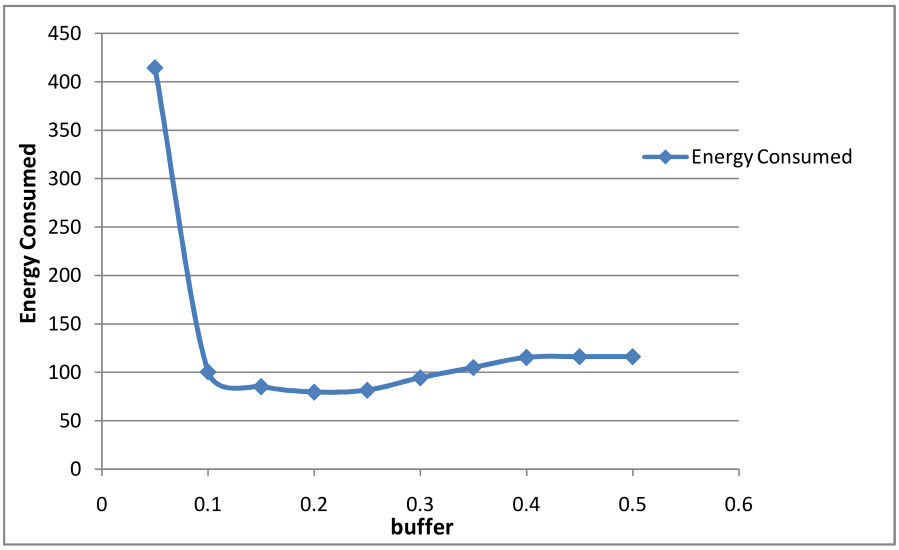

Figure 2.8 The graph demonstrates the effect of buffer on energy consumption (in kWh). We see a sudden drop of energy consumption when buffer is around 0.20, but steadily increases beyond it.

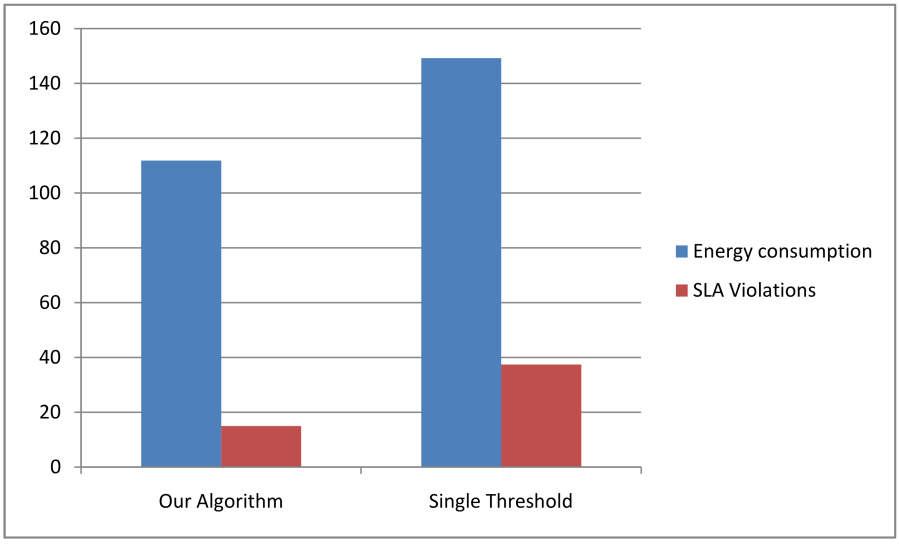

Figure 2.9 The graph demonstrates the effectiveness of our algorithm against Single Threshold algorithm in terms of both energy consumption (in kWh) and also number of SLA violations.

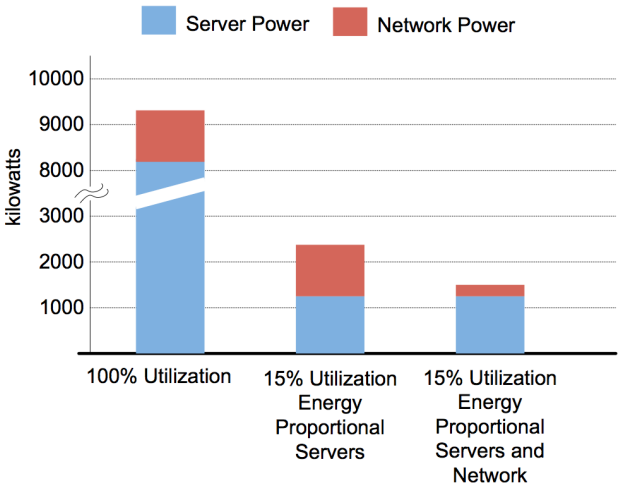

Figure 2.10 The energy consumption by server and network at various utilization

Greenberg et al. [27] show that servers and network components are responsible for nearly 45% and 15% of the amortized cost of a cloud scale data centers, respectively and argue that, innovations are necessary for making data center networks energy efficient. The increasing shift towards green computing has generated a lot of interest in research community towards energy proportional computing. Although there has been significant work around making servers more energy efficient, the research in energy efficient networks deserves more attention.

Abts et al. [11], suggest that, current flattened network topologies are efficient in terms of energy usage and propose an approach for making explicit power performance tradeoff on the modern links. ElasticTree [30] is a network-wide energy manager which, attempts to save energy by shutting down the network elements. It formalizes the problem as an optimization problem and uses the solution to decide the required active devices and links in the system.

We use OpenFlow counters for gathering traffic information about the network and detect the switches that are having minimal traffic. From each of such switches, we try to consolidate the flows to other switches. While consolidating, we reserve 20% flow capacity of switches to take care of sudden surge in traffic. Our approach is independent of the topology and configuration of the network. It is important to note that our approach does not assume the power proportionality of network devices, but it can take advantage of this behavior that is likely to be available in future devices by putting them in sleep mode.

Algorithm 4 OptimizeAllocation($Set\langle s\rangle$ S)

1: Update traffic metrics using SDN counters
2: **for** each Switch s in S such that $Utilization(s) < threshold\ \theta$ over time t **do**
3: **if** $canMigrate$(s, S-s)) **then**
4: pFlows = $prioritizeFlows(s)$
5: $incrementalMigration$ (pFlows)
6: $Poweroff$(s)
7: **end if**
8: **end for**

Table 2.2 Simulation Setup

Parameter	Value
Number of Hosts	2000
Number of Edge Switches	100
Topology	FatTree
Link Capacity	100 MBPS
Switch booting time	90 sec
Number of Ports per Switch	24

We used very basic techniques for $canMigrate$, $prioritizeFlows$ and $incrementalMigration$. Further investigation on how to optimize the flow classification/ ordering and incremental migration of flows, can lead to significant saving in network poet consumption.

We evaluated our algorithm using mininet simulator [40] with simulation parameters of a typical data center running commodity servers and network gears, shown in Table 2.2. We used Floodlight (v0.85) as our openflow controller [5], hosted on a 2 GHz, 4 GB RAM machine with Ubuntu 12.04. For each machine we generated packets to randomly chosen other hosts for required number of flows using Iperf. Table 2.2 describes our experimental setup.

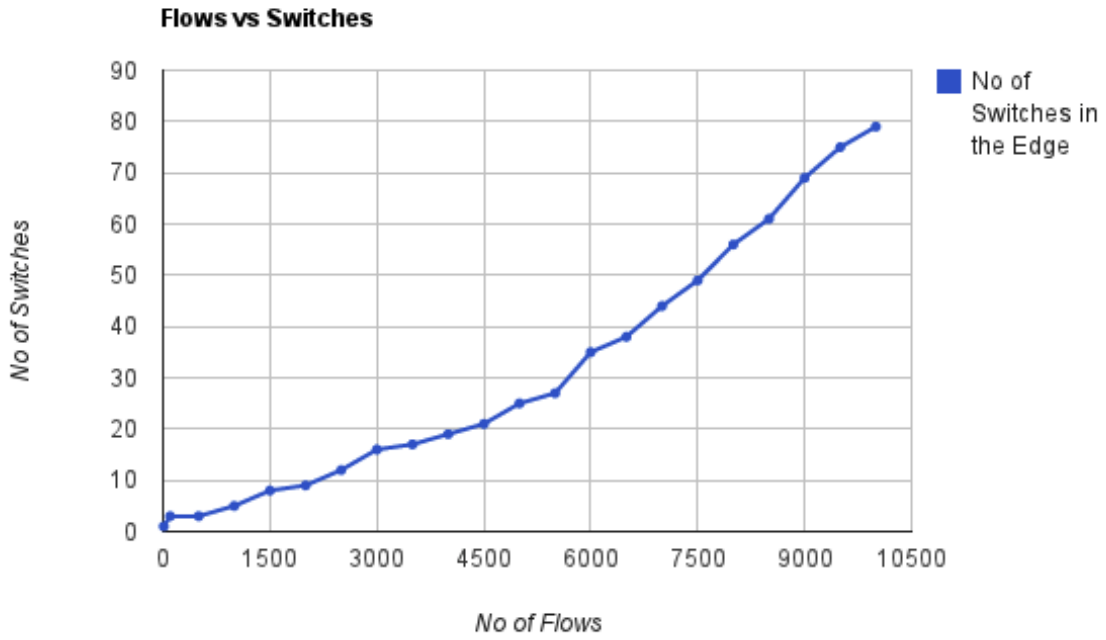

Figure 2.11 Number of active switches as the number of Flows grows almost linearly

Figure 2.11 shows the effectiveness of the algorithm in consolidating the flows on switches. When the number of flows is relatively small, the number of switches required is sub-linear in flows. As the

number of flows increase, the algorithm requires a little more than a linear increase in switches. This is expected as with large number of flows the consolidation becomes more and more complex.

We measured the effect of the consolidation on the performance of the network through maximum delay experienced by the flow packets with consolidation and the results are shown in Figure 2.5. The algorithm managed to keep the delay variance within 30 μs and the average delay being 99 μs. These results are encouraging because, in normal case without consolidation, the delay variance was 22 μs and the average delay was 84 μs without any consolidation being applied. This shows effectiveness of our approach towards the goal of saving energy while minimizing the performance effects on the network due to consolidation. All the results are reported over 15 experiments. The number of switches in the higher layer is not shown in the figure.

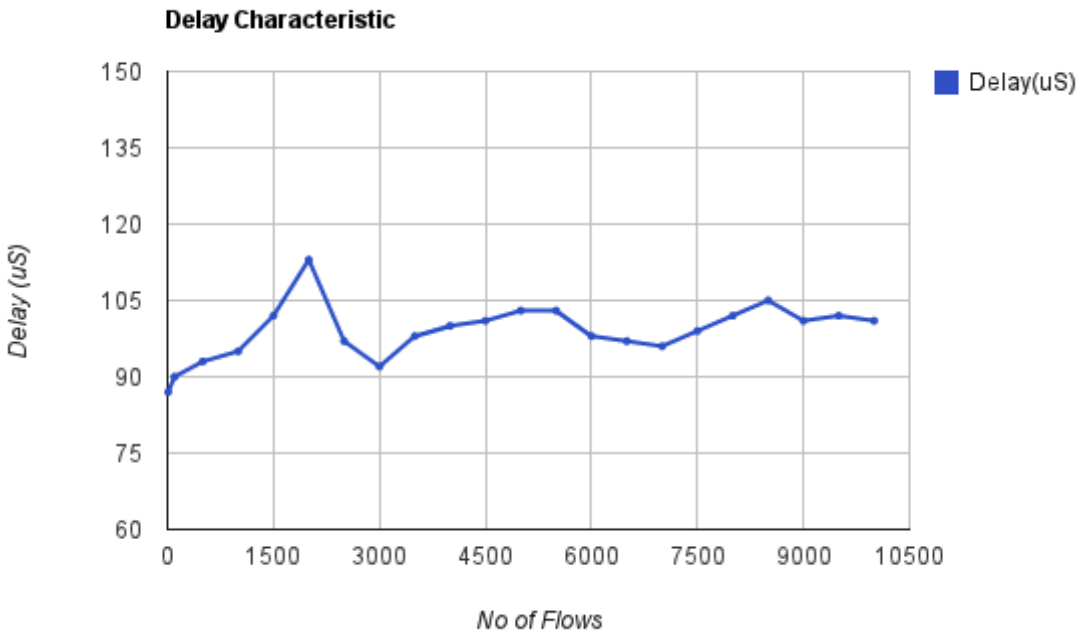

Figure 2.12 The variance in delay as number of flows grows

Without the use of consolidation, all the network switches have to be kept switched on all the time resulting in substantial waste of energy by idle switches, which can be avoided using our approach.

