University of Moratuwa

Department of Computer Science and Engineering

CS4202 - Research and Development Project

Final Year Project Report

Resource and Policy Aware Virtual Machine Scheduling in Medium-Scale Clouds

Project Group - VirtualOps

Infaz M.I.M (100202X)

Jayasekara P.M.L.R.D.P. (100216T)

Money V.A. (100341V)

Hashan M.T.S. (100180B)

Project Supervisor

Dr. H.M.N. Dilum Bandara

Coordinated By

Dr. Malaka Walpola

12th February, 2015

Abstract

Project Title	Resource and Policy Aware Virtual Machine Scheduling in Medium-scale clouds
Authors	Infaz M.I.M – 100202X
	Jayasekara P.M.L.R.D.P. – 100216T
	Money V.A. – 100341V
	Hashan M.T.S. – 100180B
Coordinator	Dr. Malaka Walpola
Supervisor	Dr. H.M.N. Dilum Bandara

Medium-scale clouds are being widely used in small/medium scale organizations such as enterprises and universities. Unlike public clouds like Amazon Web Services, medium-scale clouds have limited amount of resources in terms of computing power, memory, storage and networking resources. But there may be various cloud users with varying requirements; hence, resource requirements vary too. For example, in a medium-scale university cloud, users would include lecturers who need to deploy labs within specific time slots, students who need to run various application for classes and projects, and researchers who need to run their HPC applications in the cloud. Besides these, there may be other HPC applications which can be run on the cloud when there's no other applications running (e.g., at night) to utilize the resources. To satisfy all these requirements and utilize the existing resources in an optimum manner, various priority policies also needs to be specified. For example, a lecturer who needs to deploy a lab has to have higher priority over others, and an internal researcher should have higher priority over an external researcher. Infrastructure as a Service (IaaS) clouds needs to be aware of existing computing resource as well to determine where to spawn a new Virtual Machine (VM) with the required resources for each new incoming request. But, currently available IaaS platforms such as CloudStack, OpenStack, and Eucalyptus are not capable of supporting such dynamic VM deployment while being aware of existing resources, pre-defined policies, and user requirements. The goal of this project is to extend an existing IaaS framework to support such complex and dynamic VM deployments. This includes development a VM scheduler which can keep track of existing resources and deploy VMs on the cloud dynamically and support pre-emption of VMs for priority-based scheduling.

Table of Contents

List of Figures

List of Tables

Abbreviations

S3	-	Simple Storage Service
API	-	Application Program Interface
AR	-	Advanced Reservation
AWS	-	Amazon Web Services
BE	-	Best Effort
BIOS	-	Basic Input Output System
CLI	-	Command Line Interface
CPU	-	Central Processing Unit
CUDA	-	Compute Unified Device Architecture
DLS	-	Deadline Sensitive
DPM	-	Distributed Power Management
DRS	-	Distributed Resource Scheduler
EC2	-	Elastic Compute Cloud
HPC	-	High Performance Computing
HTTP	-	Hypertext Transfer Protocol
IaaS	-	Infrastructure as a Service
KVM	-	Kernel-based Virtual Machine
LXC	-	Linux Containers
NFS	-	Network File System
PaaS	-	Platform as a Service
PHP	-	PHP Hypertext Preprocessor
RAS	-	Resource Allocation Strategy
REST	-	Representational State Transfer
SaaS	-	Software as a Service
SLA	-	Service Level Agreement
SQL	-	Structured Query Language
UML	-	User-mode Linux
VCS	-	Version Control System
VM	-	Virtual Machine
VLAN	-	Virtual Local Area Network
XCP	-	Xen Cloud Platform
XDR	-	External Data Representation
XML	-	Extensible Markup Language

Chapter 1

1 Introduction

Cloud computing enables providing computing resources over an Internet connection to the users. In an IaaS (Infrastructure as a Service) Cloud, these computing resources include Processing power, Memory, Storage and Network resources. Cloud computing technologies including Infrastructure as a Service, Platform as a Service, and Software as a Service have changed the traditional "host on own data center" strategy and have given a good solution to prevent maintenance overhead of a private data center per organization. Medium-scale IaaS clouds are becoming more popular in small/medium scale organizations such as universities and enterprises. Medium-scale IaaS clouds are useful when organizations need to deploy their own private cloud using the compute, storage and network resources they have. There are several IaaS platforms such as OpenStack, CloudStack and Eucalyptus which users can use to deploy their own medium-scale clouds. These tools handle sharing and management of compute, storage and network resources dedicated to the cloud and perform resource allocation for various requirements.

1.1 Problem Statement

Many universities and enterprises are now setting up their own small-to-medium scale private clouds. Such private cloud are becoming popular, as they provide the ability to multiplex existing computing and storage resources within an organization while supporting diverse applications and platforms. It also provides better performance, control, and privacy. In a medium-scale cloud such as a university or enterprise cloud, there are different types of users including students, lecturers, internal/external researchers, and developers, who get benefits from the cloud in different ways. They may have varying requirements in terms of resources, priorities, and allocation periods for the resources. These different requirements may include processing intensive and memory intensive applications such as HPC (High Performance Computing) applications and data mining application, as well as labs which needs to be deployed on a specific set of hosts and for a particular period of time. Priority schemes and Dynamic VM (Virtual Machine) migration schemes should be used to satisfy all these requirements in an organized manner. However, currently known IaaS cloud platforms have no native capability to perform such dynamic resource allocations and VM preemption mechanisms. Therefore, it is important to extend existing cloud platforms to provide such policy, resource, and deadline aware VM scheduling.

1.2 Proposed Solution

We are proposing a resource scheduling framework which can be used as an extension to an existing IaaS cloud platform to support dynamic resource and policy aware VM scheduling. Resource scheduling algorithm schedules VMs while being aware of the capabilities of the cloud hosts and current resource usage, which is monitored continuously using a resource monitor. The resource scheduler will allocate resources according to predefined priority levels of a particular user who issued the resource request. Time-based scheduling (e.g., deploying labs for a particular period of time) is also performed while considering the priority levels and existing resource allocations. To provide these features we extend an existing IaaS cloud platform to include resource and policy aware VM creation, migration, and preemption.

1.3 Outline

In the rest of the document, we will discuss on our research topics, how we devise our solution to the problem that we described in the previous section and implementation details of the solution. This report consists of three main chapters. In Chapter 2, we discuss on the literature that we referred while researching the solution. In this chapter, we will further discuss on various cloud computing technologies, IaaS frameworks and hypervisors that we considered and the comparisons between them. Chapter 3 discusses on the design of our solution. We will describe our solution in terms of architectural diagrams and flowcharts. In Chapter 4, we will describe implementation details of the system including project management and version controlling. Performance analysis is presented in Chapter 5. Discussion and summary is presented in Chapter 6 and 7, respectively.

Chapter 2

2 Literature Survey

The proposed Resource Scheduler is for medium-scale IaaS (Infrastructure as a Service) cloud environments. Hence, in Section 2.1 we will first discuss about cloud computing, its variants and different cloud computing models. In Section 2.2 we describe different cloud computing Service Models. Section 2.3 describes the role of Hypervisors in cloud computing, different hypervisors and a comparison between them. Section 2.4 compares different IaaS frameworks used for private clouds including CloudStack, OpenStack and Eucalyptus. In Section 2.5, we compare OpenStack, CloudStack and Eucalyptus considering their features. Next, Section 2.6 describes public IaaS cloud platforms. In this section we describe Amazon EC2 platform which is one of the most popular public clouds. Further our solution requires the ability to effectively monitor resource utilization and availability in terms of CPU, Memory, Storage and Network usage. Hence, Resource Monitoring Systems are presented in Section 2.7. Section 2.8 describes several previous work in the areas of resource discovery, resource allocation and VM migration. Cloud services for HPC applications are described in Section 2.9. Resource description languages for resource discovery and allocation in Centralized Resource Scheduling Systems are presented in Section 2.10.

2.1 Cloud Computing

Cloud computing is a paradigm in which a large pool of systems are interconnected to provide scalable computing resources including computing capability and content storage in different ways such as application hosting, providing platform to develop applications and provide computing resources including processing power, memory, storage and network. Cloud computing provides an illusion to users of an unlimited resource pool in the cloud and migrate their internal datacenters to cloud datacenters. This paradigm brought a great evolution, as large organizations which had their own datacenters in the organization premises could migrate applications and data to the cloud getting rid of the cost of running datacenters and maintenance overhead. According to Dialogic [1], main characteristics of cloud computing can be described as:

- Shared Infrastructure:
  Cloud computing utilize virtualization to share physical infrastructure in the cloud and provides hosts in the cloud as VM's. Hypervisors are used to create virtual machines in the cloud while utilizing underlying physical infrastructure in an abstract manner.

- Dynamic Provisioning:

  Dynamic provisioning allows provisioning of resources based on the current demand requirements. A resource allocation request can be served by the cloud while considering current utilization of the cloud resources and provide the resource allocation in appropriate host(s) by dynamically taking decisions on where to allocate resources in the cloud.

- Network access:

  Resources that users allocate in the cloud is accessible over a network connection. These resources can be accessed by range of devices such as Mobile devices, PCs and various other devices over standard APIs based on protocols such as HTTP.

- Managed Monitoring

  Customers have to pay according to the resources they utilized, which is commonly called as "Pay as you go". Metering services in the Cloud continuously monitor customers' resource usage and calculate billing information according to the usage.

2.2 Cloud Computing Service Models and Cloud types

Cloud computing includes three different service models. These models are differentiated on how they provide service to the cloud users. They are:

- SaaS (Software-as-a-Service)

  SaaS provides software to the users over a network connection. SaaS subscribers can use software in the cloud without installing them in their own machines/servers (see Figure 2.2.1) e.g., Google Docs. SaaS subscribers are billed according to the functionalities they are subscribed to. When users ask for increased functionality, they are required to pay for the additional functionality being used.

- PaaS (Platform-as-a-Service)

  PaaS cloud model provides a development platform to the developers to deploy their own applications on the cloud e.g., Google App Engine. As Figure 2.2.1 illustrates, PaaS providers facilitates developers with necessary tools including Operating systems, Programming languages, Databases and guidelines to rapidly develop and deploy applications on the cloud.

Figure 2.1 – Service models of cloud computing [41].

- IaaS (Infrastructure-as-a-Service)

  Cloud subscribers are in control of the systems in terms of applications, storage and network connectivity but not the entire cloud infrastructure [1], e.g., Amazon EC2. IaaS cloud provides greater control to the subscribers and ability to use any operating system and application of their choice for the systems.

  Basic components of IaaS architecture are Compute, Storage and Network. IaaS cloud is responsible for provisioning resources to users from a resource pool containing Compute, Storage and Network resources. Figure 2.2.2 illustrates the internal architecture of a typical IaaS Cloud Computing model. According to Figure 2.2.2, basic components of an IaaS model are, Compute, Storage and Network. These three components are coordinated using virtualization techniques which creates an illusion of unlimited resource pool. Virtualization provides facility to run multiple operating systems under a single resource pool.