2.6 Conclusion

In this chapter, we described algorithms that try to conserve energy in cloud data centers. We discussed the Allocation Algorithm which tries to consolidate the virtual machines on physical machines taking into consideration of resource usage characteristics. The similarity model we discussed tries to avoid SLA violations and allocates virtual machines accordingly. The Scale-up and Scale-down algorithms keep track of resource usage of each physical machine and dynamically rebalance the data center based on the utilization. We have successfully evaluated our algorithms against Single Threshold algorithm and they show a considerable amount of energy savings. We also presented vm-network co-scheduling algorithm and showed approximately linear power consumption behavior with traffic load on the network maintaining the delay characteristics.

Chapter 3

Network Aware Placement

Resource management in modern data centers has become a challenging task due to the tremendous growth of data centers. In large virtual data centers, performance of applications is highly dependent on the communication bandwidth available among virtual machines. Traditional algorithms either do not consider network I/O details of the applications or are computationally intensive. We address the problem of identifying the virtual machine clusters based on the network traffic and placing them intelligently in order to improve the application performance and optimize the network usage in large data center. We propose a greedy consolidation algorithm that ensures the number of migrations is small and the placement decisions are fast, which makes it practical for large data centers. We evaluated our approach on real world traces from private and academic data centers, using simulation and compared the existing algorithms on various parameters like scheduling time, performance improvement and number of migrations. We observed a $\sim$70% savings of the interconnect bandwidth and overall $\sim$60% improvements in the applications performances. Also, these improvements were produced within a fraction of scheduling time and number of migrations.

In this chapter, we describe a following algorithm whose primary objective is to consolidate VMs using network awareness.

- VMCluster formation algorithm : to cluster the VMs, based on their traffic exchange patterns.
- VMCluster placement algorithm : consolidation algorithm for placing the VMClusters such that, the application performance increases and the internal datacenter traffic is localized as much as possible.

3.1 Introduction

Modern data centers are extremely large, complex and host applications with a wide range of communication requirements that make resource management a challenging task. In a cloud environment where pay-as-you-go model and on-demand computing are encouraged, VMs from the same applications can be placed across data center and network performance among VMs cannot be guaranteed. This can adversely affect performance of applications, particularly with large east-west traffic and projects that aim to build cluster computing systems on cloud. In large virtual dynamic environment where clusters are scaled up and down frequently, performance of such modern distributed application can face performance issues that are acknowledged by studies [64], showing wide variations in available bandwidth between VMs.

Using current placement algorithms, which do not take application network requirements and dynamism into account, VMs from the same cluster/application can be deployed in a non-optimal way and network can be a serious performance bottleneck. The improper VM placement not only affects I/O intensive applications but also to other non-I/O intensive applications that will see large I/O waiting time due to shared interconnect and oversubscription in network equipment common in the data centers. This network unaware placement also results into the internal network bandwidth being wasted. Thus internal network saturation not only harms the data-intensive applications but also all the other applications running in the data center.

We use network traffic history to group VMs in virtual clusters and use greedy algorithm to consolidate VMs so as to localize traffic from the same group, which results in better application performance and saving in internal bandwidth. The pervious approaches have either been too computationally intensive or application specific or have considered only CPU or suggest too many migrations, limiting their applicability in practice.

3.2 Related Work

Resource allocation and management in data centers has received a lot of attention in recent years due to the emergence of cloud computing. This has resulted in a significant research effort from the community and much advancements in this area.

Researchers have realized the limitations of current data center network architectures and have spawned parallel efforts to redesign data center architectures and topologies to accommodate requirements of the modern data center traffic. PortLand [47] and VL2 [28] are three-tier architectures with Clos [19] topologies. BCube [29] is a multi-level architecture in which the servers forward traffic on behalf of other servers and they are considered to be first class citizens of the network architecture. We approach the problem from the other end, making better use of the available interconnect bandwidth. Our goal is to come up with the VM placement algorithm that places the high-communicating VMs near each other, to help scaling the interconnect.

Most of the proposed algorithms for VM placement and consolidation focus on computation resource utilization. Considering the computing resource as utility, the VM placement problem can be modeled as various constraint satisfaction problems [48] [61]. Kusic et al.[38] have used Limited Lookahead Control (LLC) to solve the continuous consolidation problem as a sequential optimization. They require application specific learning. The complexity of the proposed model is high and can take even 30 minutes to make scheduling decision for even a very small number of nodes.

A wide range of heuristics based algorithms has been proposed for VM placement. Beloglazov et al. [14] proposed heuristic based algorithms for VM consolidation to minimize the energy utilization. Tang et al. proposed a task scheduling algorithm which is thermal and energy-aware [62]. Affinity-aware migration was proposed in [59] to minimize the communication overhead. They use a bartering based algorithm to negotiate VM relocations based on the local information. This algorithm can take longer to execute because of the negotiation. Use of local information can result in non-optimal placement of VMs. In [33] the authors consider demand, availability and communication constraints and transformed the VM placement problem into a graph coloring problem. They also show that each of these constrained subproblems are NP-hard.

There are some prior approaches to the VM placement problem, which model it as some form of optimization problem [46][33]. Tantawi et al. [63] consider the problem of placing virtual machine clusters on to physical machines in data center. They use importance sampling based approach to solve

the placement problem. They note that the implementation of this approach is inefficient and relies on a sampling process to incorporate communication needs and other constraints of requests within the placement algorithm. In [65] authors consider the problem of virtual machine placement in virtualized environments with multiple objectives. They use a genetic algorithm with fuzzy evaluation to search the solution space and combine the constraints. While the above approaches can accommodate the network requirements, they are computationally intensive and difficult to scale for real world data centers. Also, they do not consider the performance benefits of a consolidation and can lead to excessive migration for very little performance improvement that can degrade overall performance.

There has been some work to localize the network traffic using application-awareness. While these approaches are useful in specialized applications such as Hadoop, their applicability is limited to particular frameworks. Also if these nodes are running in virtualized environments, which is increasingly the case, the framework will not be effective in localizing the traffic because of no visibility into virtual infrastructure. Modern data centers run a variety of applications with a variety of network requirements and it may not be possible for the provider to know the traffic patterns beforehand. Our work does not assume any application-awareness and thus is applicable to all data center workloads. Also, as these frameworks have very specialized traffic patterns, our algorithm will benefit them automatically.

To summarize, previous work has focused on optimizing resources like CPU or energy or thermal awareness. The proposed approaches for network-aware consolidation has limited applicability in real data centers as they require intensive computations and do not consider the benefits against the migrations.

3.3 System Model

We consider typical virtualized data center model consisting of a set of physical machines (PMs) connected through hierarchical network topology. Applications can have several tasks, distributed across many VMs and can scale up and down resulting in dynamic environment where VMs are dynamically spawned and terminated and their traffic pattern changes over the time.

3.3.1 VMCluster

We introduce the concept of VMCluster to represent a group of VMs that has large communication cost over time period T. Our approach does not require knowledge about application running inside VM and uses metrics readily available in hypervisor and network devices. This is important for large data center operators such as public cloud providers, which has no knowledge about customer applications running inside VMs and their communication patterns. Membership of VMs to VMClusters is dynamic and only property that needs to hold is data exchange between the VMs for T time period.

3.3.2 Communication Cost Model

For identifying the VMClusters , we define the following as our communication cost,

$$c_{ij} = AccessRate_{ij} \times Delay_{ij} \tag{3.1}$$

Where $AccessRate_{ij}$ is rate of data exchange between VM_i and VM_j and $Delay_{ij}$ is the communication delay between them. We measured $Delay$ as ping delay between corresponding VMs. c_{ij} is maintained over time period T in moving window fashion and mean is taken as the value.

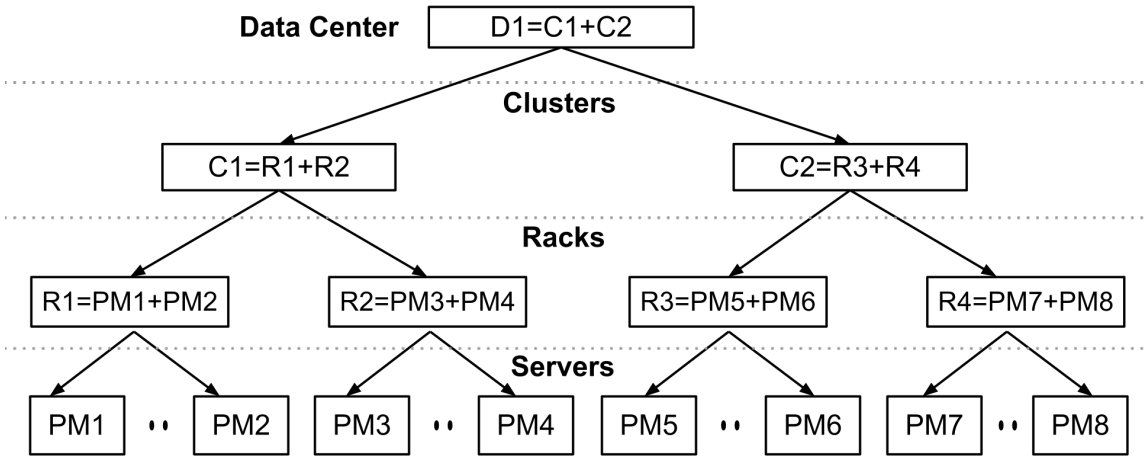

Figure 3.1 Example Cost Tree for a data center with 2 clusters each containing 2 racks

The intuition behind defining cost as Eq.3.1 is to ensure that it captures not only network delay but also how frequent data communication is between VMs. This is important, as we do not want to optimize placement of VMs which are very far but exchange very little traffic and similarly for VMs which have a very small delay, placement optimization of which will not result in significant performance benefits.

3.3.3 Cost Tree

We model data center as hierarchical tree structure called *cost tree*. The root node represents the data center and leaves represent physical servers (PM). The levels below represent clusters, server racks, etc., in superset relation as shown in Figure 3.1. The value of a node represents communication cost of the traffic flowing through the node. The value at parent node is sum of values of its child nodes. The leaf nodes have values representing the access cost to that server. Note that tree representation is logical and not tied to any specific network topology.

We note that with the advent of the Software Defined Network (SDN) based networking in large data centers, the collection of network information is possible providing a global view of network. Note that SDN provides one way of collecting network information and we do not require a data center to be operating SDN. We only require traffic information to be available.

3.4 Proposed Algorithms

We propose algorithms to form the VMClusters, selection of VM from VMCluster for migration and placement algorithm for consolidating using cost tree.

3.4.1 VMCluster Formulation

Traffic information between VMs is maintained in the form of $AccessMatrix$,

$$AccessMatrix_{n\times n} = \begin{bmatrix} 0 & c_{12} & c_{13} & \cdots & c_{1n} \\ c_{21} & 0 & c_{23} & \cdots & c_{2n} \\ c_{31} & c_{32} & 0 & \cdots & c_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \cdots & 0 \end{bmatrix} \tag{3.2}$$

Each element c_{ij} in the access matrix represents communication cost between VM_i and VM_j. It may seem that $AccessMatrix$ size $(n \times n)$ is prohibitively large, which is not the case. Note that we only collect c_{ij} for VMs which exchange traffic between them, making $AccessMatrix$ extremely sparse and disjoint and it is maintained in Compressed Sparse Row (CSR) format which makes it very practical. Further optimization is possible by maintaining only top-K candidates. We assume communication cost within VM is negligible and costs are symmetric ($\forall i, j : c_{ij} = c_{ji}$). The following is the VMCluster formulation algorithm, used to identify virtual clusters.

Algorithm 5 VMCluster Formation Algorithm

```
for each row A_i ∈ AccessMatrix do
    if maxElement(A_i) > (1 + opt_threshold) * avg_comm_cost then
        form a new VMCluster from non-zero elements of A_i
    end if
end for
```

Each row in $AccessMatrix$ represents communication cost incurred by the VM. The algorithm considers maximum of theses costs (max element of the row) and compares it with avg_comm_cost. avg_comm_cost is average communication cost (calculated as average of $AccessMatrix$ elements c_{ij}) of data center that we use as baseline to decide which communication costs are considered significant. $opt\_threshold \in [0, 1]$, is a parameter which controls the number of VMClusters to be considered by the algorithm. If the $opt\_threshold$ is near zero, all the VMClusters with above average communication cost will be considered and as the value grows the number of VMClusters will be decreasing.