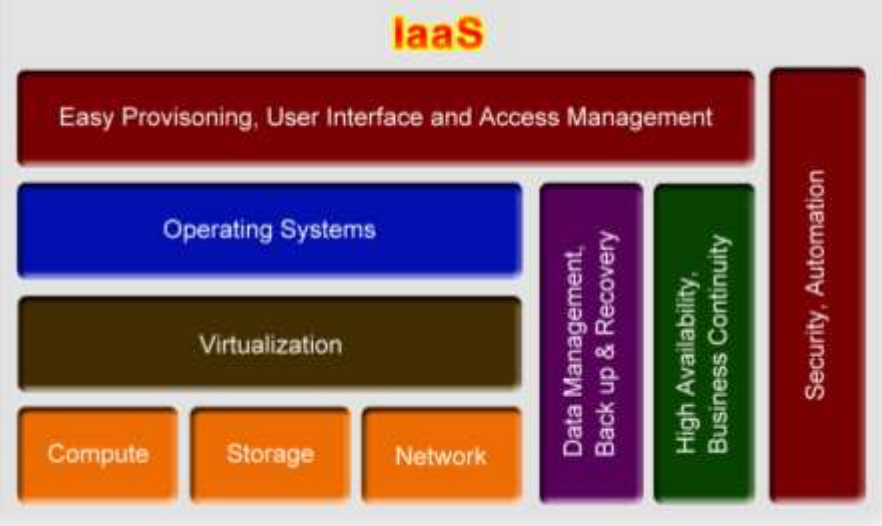

Figure 2.2 - Components of IaaS architecture [2].

Besides these cloud computing models, there are four types of cloud deployment models [1]:

- Private cloud:

  In a private cloud, cloud infrastructure is deployed, maintained and operated by the organization who owns the private cloud. Cloud infrastructure is used by the organization itself but not by external parties.

- Community cloud:

  In this deployment model, cloud infrastructure is being managed by a group of organizations [1]. Community cloud is located either in-house or at a third-party premises.

- Public cloud:

  In a public cloud, services are open to the users over an internet connection. Service of the cloud may either be free or based on Pay-as-you-go method. Public cloud provides subscribers ability to benefit from cloud services with lesser cost than any other cloud deployment models

- Hybrid cloud:

  In Hybrid Cloud deployment model (Public/Private, etc.), multiple types of clouds are interconnected together. Applications can be migrated from one cloud to another cloud.

According to our solution proposed for the complex resource allocation, we are proposing a resource scheduler as an extension to an existing IaaS framework. Next, we will describe about different IaaS frameworks which can be used and compare them in terms of their features and applicability to our solution.

2.3 Hypervisors

The term "virtualization" came into existence in 1960's where mainframe computers are logically divided to run applications concurrently. So currently virtualization is used for cloud computing where it will provide the capability of running few virtual machines in a single physical server where these virtual machines contain different operating systems and hardware specifications.

Hypervisor is the piece of software which enables virtualization. This is the software layer which is capable of running guest virtual machines within a virtual environment where these environment may or may not differ with other guests. There are two types of hypervisors which are classified as "Type 1" and "Type 2".

"Type 1" hypervisors run directly on top of the hardware where this is also called bare-metal hypervisor. Such hypervisors are responsible for allocating resources such as storage and networking. "Type 2" hypervisors will need a host operating system in order to run the software where it will create guest virtual machines. "Type 1" hypervisors have a performance edge over "Type 2" hypervisors because of access to direct hardware. However, "Type 1" may have device driver issues because there is no operating system to deal with hardware.

2.3.1 Xen Cloud Platform (XCP)

Xen is a "Type 1" hypervisor. The hypervisor technology uses a technique called paravirtualization [3]. Paravirtualization allows VM's to access the hardware using a software interface where it will access storage and memory through an interface but will not directly deal with hardware in the physical node where hypervisor resides. This technique helps Xen to work in architectures where it does not support hardware assisted virtualization. Xen is controlled by a guest VM which is also known as Dom0. Dom0 runs on the hypervisor and it has special privileges where it will control all other unprivileged guest VM's where they are identified as DomU's [4].

Figure 2.3.1 illustrates the typical architecture of XEN. Xen hypervisor runs directly on top of hardware and Dom0 works as the VM management server.

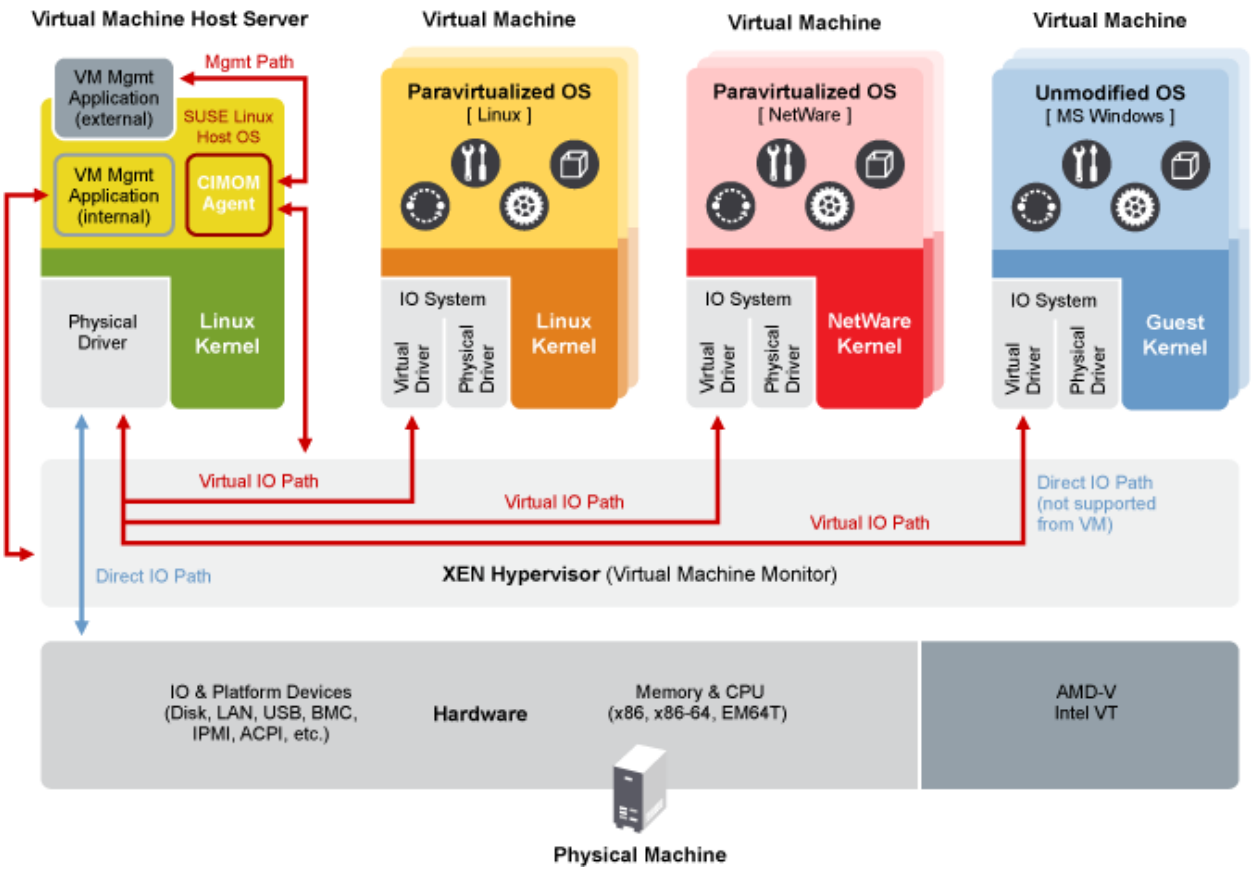

Figure 2.3- Xen hypervisor architecture [35].

When DomU's need access to hardware instead of directly dealing with the hardware, they call the hypervisor. This methodology takes away the overhead of emulating hardware. This technique is called paravirtualization and this enables to run the Xen hypervisor on hardware where it does not support full virtualization [5].

2.3.2 KVM

KVM stands for kernel-based virtual machine [6]. KVM uses the Linux kernel and changes into a loadable module such that it turns into a bare-metal hypervisor. KVM was developed once the hardware assisted virtualization because available. Therefore, hypervisor does not have to support legacy hardware systems. This made the hypervisor more powerful and lightweight because it was using the already implemented features by the hardware itself.

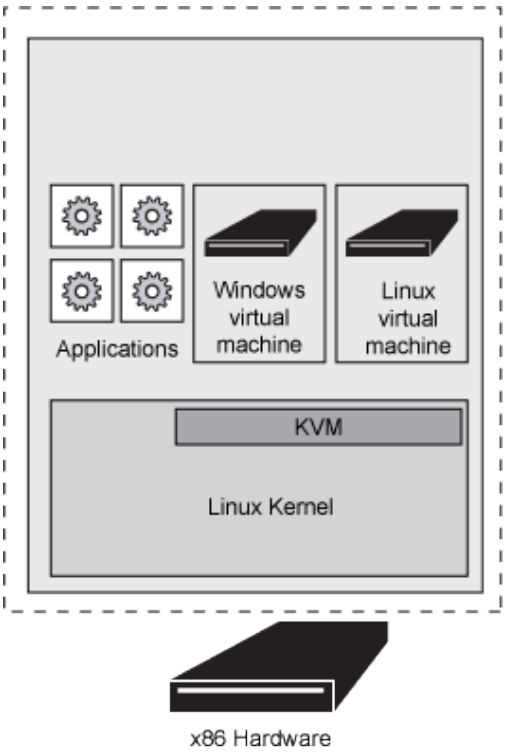

Figure 2.4 - KVM high-level architecture [36].

KVM identifies guest VM's as another process in the operating system. A virtual CPU is a Linux process and this architecture utilizes predefined Linux kernel features. Security is being preserved by the standard Linux security model. The traditional security model brings isolation of the virtual machines and resource controls. Figure 2.3.2 present the Linux kernel which is KVM and then at the top there are two VM's which run Windows and Linux as their operating systems.

KVM is part of Linux so that any hardware which is supported by Linux can be used by KVM. When new features are added into the Linux kernel, KVM will inherit these changes which enhances the flexibility of the hypervisor. KVM is supporting live migration where VM's could change their physical hosts while there are running application programs. Also KVM supports snapshots feature where it allows the user to save the state of the virtual machine in a disk so which could be restored at any point after relocation of the VM. KVM supports hybrid virtualization where paravirtualized drivers are installed in the guest VM's. This imposes a performance edge in I/O interfacing. Figure 2.3.3 shows the internal structure of KVM where the component libvirt act as the API which manages the virtualization.

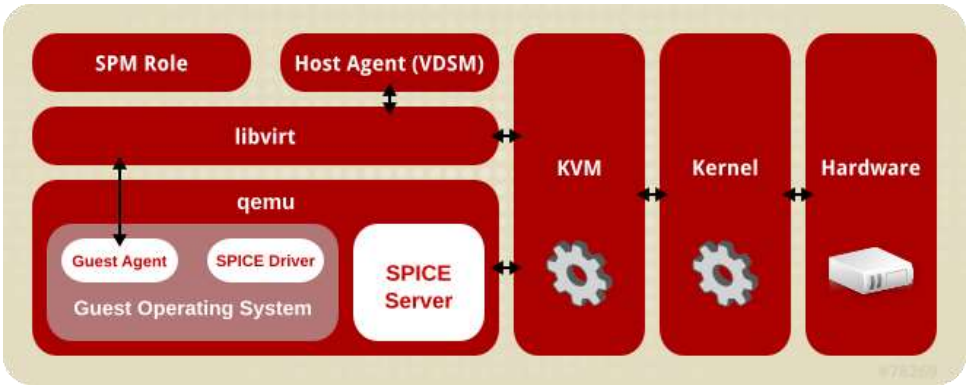

Figure 2.5 - How KVM works [37].