3.4.2 Consolidation Algorithm

The consolidation algorithm is responsible for placement of VMClusters to localize the traffic. There are two main steps of the consolidation algorithm.

VM Selection Choosing which VM to migrate is very important as it can affect the optimization achieved and the migration time. We want to choose the VM in such a way that the chosen VM has large communication cost to the rest of the VMs or in other words communication cost-wise farthest from the rest of the VMs in its cluster. The average distance from each VM to all other VMs in VMCluster is calculated. The VM with largest average distance is chosen which ensures that largest communication cost optimized at each step.

$$VMtoMigrate = \arg\max_{VM_i} \sum_{j=1}^{|VMCluster|} c_{ij} \tag{3.3}$$

Placement Algorithm The placement decision for migration destination is hierarchical using cost tree. Each level in the cost tree represents a decision about the placement. While selecting destination

for the VM, the cost tree is traversed bottom-up candidates for the migration destination using following recursive definition ($CandidateSet$ for leaf level is $\emptyset$) are tried at each level,

$$CandidateSet_i(VMCluster_j) = \{c \mid \text{where } c \text{ and } VMCluster_j \text{ has a common ancestor at level } i \text{ in cost-tree}\} - CandidateSet_{i+1}(VMCluster_j) \quad (3.4)$$

Algorithm 6 The Consolidation Algorithm

```
1: for VMCluster_j ∈ VMClusters do
2:    VMtoMigrate according to Eq. 3.3
3:    for i from leaf to root do
4:       CandidateSet_i(VMCluster_j − VMtoMigrate) according to eq. 3.4
5:       for PM ∈ candidateSet_i do
6:          if UtilAfterMigration(PM,VMtoMigrate) <overload_threshold AND
            PerfGain(PM,VMtoMigrate) > significance_threshold then
7:             migrate VM to PM
8:             continue to next VMCluster
9:          end if
10:      end for
11:   end for
12: end for
```

For example, in Figure 3.1, if we have a $VMCluster = \{PM3, PM4, PM6, PM8\}$ then the $CandidateSets$ will be

$CandidateSet_3(VMCluster) = \{PM5, PM7\}$

$CandidateSet_2(VMCluster) = \{PM1, PM3\}$

Here the members of the $CandidateSet_3$ will be tried first. If $PM5$ and $PM7$ do not have sufficient resources for the incoming VM, members of $CandidateSet_2$ are tried next, and these will be continued towards root.

Before choosing migration destination, we consider resource utilization on candidate PM after migration. Function $UtilAfterMigration$ is used to determine if the candidate PM has sufficient resources (compute and memory) available by calculating utilization of PM if the VM is migrated. The resource availability decision of PM is controlled by $overload\_threshold$. If PM is estimated to have required resources, it is chosen as migration destination. Otherwise other candidates in the $CandidateSet$ are tried. If VM cannot be allocated to any of the PM in $CandidateSet_i$, PMs in $CandidateSet_{i-1}$ is tried. This continues recursively towards root. Candidates within same $CandidateSet$ are considered in the order of their cost, the higher cost node is tried first. The algorithm also estimates the benefit that will be achieved by migrating VM to suggested PM before actual migration, in terms of percentage improvement in communication cost after migration as follows,

$$PerfGain = \sum_{j=1}^{|VMCluster|} \frac{c_{ij} - c'_{ij}}{c_{ij}} \quad (3.5)$$

Where c_{ij} is the original communication cost and c'_{ij} is the cost calculated after migration. c'_{ij} calculation is done using equation 3.1, but with the delay updated to reflect the migration. $significance\_threshold \in$

$[0, 1]$ controls the aggressiveness of the algorithm. If $PerformanceImprovement$ is above administrator defined $significance\_threshold$, then the migration is carried out, otherwise no migration is performed In both the cases, next VMCluster is tried to for performance improvement.

3.5 Experimental Evaluation

In this section we provide experimental results to illustrate performance of our algorithm.We compared our approach to traditional placement approaches like Vespa[61] and previous network-aware algorithm like Piao's approach[51]. We extended NetworkCloudSim[24] to support SDN functionalities and used Floodlight [5] as our SDN controller. Counters available in Floodlight were used to collect the required network information and T is set to 3s. The server properties were not dictated in the trace and we assumed them to have typical data center server specification (HP ProLiant ML110 G5 (1 x [Xeon 3075 2660 MHz, 2 cores]), 4GB) connected through 1G using HP ProCurve switches in three-tier. We used traces from three real world data centers, two from universities (uni1, uni2) and one from private data center (prv1), to simulate the traffic to evaluate our approach [15]. The statistics of the traces used for simulation are shown in Table 3.1.

Property	Uni1	Uni2	Prv1
Number of Short non-I/O-intensive jobs	513	3637	3152
Number of Short I/O-intensive jobs	223	1834	1798
Number of Medium non-I/O-intensive jobs	135	628	173
Number of Medium I/O-intensive jobs	186	864	231
Number of Long non-I/O-intensive jobs	112	319	59
Number of Long I/O-intensive jobs	160	418	358
Number of Servers	500	1093	1088
Number of Devices	22	36	96
Over Subscription	2:1	47:1	8:3
Traffic report delay	10 μs		
Trace duration	12 hours over multiple days		

Table 3.1 Trace Statistics

Appendix 3.6 shows the summary of the results achieved across all the three traces. The discussion on various properties follows.

3.5.1 Performance Improvement

We have categorized the jobs into 6 different types based on their I/O requirement and running time for finer evaluation of performance on various classes of jobs. We measured percentage improvement in runtime of jobs of various classes of jobs. As shown in Figure 3.2, I/O intensive jobs are benefited the most but short jobs also have significant improvements. It is important to note that although short jobs have generally low n/w requirements, number of such jobs are typically much larger, making them important for overall improvement of performance.

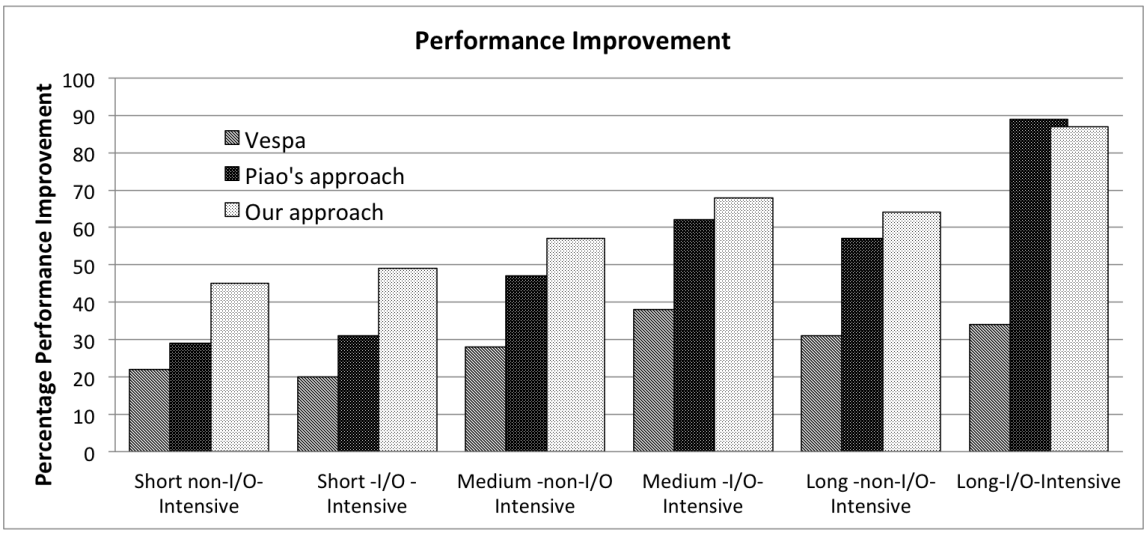

Figure 3.2 Performance Improvement for various class of traffics

3.5.2 Traffic Localization

The goal of consolidation algorithm is to localize traffic for better interconnect scaling and improving application performance. The traffic localization can be measured by the traffic reduction in the higher-level network switches and increase in the lower-level network switches. The effect of consolidation on the core and Top of Rack (ToR) switch traffics in all the tree trace scenarios is depicted in the figure 3.3. In the figure, the traffic after consolidation is shown for switches are compared for our approach and Piao's approach (Vespa isn't n/w aware). As seen in graph, $\sim 60\%$ increase in ToR traffic and $\sim 70\%$ reduction in core traffic confirms effective localization using our approach, while Piao's approach resulted in nearly $\sim 30\%$ increase and $\sim 37\%$ decrease correspondingly. This result also confirms that the network aware placement can mitigate network scalability issues.

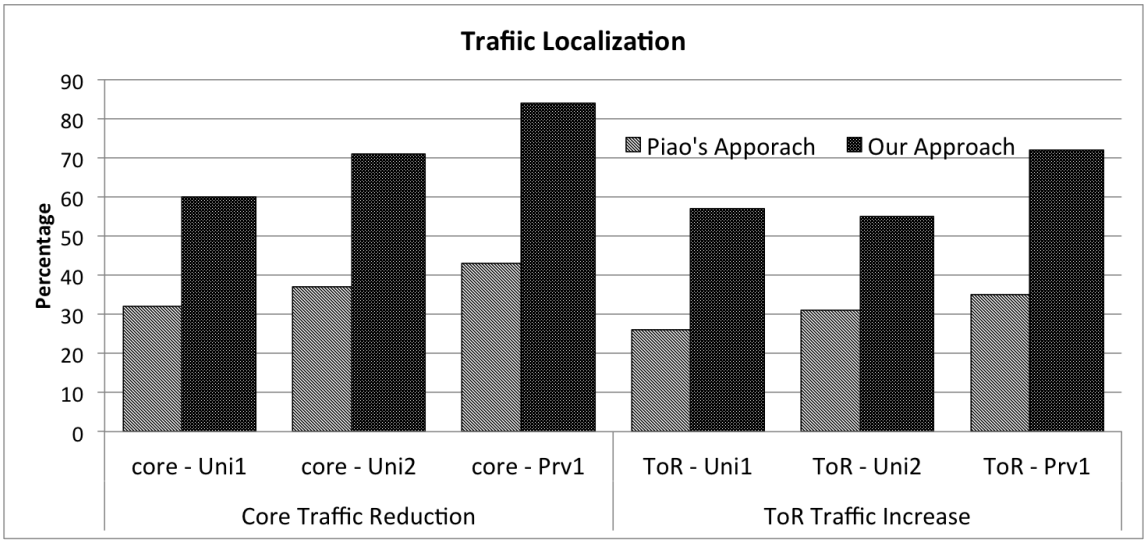

Figure 3.3 Localization for core and ToR traffic

3.5.3 Complexity

Complexity of scheduling is an important measure in practical settings. We report decision making time, stability (variance) and number of migration for all the three approaches on various measures in Table 3.2. Our algorithm performed significantly better than both of the approaches. Also the worst-case time and variance in scheduling are important factors for building practical systems and affect interdependent components. Our approach took 558 ms to in worst case and only 60 ms variance across traces that shows the stability of our approach. Vespa on the other hand, took 973 ms in worst case, with 130 ms variance. Piao's approach is clearly showing the effect of not keeping time complexity in mind and in worst case took more than 1.3 secs.