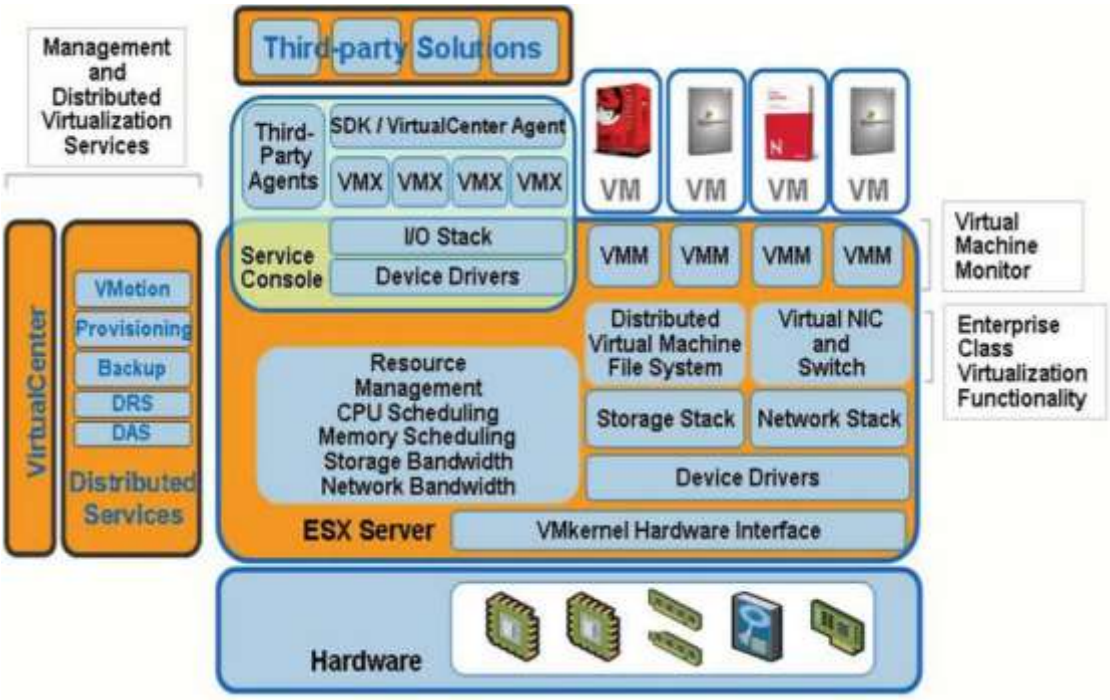

Figure 2.6 - VMware Hypervisor Architecture [38].

2.3.3 VMware

VMware is a commercial hypervisor which has many features compared to other open source hypervisor such as KVM and Xen [7]. Figure 2.3.4 outlines an architecture diagram of VMware ESX. VMware works as a bare-metal hypervisor where it runs directly on hardware. It has its own kernel which is called VM kernel component. The VM kernel works as the first virtual machine and becomes the interface for hardware operations. Each virtual machine represents a complete system—with processors, memory, networking,

storage and BIOS - so that an operating system and software applications will run concurrently with isolation. VMware provides live migration just as KVM and other features such as power management [8].

2.3.4 Comparison of Hypervisors

Table 2.1- Comparison among hypervisors.

Characteristic	Xen Cloud Platform	VMware	KVM
Hypervisor Type	Type 1	Type 2	Type 2 and Type 1
Supports Para virtualization	Yes	Yes	Yes
Memory-Disk Snapshot	No	Yes	Yes
Commercial or Open source	Open source	Commercial	Open Source

When considering the objectives of the project, the hypervisor should support Memory-disk snapshots [9]. The reason is there are situations where we have to preempt a job when high-priority requests gets submitted into our system. For a successful preemption we need to transfer the memory state and the saved files in the virtual machine to the snapshot so when restarting the VM it will not lose any information. VMware does support the feature but it is a commercial product and Xen cloud platform which is the open source version does not support this feature. But KVM is an open source hypervisor which supports memory-disk snapshots which suits our requirement.

2.4 IaaS Cloud Platforms for Private Clouds

This section describes three IaaS platforms developed to deploy IaaS clouds and which were considered to be applicable for our solution. We describe three IaaS frameworks, OpenStack, Eucalyptus and CloudStack.

2.4.1 OpenStack

OpenStack [10] is a free and open source IaaS cloud platform which was initiated in 2010 as a joint project by Rackspace [11] and NASA [12]. OpenStack is a collection of a series of projects which collaboratively build the composition of OpenStack architecture providing authentication services, metering services, database services and infrastructure including compute, network and storage. Figure 2.4.1 shows the components of the OpenStack architecture. Table 2.4.1 describes each component illustrated in Figure 2.4.1.

OpenStack contains nine different projects where a single project can be hosted in a single node where it can be identified as highly structured and scalable. For example, Keystone Project where it take cares of authentication can be scaled into number of physical nodes if the number of users who are using the system gets higher. This high scalability adds to the flexibility of the product but it becomes tough to configure the environment. OpenStack is suitable for a medium to large-scale cloud where it could provide the physical nodes to host the whole system but in a small to medium environment spending resources to host the cloud framework is not a good engineering decision.

Table 2.2 - Components of OpenStack.

Component	**Description**
Horizon	OpenStack Horizon is the administrative dashboard of OpenStack (Figure 2.4.1). Horizon provides a front-end for OpenStack administrative APIs.
Nova	OpenStack Nova provides Compute capability to OpenStack cloud. Nova communicates with the hypervisor via a virtualization API to create virtual machines. OpenStack currently supports hypervisors including KVM, Xen (Xen Server and Xen cloud platform)
Neutron	Neutron provides networking-as-a-service capabilities to OpenStack cloud. Neutron provides OpenStack users to define networks on VMs via Neutron API (Figure 2.4.1).
Swift	Stores and retrieves arbitrary unstructured data objects via a RESTful, HTTP based API [1]. Swift also provides fault tolerance by means of Data replication.
Cinder	Cinder provides block storage for OpenStack. Cinder can be used to create block storage devices as secondary storage for VMs created by neutron.
Keystone	Keystone is the component in OpenStack which provides authentication and authorization services for OpenStack. This provides identity service for other OpenStack components via OpenStack Identity API (Figure 2.4.1).
Glance	Glance is the image storage for OpenStack based IaaS clouds. Glance stores VM images and these VM images can be streamed over a REST API by Nova for VM instance provisioning.