Measure	Trace	Vespa	Piao's approach	Our approach
Avg. scheduling Time (ms)	Uni1	504.64	677.66	217.36
	Uni2	784.24	1197.54	376.84
	Prv1	718.33	1076.33	324.66
Worst-case scheduling Time (ms)	Uni1	846	1087	502
	Uni2	973	1316	558
	Prv1	894	1278	539
Variance in scheduling Time	Uni1	179.64	146.76	70.68
	Uni2	234.24	246.64	98.74
	Prv1	214.33	216.66	89.99
Number of Migrations	Uni1	154	213	56
	Uni2	547	1145	441
	Prv1	423	597	96

Table 3.2 Complexity – Time, Variance and Migrations

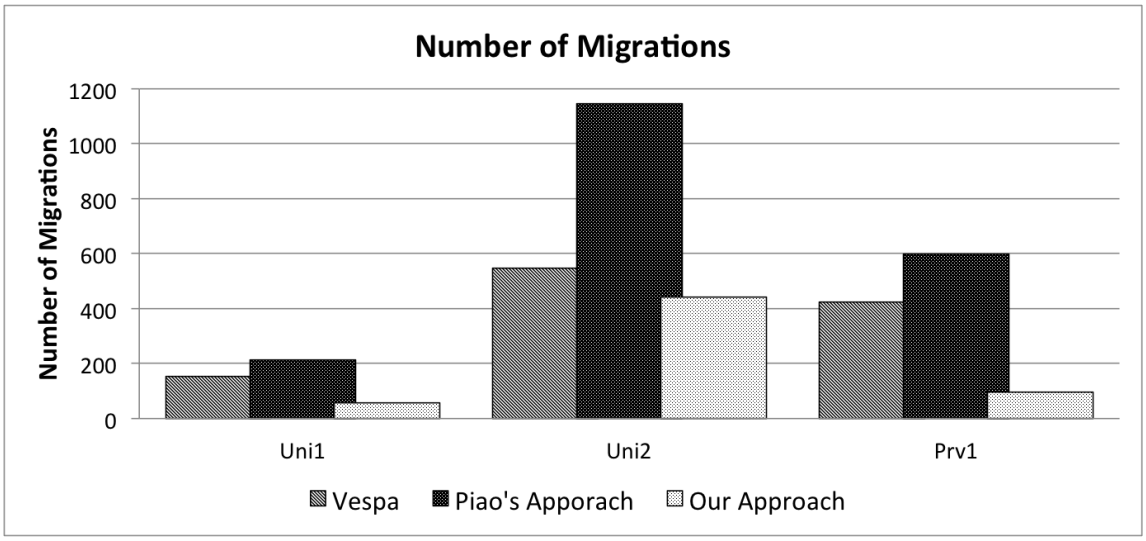

Figure 3.4 Number of Migrations by different approaches

Figure 3.4 compares number of migrations and variance as it can affect application performance and overall stability. Small variance in time shows the predictable behavior of our algorithm compared to other two algorithms. The lack of consideration for number of migration is clearly visible in both other algorithms and our approach required much lesser number of migrations. This also confirms that large number of migrations is not required for significant performance improvement, opposite to what is suggested by most optimization based approaches.

3.5.4 Dependence on Parameters

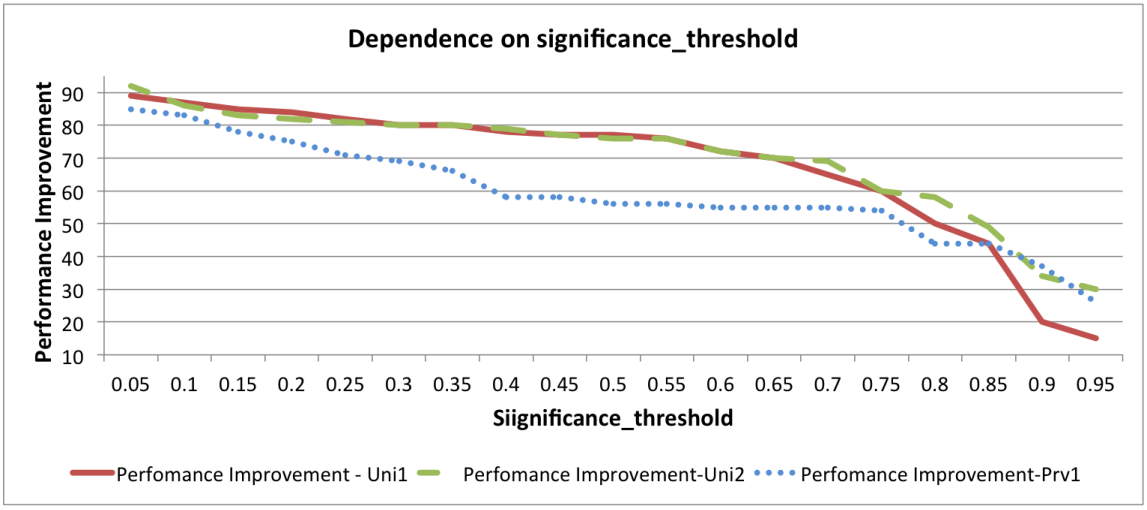

Figure 3.5 Dependence of performance improvement on $significance\_threshold$

Figure 3.5 shows how the performance improvement depends on the parameter $significance\_threshold$. As noted earlier the parameter controls the aggressiveness of the algorithm. As seen in the graph, the initial slow decrease in the performance (10% decrease when $significance\_threshold$ varies from 0.05 to 0.40 in both university traces and 20% in the private trace) indicates that the benefits of the algorithm saturates around 0.60–0.70. Also much higher values of $significance\_threshold$ will force the algorithm to consider most of the possible improvements as insignificant and very little performance improvements can be realized.

The number of VMClusters to be considered by the algorithm is controlled by $opt\_threshold$. Figure 3.6 shows how the variation in number of clusters formed with various values of $opt\_threshold$ for all three traces. The step-wise decrement in VMClusters shows that performance of our algorithm is stable against small variations in the $opt\_threshold$, but number of VMClusters decrease with increase in $opt\_threshold$. Number of VMClusters starts saturating around value 0.6-0.7 and further increase in $opt\_threshold$ does not affect VMClusters due to the inherent traffic patterns of the workload.

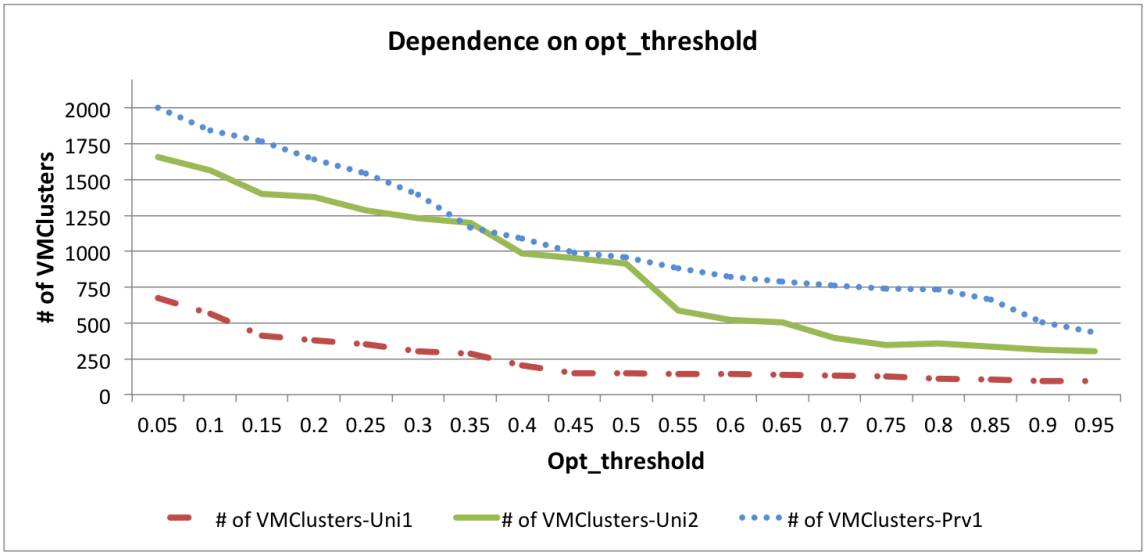

Figure 3.6 Dependence of number of VMClusters on $opt\_threshold$

From both of the graphs, it is clear that the algorithm is stable against small variation in the parameters, but allows the operator making the tradeoffs between performance improvements and the complexity of the placement.

Table 3.6 summaries the performance improvements achieved by all three algorithms.

3.6 Conclusion

In this chapter, we described algorithms to identify virtual clusters and consolidating them in order to improve application performance and localize traffic. We analyzed performance improvement, number of migrations and scheduling time of our algorithm and existing approaches on real world traces, with variety of jobs. Our scheduling decisions achieved better performance and were an order of magnitude faster and also required a fraction of migrations compared to others, making it practical for large data

centers. Also we believe that the identified VMClusters using VMFormation algorithm can be useful outside of the scheduling decision too, for example in automatically replicating or migrating an entire service, hosted on group of VMs and would like to explore it further.

Measure	Job type	Trace	Vespa	Piao's appr.	Our appr
Percentage Improvement in Avg. Completion Time	Short non-I/O-Intensive	Uni1	15	31	41
		Uni2	26	36	51
		Prv1	23	33	45
	Short I/O-Intensive	Uni1	17	56	43
		Uni2	23	50	59
		Prv1	25	58	55
	Medium-non-I/O-Intensive	Uni1	28	47	58
		Uni2	31	52	62
		Prv1	27	45	68
	Medium-I/O-Intensive	Uni1	32	59	69
		Uni2	37	69	76
		Prv1	29	74	73
	Long-non-I/O-Intensive	Uni1	28	65	68
		Uni2	29	67	81
		Prv1	31	77	79
	Long-I/O-Intensive	Uni1	25	89	83
		Uni2	26	81	86
		Prv1	32	85	88
Variance in Completion Time Improvement	Short non-I/O-Intensive	Uni1	24	25	21
		Uni2	31	37	25
		Prv1	29	35	19
	Short I/O-Intensive	Uni1	27	24	18
		Uni2	36	25	24
		Prv1	37	24	20
	Medium-non-I/O-Intensive	Uni1	26	38	21
		Uni2	34	39	26
		Prv1	29	29	23
	Medium-I/O-Intensive	Uni1	27	15	18
		Uni2	29	27	25
		Prv1	22	23	28
	Long-non-I/O-Intensive	Uni1	26	20	21
		Uni2	33	31	22
		Prv1	19	26	24
	Long-I/O-Intensive	Uni1	23	22	15
		Uni2	31	24	19
		Prv1	28	23	18

Chapter 4

Scheduling in Mobile Cloud

The availability of increasingly richer applications is providing surprisingly wide range of functionalities and new use cases on mobile devices. Even tough mobile devices are becoming increasingly more powerful, the resource utilization of richer application can overwhelm resources on these devices. At the same time, ubiquitous connectivity of mobile devices also opens up the possibility of leveraging cloud resources. Seamless and flexible path to mobile cloud computing requires recognizing opportunities where the application execution on cloud instead of mobile device. In this chapter we describe a cloud aware scheduler for application offloading from mobile devices to clouds. We used learning based algorithm for predicting the gain attainable using performance monitoring and high level features. We evaluated prototype of our system on various workloads and under various conditions.

4.1 Introduction

The exponential growth of Mobile Cloud Computing (MCC) is underpinned by a triad of challenging requirements around three main groups of ubiquity, trust, and energy efficiency. Large variation in platforms, variations in network technologies and quality as well as the devices causes horizontal heterogeneity and complicates MCC more than cloud computing [56]. Ubiquity requires seamless connectivity, context awareness and data integrity. Energy efficiency encompasses remote execution of mobile workflows, latency time reduction, fidelity adaptation, mobile mashup application and resource aware Algorithms.

We propose a Mobile OS scheduler customization and workload migration decision framework for MCC, which can be used to configure and administer the MCC systems dynamically during runtime, such that a majority of the above requirements can be met in an adaptive fashion. This framework is designed and implemented along four modules:

1. Resource usage profiling/fingerprinting
2. Job completion and resource usage prediction
3. Cloud aware scheduler for opportunistic offloading of tasks
4. Dynamic modeling of gain

For job completion and resource usage prediction using learning based algorithms, we use the MADMAC framework we had developed earlier [57]. The idea is to collect data when the app runs natively

and under different circumstance, and using these data to predict the resource requirement and then predict the completion time. Multi-attribute Decision Making (MADM) has been shown to provide a rational basis to aid decision making in such scenarios. We present a MADMAC framework for cloud adoption, consisting of 3 Decision Areas (DA) referred to as the MCC requirements triad: trust, ubiquity and energy efficiency as above. It requires definition of attributes, alternatives and attribute weights, to construct a decision matrix and arrive at a relative ranking to identify the optimal alternative. Decision Support System (DSS) presented showing algorithms derived from MADMAC can compute and optimize Cloud Adoption (CA) decisions separately for the three stages, where the attributes differently influence CA decisions.

In the context of mobile clouds, scheduling algorithms' function is to dynamically allocate threads, processes or data flows to particular nodes in the MCC architecture, which consist of mobile devices and cloud servers (VMs), given access to the state of system resources such as the processor time and communications bandwidth. Goal here is to migrate resource intensive processes, threads or data flows from the mobile device (where resource availability typically is limited) to the cloud servers, to achieve a target quality of service. The need for a scheduling algorithm arises from the requirement for deciding which processes and workflows to migrate, when and how to merge the remotely executed processes back with the mobile-based workflow. For example, a graphics intensive or media intensive application running on the mobile device could be offloaded to a remote server on the cloud, and the results merged back into the mobile workflow. The scheduler makes such runtime decision based mainly on key resources such as the throughput which can be measured in terms of the total number of processes that complete their execution per time unit. Similarly, latency is typically measured as the total turnaround time between submission of a process and its completion. Also important to usability (user experience) is the response time, which is the amount of time taken between the submission of a request and first response is produced on the mobile device.

A modified Wide-band Delphi method is proposed for assessing the relative weights for each attribute, by workload. Relative ranks are calculated using these weights, and the Simple Additive Weighting (SAW) method is used to generate value functions for all the alternatives, and rank the alternatives by their value to finally choose the best alternative. The above algorithms are implemented in a OpenStack based MCC framework proof of concept, using Android as the OS on the mobile device and KVM Linux as the OS for the remote server cloud. For Resource usage profiling, we use established techniques in non-intrusive and scalable way.

The contributions of our work may be summarized as follows:

- A novel scheduling algorithm is proposed for MCC which optimizes on the Gain value of any cloud resource to augment the resources for mobile applications.
- We establish its inclusion in principle into a more elaborate multi-attribute decision framework (MADMAC) for optimizing across multiple, sometimes conflicting criteria.
- We demonstrate the use of learning based algorithm with features which includes not only the system resource utilization and performance related features but also higher level features.
- We proposed an algorithm for offloading decision based on the expected gain while migrating to the cloud.
- We built a prototype of the proposed system and evaluated its performance under various scenarios.

4.2 Related Work

Mobile cloud computing has recently received considerable attention by researchers mainly from academia, but industry-interest is also growing. This has resulted in various efforts by multiple research groups in the area.

Bahl et al. [12] provides an excellent overview of various suggested techniques for mobile cloud environment and unanswered challenges. They predict that mobile applications will become more context aware and personalized, but only envision traditional use cases of social media and content sharing etc. We believe that mobile device have opportunity to provide extremely personal experience if not only the apps , but device operating system also adapts the user. Kovachev D. et al. [36] provides an excellent survey on advances in mobile cloud computing and indicates alternative models of execution for mobile applications. Our approach is similar to the*Augmented Execution Model* mentioned by them, which uses cloned replica of VM on cloud.

CloneCloud [18] provides a framework and runtime for offloading part of the application execution on cloud. They use thread based application partitioning for elastic execution between mobile and cloud. Internet Suspend Resume (ISR) [37] project uses execution and customized state on distributed storage. They used layered virtual machine technology for improving efficiency independent of hardware.

Wanghong Yuan et al. [66] proposed a soft real time scheduler for improving multimedia system on mobile devices. They mainly focus on codec for multimedia applications and focus on soft performance guarantees. Multiple research efforts have focused on special scheduling techniques to reduce battery usage and many solutions have emerged in this space. Aki Saarinen et al. designed SmartDiet [55] for offloading mobile applications to cloud for energy savings using execution traces.

Ajay et al. [60] suggested a logging-based method to achieve VM migration to cloud. They used cryptographic hashing techniques to identify the divergent VM state and use compression based techniques to transfer and replay the state on cloud. The method proposed by them helps in reducing network transfer required, they do not address how to identify migration opportunities.

MADMAC system [57] has defined a framework to evaluate multi-attribute decision making for cloud adoption in various areas. They provide a framework for evaluate various attributes and alternatives for decision making. While they have not applied the framework for mobile cloud, the framework is general and extendable.

4.3 System Model

Figure 4.1 shows our system model and interaction among the components for our system. Scheduler and profiler runs on mobile device and leverages cloud resources through cloud API. One of the goals of our system was to be transparent to the end user and developer. Thus, both scheduler and profiler run as part of mobile operating system and are transparent. We explain main components of the system in following sections.

4.3.1 Mobile Operating System

We chose Android as operating system for mobile device. Android is a Linux based operating system for mobile and tablet devices. Android is currently the leading open source operating system (with more than 75% market-share). Android kernel was forked from the linux kernel and runs each process as a

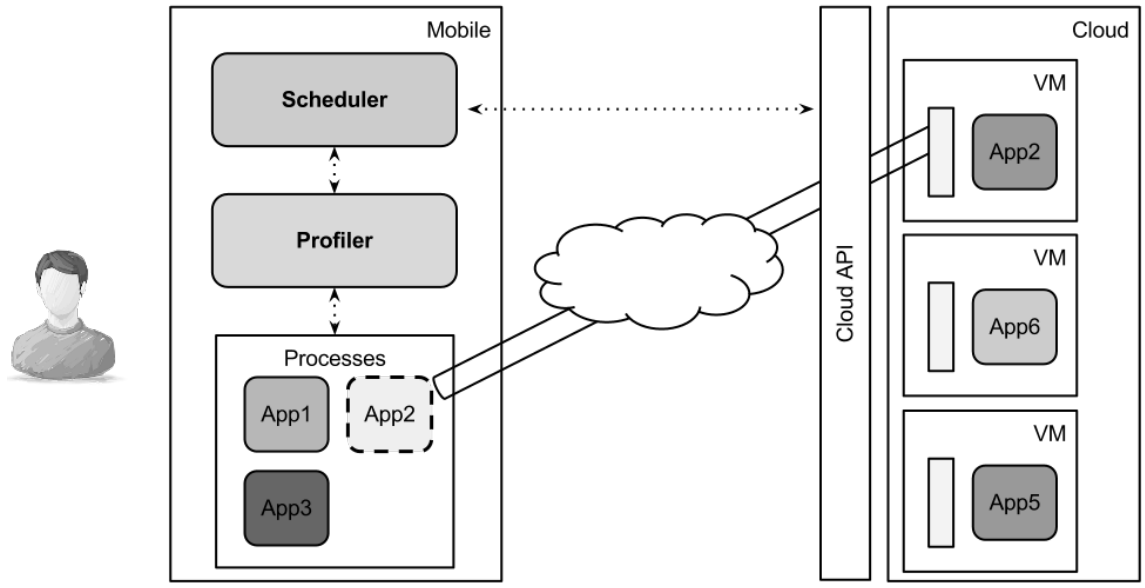

Figure 4.1 The System Model

separate user, isolating applications from other applications. This isolation makes it easier to profile and migrate applications to cloud.

4.3.2 Profiler

The profiler module is responsible for collecting all the statistics (described in based on which the offloading decision is made. It not only collects performance metrics for all user applications, but also gathers higher level features. We used combination of tools like perf and native apis for sensors to get the required data which is then sent to the scheduler. Scheduler also controls the overhead of profiling by limiting the profiler invocations at different rate for different features.

4.3.3 Scheduler

Scheduler is core part of the system and responsible for offloading decisions. Mobile cloud aware scheduler has additional responsibility of including cloud parameters (cost, network connectivity etc.) in scheduling decision. We modified the android scheduler to include these parameters while making offloading decision.

4.3.4 Cloud API

Cloud API provides the functionality of interacting with the cloud infrastructure. Ideally, we would like to use standardized and provider-independent API so that the user can leverage resources from best possible (nearest, cheapest) cloud. While there are efforts in this direction [4], we still do not have

the all-agreed cloud standards. We used OpenStack [7] as our cloud infrastructure management OS and used widely used EC2 [2] APIs which are supported by multiple cloud providers. We used Infrastructure as a Service (IaaS) model for using cloud resources as they provide the most flexibility.

We leverage the ability to run custom images for virtual machines by cloud providers and used custom image, with required softwares pre-installed.

4.4 Proposed Solution

In this section, we describe our proposed solution for offloading decision and execution of migration to cloud.

4.4.1 Profiling and Monitoring

We used following features to model the application and the user interactions with application.

4.4.1.1 Dynamic Features

- High level features: this comprises of features that are concerned to user. It includes battery status, date and time, user location (moving/stable), etc.
- Application features: this captures application usage habits including frequency of usage of the application, stretch of usage, use of local and remote data, etc.
- Network Status : network condition between cloud and mobile device. This includes multiple parameters including bandwidth, latency and stability.
- Resource usage by other applications running on device : This is a combined vector of all individual applications. This is to capture the perceived use of applications and resources.

4.4.1.2 Non-Dynamic Features

- Device Configuration: this captures all the hardware and software configuration of the device. This includes cpu frequency, cpu power steps, operating frequency, etc.
- Cloud Configuration: This captures characteristics of the cloud provider. This includes cost of running a VM for various hardware/software configuration.

4.4.2 Gain Model and Offload decision

The decision to leverage cloud depends on the benefits that can be achieved using cloud. To estimate the gain, we use following model.

$$Gain = \frac{\sum(w_i \times (m_i - c_i)/m_i)}{\sum w_i} \tag{4.1}$$

where w_i is the weight of i the feature gain and are normalized to unity. m_i and c_i are costs of running the application on mobile device and cloud respectively and range from 0 to 1. Costs m_i and c_i are for

each feature described earlier. For example, cost for battery might higher when running on mobile(m_i) compared to running on cloud (c_i). On the other hand, cost of network usage can be higher while application runs on cloud(c_i than running on mobile device (m_i) for highly interactive applications. The reason for using w_i in above model is to allow giving different importance to different features while estimating the gain.

Algorithm 7 Offloading Decision

for each application i **do**
  Update the model with current metrics
  predict c_i values for all the features using model
  $Gain_p$ according to Eq. 4.1 predicted values of c_i and measured values of m_i
  if $Gain_p \geq significance\_threshold$ **then**
    Execute the p remotely on cloud.
  else
    continue executing p locally.
  end if
end for

Offloading algorithm is described in Algorithm 7. For each application, the model is updated in online setting with current metrics for various features and new values for c_is are predicted by the scheduler prediction module. The $Gain_p$ is calculated with new values of c_i and measured values of m_i and compared against $significance\_threshold$. If the $Gain_p$ is sufficiently high, the process is migrated to cloud and user interactions continue remotely. Otherwise next application is tried for possible optimization. The choice of parameter $significance\_threshold$ controls aggressiveness of offloading decisions. Very high values of $significance\_threshold$ indicate that algorithm will run most of the applications locally while lower values will allow applications to be offloaded to cloud even if gain achieved is not very high. We suggest its value between 0.25 to 0.65.

4.4.3 Migration

We ran applications in isolated process in android environment which simplifies the migration of application on cloud. We would like to note that focus of our work is not to improve migration of applications to cloud and relied on the simple standard techniques for migrating user application from the mobile device to cloud. We used remote frame buffer protocol [54] to continue the user interaction after migration to cloud. We used VNC viewer on the mobile side and ran VNC server on the cloud side. The algorithm runs at each epoch.

4.5 Experiments and Evaluation

In this section we describe our prototype implementation and experiments to validate our approach. Figure 4.2 describes our implementation. We used Vowpal Wabbit (vw) [39] for learning weights (w_i). Vowpal Wabbit is a fast machine learning package. We considered the problem as a regression problem and used online learning with square loss function. Because of the online learning we can also control the overhead of profiling by varying profiler invocation frequency. We used traffic

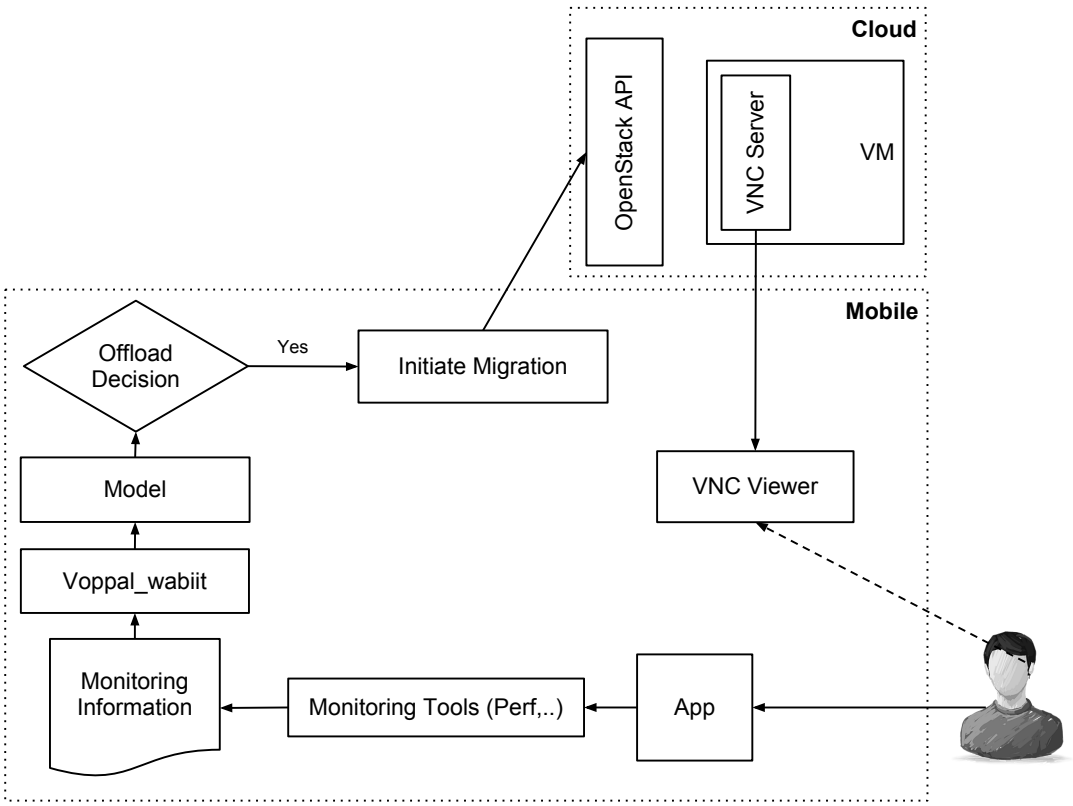

Figure 4.2 Implementation Details

control utility (tc) [6] for emulating performance with various network scenarios.We only have one parameter for user ($significance\_threshold$) which makes it easy to use. This results were noted at $significance\_threshold$ set to 0.35. Table 4.5 describes our experimental setup. We evaluated our system for representative workloads from 4 different classed shown in Table 4.5. As described theses workloads cover range of applications various levels of interactivity, resource usage and runtime.