Ceilometer	Ceilometer provides metering services for OpenStack. Main tasks of Ceilometer are billing, benchmarking, scalability and statistics [1]
Heat	Heat provides the orchestration for an OpenStack cloud
Trove	Trove provides a database-as-a-service in a scalable and reliable manner.

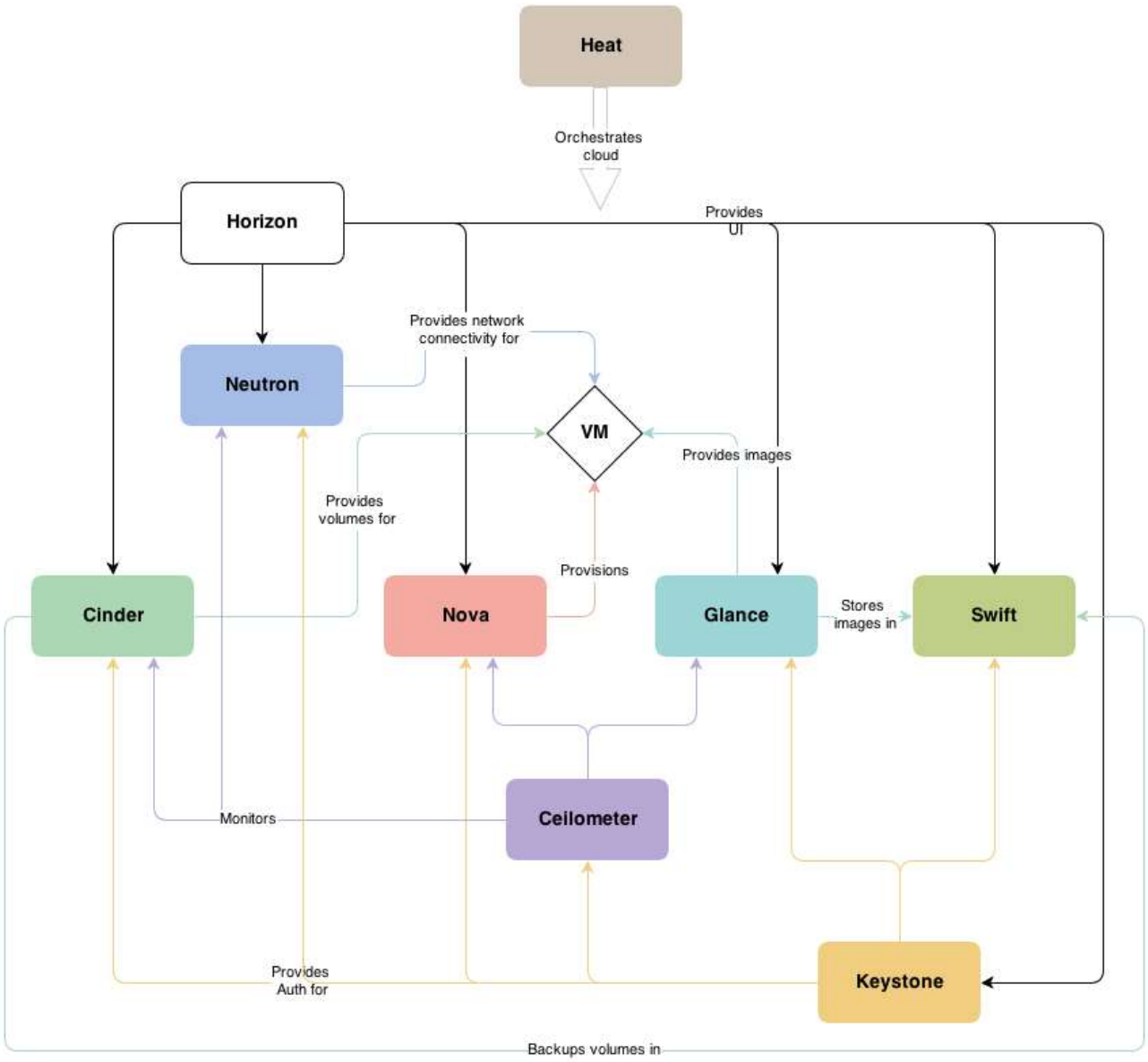

Figure 2.7- OpenStack Conceptual Architecture [13].

2.4.2 Eucalyptus

Eucalyptus is a cloud platform with a modular architecture which is open to experimental instrumentation and study [16]. One advantage of Eucalyptus over OpenStack is, Eucalyptus' external interface API is based on Amazon EC2 API [14] which enables Eucalyptus to interact with the same tools which are used for Amazon EC2. The basic components of Eucalyptus are listed in Table 2.4.2.

Table 2.3 - Components of Eucalyptus.

Component	**Description**
Node Controller	Each physical host in the Eucalyptus cloud runs an instance of Node Controller. Node controller in each host is responsible for initializing and management of VMs on the host.
Cluster Controller	Physical hosts are grouped into clusters and the Cluster controller is the main controller of a Cluster. It gathers information about the VM deployments and schedules VM deployments. And also Cluster controller is in charge of managing the private network of hosts within a cluster.
Storage Controller	Storage controller is a storage service which implements Amazon S3 interface which can be used to store and retrieve VM images and user data [16]
Cloud Controller	Cloud Controller is the central controller which orchestrates entire Eucalyptus cloud and provide and external interface to administrators to manage cloud infrastructure. It receives commands from external Amazon EC2 API and uses Cloud controllers to execute actions while taking high level decisions on resource scheduling.