Parameter	value
Cloud Operating System	Ubuntu 12.04(kernel 3.2)
Cloud VM configuration	4 GB, 2.66GHz
Device Operating System	Android 4.2
Device Configuration	1GB, 1.5 GHz

Table 4.1 Experimental Setup

Workload	Description	Characteristics
Kernel	kernel download + build	long + resource intensive
GIMP	Image editing + applying image filters	interactive + little intensive
Video conversion	download & convert a (500MB) video	short + resource intensive
Browser	browsing 5 sites	interactive

Table 4.2 Workloads

We start with uniform weights and each time scheduler decides to offload to the cloud, we asked user to approve the decision and showed the gain and other related information. The users approval or disapproval is fed back into the learning algorithm as training data and algorithm uses them to modify the importance of features (w_i). As this process continues the scheduler becomes more personalized about user preferences for various features. For example, some users might prefer to get results fast by running on cloud while others might be more concerned about the monetary cost of using cloud. So the personalized scheduler becomes *aware of* context and user preferences.

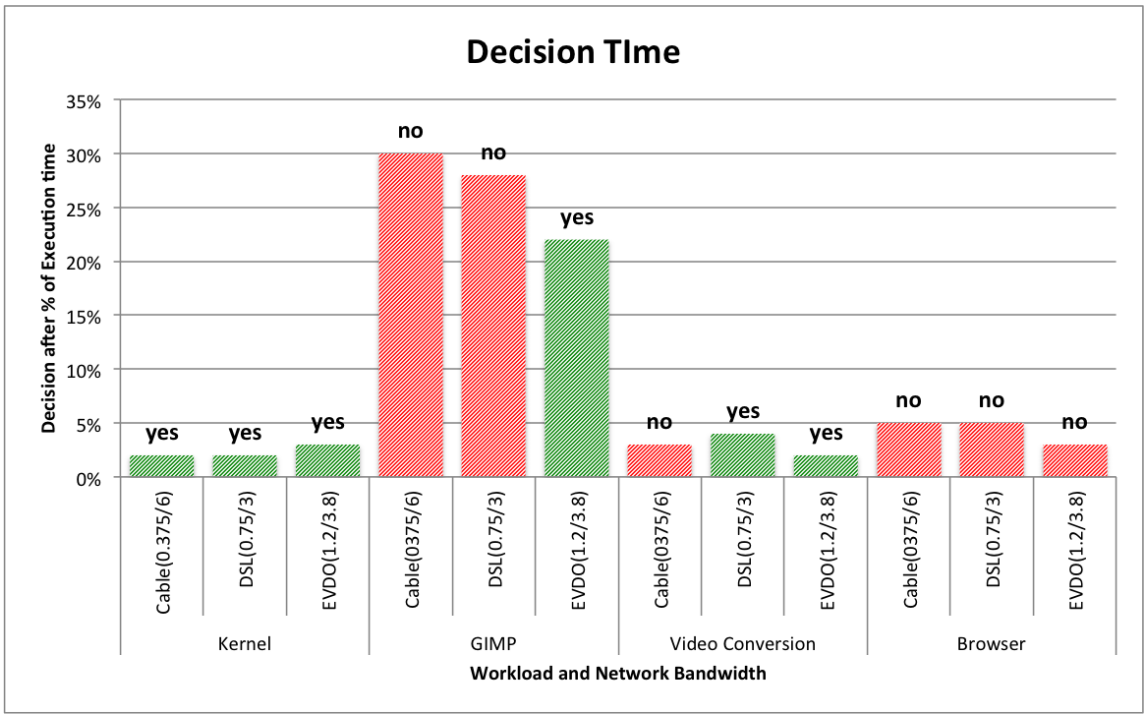

Figure 4.3 Decision Time for various Applications under various network scenarios

Figure 4.3 shows the output of decision and time taken for making it for various workloads under various network scenarios. We emulated three classes of bandwidth cable(0.375/6), Digital Subscriber

Line (DSK 0.75/3) and Evolution-Data Optimized (EVDO 1.2/3.8) where numbers in bracket specifies the uplink and downlink bandwidths in Megabits per sec. This is plotted against the the % of time application run when the decision was made about offloading. The decision of offloading is shown on top of bar. For kernel workload algorithm was able to decide early (within 2%) that it is suitable for cloud offloading. Similarly for browser workload also, it was able to understand the unsuitability of offloading early-on (within 5%). For GIMP workload, the algorithm spent time in understanding the user interaction and decided not to offload in case of poor network condition but decided to offload if network condition is good. This is important as poor network condition will result in poor user experience for interactive workloads. For video conversion workload the algorithm decided to offload even if the network condition is not very good. This is again expected as the video conversion workload is non-interactive and intensive, the algorithm rightly decided to offload.

Figure 4.4 shows the overhead of our approach for various number of applications. We measured overhead as the percentage increase in the resource utilization with and without running our system.

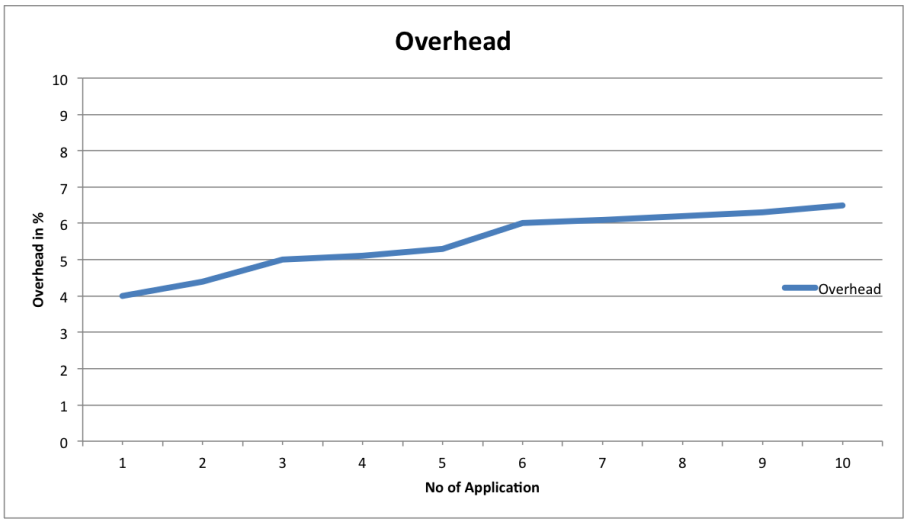

Figure 4.4 Overhead of running our system for increasing number of applications

4.6 Conclusion

In this chapter, we described a scheduling algorithm which tries to make use of cloud resource to augment the resources for mobile applications. We used learning based algorithm with features which includes not only the system resource utilization and performance related features but also higher level features. We proposed an algorithm for offload decision based on the expected gain while migrating to the cloud. We built a prototype of the proposed system and evaluated its performance under various scenarios. We found that our approach is able to identify high level workload characteristics and make appropriate decision about running application locally or on cloud.

Chapter 5

MultiStack: BigData on MultiCloud

The shift towards cloud marketplace due to instant availability of cheap resources through public cloud providers. Organizations want to leverage private and public cloud resources seamlessly and the hybrid cloud adoption is growing. As cloud computing is growing, the enterprises have various cases suited for various clouds. They want to run many tasks on their on-premise for performance or security reasons, while want to leverage public resource for other tasks. Many enterprises today use cloud resources from providers. There is currently no multi cloud framework for evaluating and developing multi cloud applications. We present MulitStack as our attempt to fill that gap and we take BigData applications as our first use-case. We believe that the availability of such framework will lead to better understating of MultiCloud environment.

There are multiple challenges involved in implementing such a framework. For example, abstracting out the differences between providers. As, cloud computing has evolved in the industry much faster the formation of common standard, different cloud providers do not agree on various abstraction that they provide. Unit of compute capacity is one such example. While hiding the difference between cloud providers, its also important to take advantage of provider specific features. The performance of different cloud providers differ on various matrices. In this chapter, we present design and implementation of MultiStack, a multi cloud framework and present data processing frameworks on top of it. We target data processing as our first application because data processing has been a major cloud use-case.

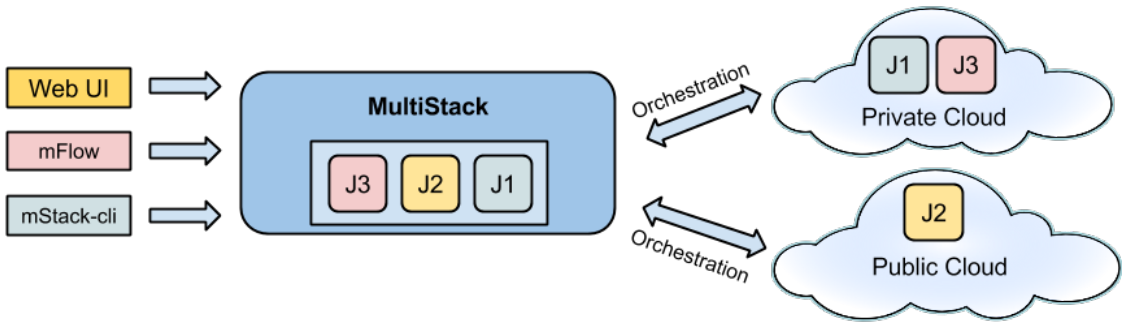

Figure 5.1 MultiStack Overview

We aim to provide following features in MultiStack.

- Ability to run across multiple cloud providers - If you have multiple jobs and access to multiple cloud providers, MultiStack provides you the ability to run different jobs on different clouds.

- Priority based Job scheduling for minimizing cost and completion time - MultiStack uses machine learning to smartly allocate jobs across multiple cloud providers aiming to reduce your cost and job completion time.

- Auto Scaling - Based on the deadline, infrastructure and cost requirements, the scheduler will smartly scale the resources allocated to a job. The user is also provided an option to manually scale up/down.

- Performance optimization with storage integration - With storage administration privileges, MultiStack auto-replicates the resources that being utilized heavily, thus improving performance.

5.1 Architecture

MultiStack as a MultiCloud frameworks have many components to achieve its goal. Figure 5.2 shows various components and their interactions. As a modern operating system for MultiCloud all the interactions are among components are asynchronous and through ReST calls. The plugin architecture makes it very easy to adapt and extend the system to suit to particular environment. Also, all the components are designed to scale horizontally to accommodate mode load.

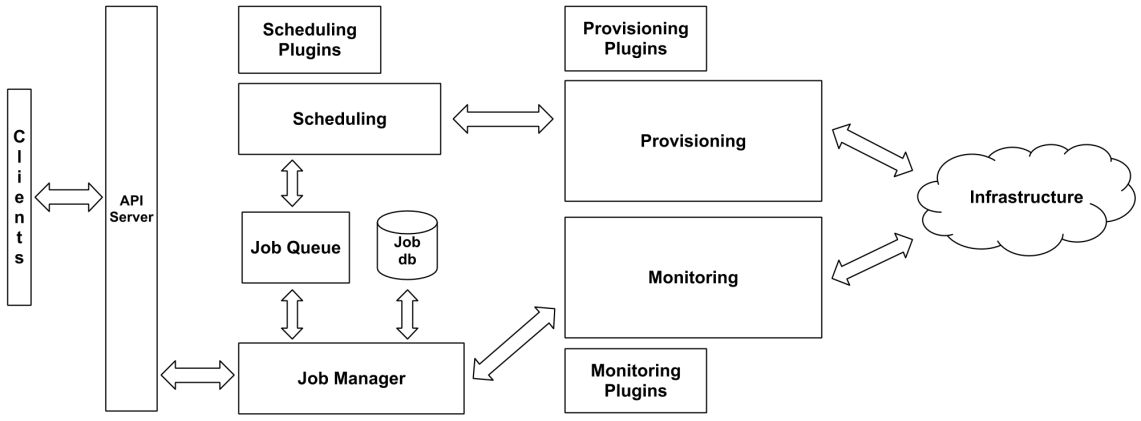

Figure 5.2 MultiStack Architecture

5.1.1 API Server

One of the central design choice in MultiStack is making everything pluggable and available and extendable through APIs. The table below lists current API,

Table 5.1 API Description

API Group	HTTP verb	Description
Cluster	POST	Create a cluster
	PUT	Add machines to existing cluster
	GET	List cluster(s)
	DELETE	Destroy a cluster
Job	POST	Run a job
	PUT	Add machines to existing job
	GET	List job(s)
	DELETE	Cancel a job

5.1.2 Job Manager

Job manager is responsible for management of the life cycle of the jobs. It includes job manager which maintains job queuing and various statistics about jobs. It maintains this information in a database and makes it available through API requests. It also persists all the user related information including credentials, quota, job status, various aggregates regarding cost, resource usage etc, which are used by multiple components. Web-interface uses this information to provide dashboard and scheduler uses this information in scheduling decisions.

5.1.3 Provisioning

Provisioning module is responsible for spawning and configuring instances for the job. It also has deprovisioning module responsible for deallocation of resources after the job has finished execution. Different plugins for provisioning handle provisioning and deprovisioning for various clouds. Currently MultiStack has provisioning plugins for OpenStack [7] and Amazon EC2 [1].

5.1.4 Monitoring

Monitoring is very important part of the system as availability of detailed monitoring information is essential for making optimized choices. This subsystem is responsible for aggregates system info from multiple instances, consolidating them and providing them as requested through API. The monitoring includes

- Job Monitoring - Collects information regarding the status and progress of jobs. This is provided by job specific monitoring integrations. For example, for monitoring Hadoop jobs we integrate with Hadoop monitoring subsystem.
- Cloud Monitoring - Collects monitoring information of instances running across various clouds. This includes minimal resource usage information of CPU, memory, disk and network. It does this by It can also provide more elaborated information via plugging into the monitor
- Miscellaneous Information - This includes informations which are traditionally not considered as monitoring information. For example price information of resources in various clouds,

5.1.5 Client tools

The client tools simplifies the interactions with the system.

- Web Interface - The web component of MultiStack will allow a user to submit jobs, mention the input and output location and the deadline. The admin interface has setup component where it accepts credentials for various cloud providers and allows user management. The resource allocation will be completely transparent to the user and optimized to minimize cost and job completion times.
- mFlow - Its a web interface for composing job workflows. It allows users to chain different jobs functions. These functions can be submitted by a user or may belong to an existing repository.
- mstack-cli - This component is built keeping devops in mind. Using command line tool, a variety of jobs can be automated.

5.2 Policy-based scheduling

One of the simplest and yet effective scheduling technique in multi cloud is policy based. Policy based scheduling carries out scheduling decisions to obeying the policy laid out by the administrator. The policy can be based on business requirement (like keeping some data within organization boundary) or it can be based on resource requirements (like need of particular system for a job). The policy based scheduling is well studied in the literature and trivially extendable to multi cloud scenarios.

5.3 Cost aware scheduling in MultiCloud and optimal cluster sizing

Scheduling in MultiCloud has many challenges associated which are not present in traditional cluster scheduling. The difference come from various sources including new model of costs, vary little control of internal scheduling provided by cloud providers, the illusion of infinite resources and so on. Thus there are new tradeoffs like instant availability of infinite resources in exchange of cost. This can lead to much better turnaround time for scale out workload at potentially same cost. Also new cost structure like resource auctioning sets new dynamic in scheduling. We describe next how to bring cost awareness in multi cloud scheduling.

Cluster size has a direct impact on the cost and completion time of jobs in cloud environment. Cloud offers different instance size and properties at different properties. For example, spot instances in aws marketplace allow users to bid for spare Amazon EC2 compute capacity. Spot instances provides a potentially very cost effective way to scale clusters especially big data clusters, since the data processing frameworks have ability to tolerate loss of nodes from clusters. We try to optimize the cluster size for effectively reducing cost and turnaround time. Also, deadline, current progress and predicted completion time would help in public cloud environment where costs are calculated on hour-boundaries. Based on that, we formulate the following as the optimal cluster size,

$$OptimalInstanceCount = \arg\max_i((d_m - d_i) - \beta \times Cost(i) \times d_i - \gamma i) \tag{5.1}$$

where, d_m is estimated deadline with running current set of instances, d_i is estimated deadline after scaling the cluster with i instances and $Cost(i)$ is estimated cost of running i instances. The cost can

be fixed or simple regression over past pricing data. β and γ represent the weights that controls penalty. With this equation we are trying to maximize the gain achieved by adding i more instances. The first term captures time saving by adding extra instances. Second term represents the cost of running i spot instances. We penalize spawning many more instances by using third term. Based on Eq. 5.1, we present following algorithm for cluster scaling,

Algorithm 8 Cost-aware autoscaling of clusters

1: Monitoring data about each Job
2: find optimal type of instance for scaling using eq. 5.1
3: Predict the expected finish time with current set of instances
4: **if** Current instance cost $<$ Upper limit by user **then**
5: Find optimal number of spot instances to add using spot instance price history and deadline prediction using equation 5.1
6: **end if**

MultiStack will provide users the ability to run different jobs on multiple cloud providers. It provides both command line and web-based clients to submit jobs. It employs smart scheduling to estimate job completion time and costs involved, and schedule them accordingly to minimize both.

5.4 Big Data Use-cases

Big data processing was one of the popular use case for adoption for cloud computing and we believe big data clusters are good candidates to demonstrate the benefits of multicloud scenarios. In this section, we give a brief overview of the wide spread data processing frameworks, that are supported by MultiStack. We are looking forward to integrate other frameworks (for example, GraphLab) with MultiStack.

5.4.1 Hadoop

Hadoop and tools from Hadoop family are the most popular data processing tools and are synonyms for big data processing for many. Apache Hadoop [3] is an open source implementation of MapReduce [20] for large scale distributed batch processing. Hadoop has shown the possibility of using commodity machines to process huge amount of data very fast. Hadoop processing on cloud has increasing adoption and Hadoop-as-a-Service is expected to grow tenfold. [1]

5.4.2 Berkeley Data Analytics Stack

Berkeley Data Analytics Stack (BDAS) is a data processing tool-chain developed by AMPLab. Currently we are focused on the above two data analytics stacks as they provide tools for analyzing data from a wide range of applications. The stack provides tools for fast and memory optimized processing including spark, MLBase for large scale machine learning, tachyon for in-memory file system caching among others. Thus it is quickly becoming popular among practitioners and by supporting that as big data framework, we also validate the generalization of the platform.

[1] http://www.technavio.com/report/global-hadoop-service-market-2012-2016

5.5 Summary

In this chapter, we detailed MultiStack, a multi cloud operating system and building big data applications on top of it. We discussed architecture, design and implementation of MultiStack. We also described how scheduling problems of such system is drastically different from traditional cluster schedulers and described few simple scheduling techniques. MultiStack is under development and there are lot of interesting features being added. We hope that MultiStack will be used as platform for MultiCloud research and provide better understanding and solution of MultiCloud problems.

Chapter 6

Contributions and Further Directions

The growth of cloud computing has posed new challenges for data center operators. We discussed few challenges involved in scheduling and approaches to solve them. In this chapter we describe summary of approaches we proposed for different scenarios and conclude the thesis with some of exciting current and future directions of this work.

6.1 Contributions

Scheduling has a huge impact on the performance of the any system and is a very challenging problem, especially in the large scale deployment. We focused on different challenges associated with scheduling in different settings in this thesis.

6.1.1 Energy Efficiency and SLAs

Energy consumption in large scale cloud deployments, is not only a significant cost factor for cloud operators, but also have severe negative impact on environment. Service Level Agreements (SLA) is a way to ensure that business critical application and workloads have sufficient resources all the time. Because SLAs are so important to the business, its often the case that operators allow excessive resources to them, which leads to significant wastage of electric power. A virtualized data center mitigates the resource underutilization problem by running multiple of these applications on a single physical host.

Applications hosted via virtual machines(VM) in typical data centers, only partially use the resources allocated to it. Thus a consolidation algorithm can allow dense mapping of VMs to physical machines, but it has to be careful about maintaining SLAs of these applications. Thus a consolidation algorithm, aware of SLAs is needed. We proposed one such algorithm that uses resource usage characteristics to make effective consolidation, scale-up and scale-down decisions. The proposed algorithm relies on similarity model combining resource usage characteristics for CPU, Memory, disk I/O and network bandwidth. We compared performance of our algorithm with other well established algorithms and reported significant energy savings while minimally effecting SLAs.

While servers are primary energy consumers, network infrastructure also consumes significant electric power in large scale deployment. We showed how a VM-network co-scheduling algorithm can achieve better energy efficiency of network infrastructure using knowledge of network flows. The algorithm based on network flow information, can put network devices in low power mode.

6.1.2 Network awareness

Powering modern consumer, scientific and enterprise applications often requires large amount of processing, which is increasingly difficult to achieve on a single machine. Thus most modern applications are build using scale out architecture. Hadoop, various graph processing systems, analytics systems, and other big data systems and scientific systems require a lot of data and messaging between the nodes in the systems. When these applications are run within virtual machines, the network performance between these VMs can degrade significantly. These rise of the East-West traffic compared to North-South traffic in data center, poses challenges for network scalability and application performance. For example, a VM placement algorithm, which does not take this into account, can place VMs belonging to the same application far apart in network, which not only results in poorer application performance, but also burdens the network infrastructure.

We proposed a network aware VM consolidation method, specifically we proposed VMCluster identification algorithm to identify group of VMs with high network traffic among them that can highly benefit from better placement and VMCluster placement algorithm to place the identified VMClusters in a greedy fashion to localize network traffic. We compared our consolidation method with traditional algorithms on number of different parameters on real world traces. We consistently achieved very high localization of network traffic, leading to better application performance. Stability of VM placement is very important for data center operators. We evaluated our method on number of migrations required to achieve traffic localization and repotted most of the benefits come from a very small number of migrations.

6.1.3 Mobile cloud

The always connected mobile device with feature rich applications, opens up an interesting opportunity to leverage cloud resources for mobile applications. Most of the new apps are cloud enabled, there are multitude of applications which can benefit from using cloud resources. Instead of each app doing this on its own, the mobile scheduler might be able to use cloud and mobile resources more intelligently.

We proposed a learning based cloud-aware mobile scheduler, which identifies application execution that will benefit by leveraging cloud and does a transparent tunneling to applications between mobile and cloud resources. The cloud offloading decision is modeled as a learning problem using Multi-Attribute Decision Making framework. We used not only system utilization metrics but also high level features in offloading decision. Algorithm tried to predict the gain from a cloud offloading and based on this gain scheduler decides to run application locally or on cloud. We implemented our algorithm on Android Mobile operating system and evaluated for a variety of workload under different conditions.

6.1.4 MultiCloud

The cloud marketplace has different providers providing different resources at different cost. Enterprises increasingly are using resources from multiple clouds. The lack of common standards and APIs means that for using available resources from multiple clouds, we need a framework to hide the differences among clouds. We proposed one such framework, targeting big data applications for multi-cloud environment. We described scheduling challenges in multi-cloud environments and features of such a system.

6.2 Future Directions

VM Scheduling is only one of the component (although most important one) in data centers. We now describe, some of the directions in which the current work is being leveraged or extended to improve VM scheduling.