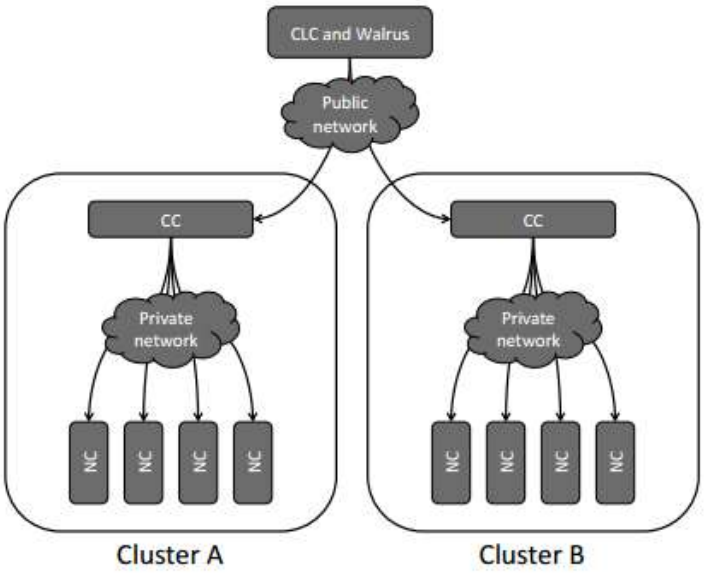

Figure 2.8 - Eucalyptus Architecture [16].

One of the Eucalyptus' great features are its compatibility Amazon's EC2 API and S3 API for storage [16]. It has a relatively simple architecture than OpenStack but modular architecture compared to cloud platforms such as CloudStack which will be described in the next section. Since Eucalyptus was developed for research purposes and a novel framework compared to other IaaS platforms it lacks community support. Although it support variety of hypervisors including Xen hypervisor, KVM and VMware, its current developing state has caused it not been widely adopted in medium-scale Cloud deployments.

OpenStack and Eucalyptus have their own advantages and drawbacks which affect how they can be integrated into our solution. Although OpenStack is much more scalable compared to others and supports vast number of hypervisors including KVM, Xen, VMware vSphere, LXC, UML and Microsoft HyperV [17], it is not suitable for small to medium-scale clouds such as University/Enterprise clouds due to intensive deployment effort and complexity. Although Eucalyptus implements Amazon EC2 API as the external interface, eucalyptus is not widely used due to its development state.

2.4.3 CloudStack

CloudStack [18] is an open-source IaaS platform which offers establishing on-demand elastic cloud computing services and end-user provisioning of resources. CloudStack works with a variety of hypervisors such as XenServer, KVM and Hyper-V; hence, allows multiple hypervisor implementations within one cloud. Scalability of CloudStack allows it to have thousands of physical servers which can be geographically distributed and successfully manages them via the management server which scales accordingly. Automatic Configuration Management is another functionality which CloudStack provides. When a virtual machine is deployed, CloudStack automatically sets up the network and storage settings for them. Other than that, CloudStack is identified for its user friendly Graphical User Interfaces which are there for both administrators and end-users. CloudStack also provides a REST-like API which allows Operation, Management and Utilization of cloud. As Amazon EC2 is the most common cloud provider, and there are many tools developed for it, CloudStack offers an EC2 API translation layer which allows using common EC2 tools in the CloudStack Environment. CloudStack provides multi-node installation of management server and several other features to increase availability of the system as High-Availability is greatly considered by CloudStack.

Diagram below shows the basic deployment architecture of CloudStack.

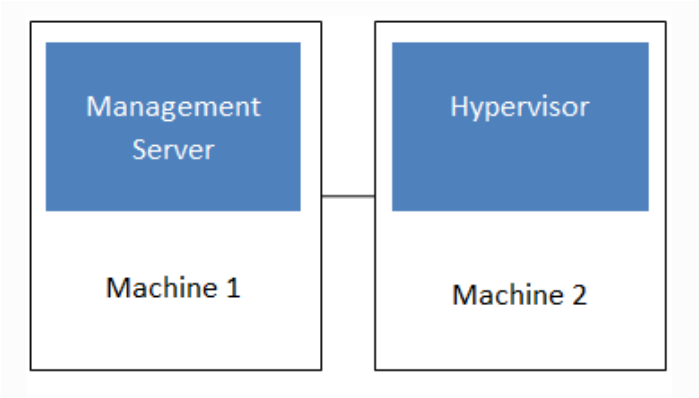

Figure 2.9 - Basic Deployment Architecture of CloudStack [18].

As shown in Figure 2.4.3, a basic CloudStack deployment consists of a management server and the resources it manages such as a set of hypervisors, storage devices, VLANs and IP address blocks. At the minimum installation both the management server and hypervisor can run on the same node. A full featured installation consists of thousands of machines, several networking technologies and availability increasing features like multi-node management server installation.

To ease the management of resources, they are categorized into several units. Figure 2.4.4 shows the cloud infrastructure of a CloudStack deployment.