6.2.1 Benchmarking and monitoring for Cloud and its relation to scheduling

During our work on scheduling, one of the important realization was that there was no repeatable and portable way of comparing performance of different algorithms in different cloud environments. The need for portable, scalable and simple benchmarking suite for comparison of various algorithms including scheduling is very useful in research and production. Traditional benchmarking suits are not well suited for cloud environments.

We characterize the process resource consumption and hardware capabilities. We propose a cloud benchmarking tool that is,

- Portable across hardware
- Repeatable in different environment
- Adoptable for wide range of cloud application
- Scalable to very large cloud environment

We have done some initial work on the design and implementation of a cloud benchmarking suite, YACB (Yet Another Cloud Benchmarking) [9] to address these needs. We briefly describe the components of YACB design,

- Monitoring - This module is responsible for collecting fine grained monitoring data. The existing monitoring system can be easily integrated with this module as the architecture is pluggable. The monitoring information generated is portable across clouds and workloads.
- Model Generation - This module uses monitoring information and characterizes various processes. It also generates binary code that can be executed on actual target cloud to measure the performance of application on the target cloud.
- Performance Estimation - This component is useful in scenarios where we don't have access to the target environment and want to evaluate application performance. It provides estimation of resource consumption of target environment. It takes care of heterogeneity in workloads and hardware. It is useful also to-do what-if evaluations.
- Scheduling - This component is responsible for scheduling processes in the given environment using monitoring information. This again is pluggable and we provide implementation of various scheduling policies with the tool.

We have made some progress in building YACB and are in the process of evaluating the performance of our tool in terms of the listed criteria. We believe that the YACB will be useful for wide range of applications and environment. Apart from evaluation of scheduling policies in academic rigorous manner, it can be used for workload modeling, identifying performance bottlenecks and ideally solving them with the integration of the rest of the resource management framework.

6.2.2 Performance modeling

Modeling resource requirements of applications can not only benefit scheduling decision, but also is helpful in understanding behavior of the application under different resource availabilities. The modeling of distributed jobs is thus a very useful problem to solve in large scale environment.

We are working on Perforator, a system for modeling resource requirement and execution time for MapReduce jobs created by Hadoop, Hive or Dryad. We model key phases of a MapReduce job using read, compute, write and shuffle operations. We do a critical path analysis on the graph containing these operations running in a distributed environment. To estimate cost of each operations we leverage combination of micro-benchmarking, execution history and sample-runs. The initial results of Perforator are promising and we are working to integrate it with YARN scheduler to improve resource management [8].

6.2.3 Combing storage sub-system with VM scheduling

Storage subsystem is very critical for the overall performance of applications in data center. Distributed Storage systems like CephFS, GlusterFS are a preferred choice for communities and enterprises for running Big Data Applications. With the ability to access data through multiple interfaces, they act as unified storage solution, fulfilling all storage requirements. While this generic behavior is suited for most use-cases, a performance demanding job may be heavily affected. Traditionally approaches of load balancing are not well suited for unified storage system with varying load patterns. Recently there has been some development in optimizing replica placement object based storage systems These approaches are limited to use of access frequency only.

Dynamic replication is a challenge for Object Storage systems. It requires identifying optimal number of replicas and placement for an object. It would be interesting to explore dynamic prediction of object demands and identify optimal number of replicas and their placements. Object demand can be used to adjust object replicas and move objects across different storage layers (archived, not-replicated, highly-replicated etc.). These will be more efficient and cost effective as unused data is automatically archived and high demand data will be available on more nodes. Uncontrolled auto-replication can lead to performance issues. We take into account available network resources along with other factors in deciding replica count and their placement. The integration of such intelligent system coupled with scheduling can benefit not only to data processing frameworks but other workloads too. Exploring the opportunities of such integration will be valuable.

6.3 Summary

We would like to conclude this thesis in hope that this will be useful for wide range of researchers and practitioners in the area. We hope that our work and developed system will be beneficial in advancing state of the art for cloud computing community.

Related Publications

- **MECCA: Mobile, Efficient Cloud Computing Workload Adoption Framework using Scheduler Customization and Workload Migration Decisions.**
  Dharmesh Kakadia, Prasad Saripalli and Vasudeva Varma.
  In *Proceedings of the first international workshop on Mobile cloud computing and networking (MobileCloud '13)* collocated with MobiHoc. ACM, New York, NY, USA, 41-46.

- **Optimizing Partition Placement in Virtualized Environments**
  Dharmesh Kakadia and Nandish Kopri.
  Patent No. P13710918. Filed December 2012.

- **Network-aware Virtual Machine Consolidation for Large Data Centers.**
  Dharmesh Kakadia, Nandish Kopri and Vasudeva Varma.
  In *Proceedings of the third international workshop on Network-aware Data Management (NDM '13)* collocated with SuperComputing'13. ACM, November 17, 2013, Denver CO, USA.

- **Energy Efficient Data Center Networks - A SDN based approach.**
  Dharmesh Kakadia and Vasudeva Varma.
  In *IBM Collaborative Academia Research Exchange (I-CARE)* 2012. Bangalore, India.

Bibliography

[1] Amazon elastic compute cloud (ec2). http://aws.amazon.com/ec2.

[2] Amazon web services. http://aws.amazon.com/.

[3] Apache Hadoop. . http://hadoop.apache.org/.

[4] The cloud standards. http://cloud-standards.org/.

[5] Floodlight openflow controller. http://www.projectfloodlight.org/.

[6] Linux Advanced Routing & Traffic Control. http://www.lartc.org/.

[7] Openstack open source cloud computing software. http://openstack.org/.

[8] Perforator project - tech report. `http://research.microsoft.com/en-us/projects/perforator/`.

[9] Yet another cloud benchmark. https://github.com/dharmeshkakadia/YACB.

[10] 1E. Server Energy and Efficiency Report. Technical report, 1E, 2009.

[11] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. *SIGARCH Comput. Archit. News*, 38(3):338–347, June 2010.

[12] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan. Advancing the state of mobile cloud computing. In *Proceedings of the third ACM workshop on Mobile cloud computing and services*, MCS '12, pages 21–28, New York, NY, USA, 2012. ACM.

[13] L. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33 –37, dec. 2007.

[14] A. Beloglazov and R. Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 577–578, Washington, DC, USA, 2010. IEEE Computer Society.

[15] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.

[16] K. Birman, G. Chockler, and R. van Renesse. Toward a cloud computing research agenda. *SIGACT News*, 40(2):68–80, June 2009.

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

[18] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[19] W. Dally and B. Towles. *Principles and practices of interconnection networks*. Morgan Kaufmann, 2003.

[20] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[21] F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the sprite implementation. *Software - Practice and Experience*, 21:757–785, 1991.

[22] E. Pinheiro, R. Bianchini, E. V. Carrera and T. Heath. Dynamic cluster reconfiguration for power and performance. In *Power and energy management for server systems*. IEEE Computer Society, November 2004.

[23] EPA. EPA Report to Congress on Server and Data Center Energy Efficiency. Technical report, U.S. Environmental Protection Agency, 2007.

[24] S. K. Garg and R. Buyya. Networkcloudsim: Modelling parallel applications in cloud simulations. In *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing*, UCC '11, pages 105–113, Washington, DC, USA, 2011. IEEE Computer Society.

[25] Gary Cook, Jodie Van Horn. How dirty is your data? A Look at the Energy Choices That Power Cloud Computing. Technical report, Green Peace International, April 20, 2011.

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

[27] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, Dec. 2008.

[28] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 39(4):51–62, Aug. 2009.

[29] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *SIGCOMM Comput. Commun. Rev.*, 39(4):63–74, Aug. 2009.

[30] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 17–17, Berkeley, CA, USA, 2010. USENIX Association.

[31] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[32] C. Hsu and W. Feng. A power-aware run-time system for high-performance computing. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 1, Washington, DC, USA, 2005. IEEE Computer Society.

[33] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible. Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 72–79. IEEE, 2011.

[34] Y. A. Khalidi, J. M. Bernabeu, V. Matena, K. Shirriff, and M. Thadani. Solaris mc: a multi computer os. In *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, pages 16–16, Berkeley, CA, USA, 1996. USENIX Association.

[35] K. H. Kim, R. Buyya, and J. Kim. Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '07, pages 541–548, Washington, DC, USA, 2007. IEEE Computer Society.

[36] D. Kovachev and R. Klamma. Beyond the client-server architectures: A survey of mobile cloud techniques. In *Communications in China Workshops (ICCC), 2012 1st IEEE International Conference on*, pages 20–25, Aug.

[37] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, WMCSA '02, pages 40–, Washington, DC, USA, 2002. IEEE Computer Society.

[38] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. In *Proceedings of the 2008 International Conference on Autonomic Computing*, ICAC '08, pages 3–12, Washington, DC, USA, 2008. IEEE Computer Society.

[39] J. Langford, L. Li, and A. L. Strehl. Vowpal wabbit (fast online learning), 2007. `https://github.com/JohnLangford/vowpal_wabbit/wiki`.

[40] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[41] Y. C. Lee and A. Y. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 92–99, Washington, DC, USA, 2009. IEEE Computer Society.

[42] C.-C. Lin, P. Liu, and J.-J. Wu. Energy-aware virtual machine dynamic provision and scheduling for cloud computing. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 736–737, july 2011.

[43] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. A power benchmarking framework for network devices. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 795–808, Berlin, Heidelberg, 2009. Springer-Verlag.

[44] N. Maheshwari, R. Nanduri, and V. Varma. Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework. *Future Generation Comp. Syst.*, 28(1):119–127, 2012.

[45] P. Mell and T. Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011.

[46] X. Meng, V. Pappas, and L. Zhang. Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, Mar. 2010.

[47] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 39–50, New York, NY, USA, 2009. ACM.

[48] H. Nguyen Van, F. Dang Tran, and J.-M. Menaud. Autonomic virtual resource management for service hosting platforms. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[49] M. S. Pérez, A. Sánchez, J. M. Pena, and V. Robles. A new formalism for dynamic reconfiguration of data servers in a cluster. *Journal of Parallel and Distributed Computing*, 65(10):1134 – 1145, 2005. Design and Performance of Networks for Super-, Cluster-, and Grid-Computing Part I.

[50] D. Petrou, S. H. Rodrigues, A. Vahdat, and T. E. Anderson. Glunix: A global layer unix for a network of workstations. *Softw., Pract. Exper.*, 28(9):929–961, 1998.

[51] J. Piao and J. Yan. A network-aware virtual machine placement and migration approach in cloud computing. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 87–92. IEEE, 2010.

[52] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *In Workshop on Compilers and Operating Systems for Low Power*, 2001.

[53] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[54] T. Richardson and J. Levine. The remote framebuffer protocol. 2011.

[55] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui. Smartdiet: offloading popular apps to save energy. In *SIGCOMM*, pages 297–298, 2012.

[56] Z. Sanaei, S. Abolfazli, A. Gani, and R. H. Khokhar. Tripod of requirements in horizontal heterogeneous mobile cloud computing. *arXiv preprint arXiv:1205.3247*, 2012.

[57] P. Saripalli and G. Pingali. Madmac: Multiple attribute decision methodology for adoption of clouds. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, pages 316–323, Washington, DC, USA, 2011. IEEE Computer Society.

[58] J. E. Smith and R. Nair. The architecture of virtual machines. *Computer*, 38:32–38, May 2005.

[59] J. Sonnek, J. Greensky, R. Reutiman, and A. Chandra. Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. In *Parallel Processing (ICPP), 2010 39th International Conference on*, pages 228 –237, sept. 2010.

[60] A. Surie, H. A. Lagar-Cavilla, E. de Lara, and M. Satyanarayanan. Low-bandwidth vm migration via opportunistic replay. In *Proceedings of the 9th workshop on Mobile computing systems and applications*, HotMobile '08, pages 74–79, New York, NY, USA, 2008. ACM.

[61] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 331–340, New York, NY, USA, 2007. ACM.

[62] Q. Tang, S. Gupta, and G. Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *Parallel and Distributed Systems, IEEE Transactions on*, 19(11):1458–1472, 2008.

[63] A. N. Tantawi. A scalable algorithm for placement of virtual clusters in large data centers. *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 0:3–10, 2012.

[64] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 1163–1171, Piscataway, NJ, USA, 2010. IEEE Press.

[65] J. Xu and J. A. B. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, pages 179–188, Washington, DC, USA, 2010. IEEE Computer Society.

[66] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 149–163, New York, NY, USA, 2003. ACM.