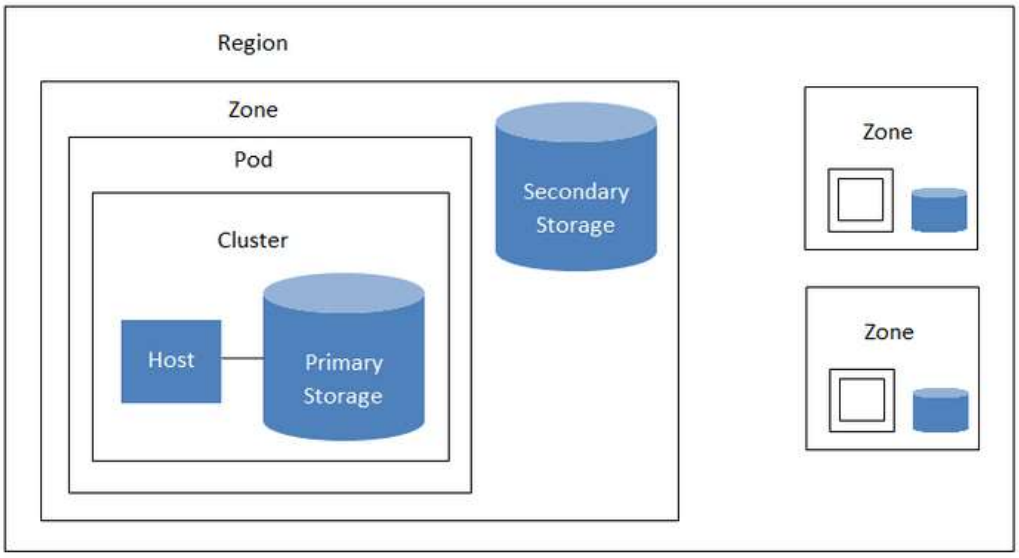

Figure 2.10 - Cloud Infrastructure [18].

Cloud Infrastructure has seven different components: Region, Zone, Pod, Cluster, Host, Primary Storage and Secondary Storage. They are described below.

- A region is the largest unit within a CloudStack deployment which is made up from several zones. Each region is managed through its particular cluster of management servers. Through regions CloudStack has increased its availability as if one region is unavailable the deployment can be done in the other.
- A zone is the second largest unit and it roughly equivalent to a datacenter. The advantage of having a zone is it provides physical isolation and redundancy. Each zone can have its own power supply and a network uplink. Zone consists of several pods.
- A pod is a single rack. Hosts in a pod belongs to a single subnet.
- A group of hosts is categorized as a cluster. Cluster is the place where VM migration is possible within its hosts. A cluster can consists of one or more hosts and primary storage servers. Hosts in a cluster has the same hardware, same hypervisor deployed, belong to the same subnet and share the same primary storage.
- A host is the smallest unit and consists of a single machine.
- Primary storage stores the disk volumes of all VMs running in a host. It is associated to a cluster or a zone.
- Secondary storage is for a zone or a region. It stores templates, ISO images and disk volume snapshots and these items are available for all hosts.

Main component of a CloudStack cloud is Management Server. CloudStack API provides a programmable interface to manage CloudStack Cloud through CloudStack Management Server (see Figure 2.4.5). Accessing CloudStack API can be done using CloudStack rich Web UI, Command Line Interface using CloudMonkey or third-party tools written for CloudStack API.

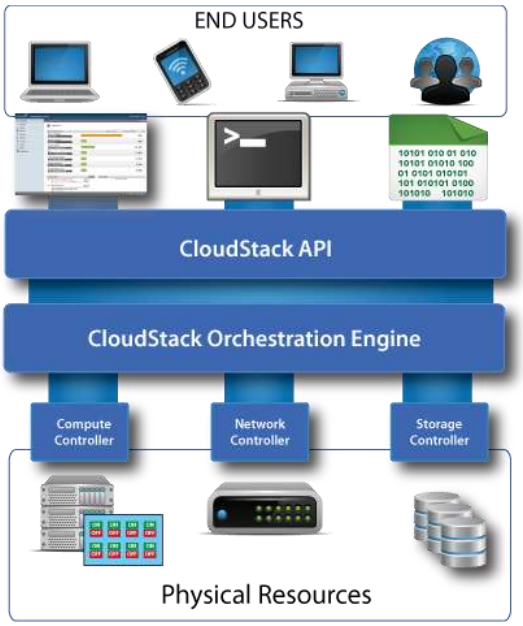

Figure 2.11 - CloudStack API [42].

Compared to IaaS platforms like OpenStack, CloudStack has a simple Architecture so that the development with CloudStack is easier. Even the Deployment Architecture is very simple compared to OpenStack.

2.5 Comparison of IaaS Platforms

Daniel Kranowski has presented [33] a comparison among these three IaaS. Table 2.5.1 compares OpenStack, Eucalyptus and CloudStack based on the characteristics such as architecture, installation difficulty, administrative access, security features and availability.

Table 2.4 - Comparison of IaaS platforms.

Characteristics	OpenStack	Eucalyptus	CloudStack
Architecture	Highly distributed	Similar to AWS/ 5 Components	Monolithic architecture
Supported Hypervisors	Native support for XenServer/XCP and KVM, VMware can be configured to supported through web services	Support all major hypervisors including XenServer/XCP, VMware, QEMU, KVM etc.	Support all major hypervisors including XenServer/XCP, VMware, QEMU, KVM etc.
Installation	Difficult and required expertise	Moderate expertise required	Moderate expertise required
Administration	Web UI, euca2ools for Amazon EC2 API, CLI	Amazon EC2 compatible CLI/Limited User Interface	Good interface/CloudStack API/ Amazon EC2 compatible interface
Security	Baseline security, authentication and authorization service of Keystone	Baseline security (VLAN/Firewall VM protection)	Baseline security
Scalability	Highly scalable	Less scalable compare to CloudStack and OpenStack	Highly scalable, but less scalable than OpenStack
Availability	Primary-secondary component failover	Swift rings [19]	Load balancing/Multi-node controller
Community Support	Large community	Smaller than in OpenStack	Smaller community than OpenStack and CloudStack

Compared to OpenStack and Eucalyptus, CloudStack has a relatively simple architecture and easier installation compared to other two which is more suitable for medium-scale cloud deployments. Installation of CloudStack Management Server installs all the components that are required to deploy an IaaS Cloud. In contrast, OpenStack requires all components separately (Nova, Neutron, Swift, Keystone, etc.) to be installed and configured before starting up the cloud service. CloudStack also provides an easy to use web user interface, API and an AWS EC2 compatible API for administrative purposes. When the installation and configuration complexity in OpenStack and Eucalyptus are considered, using them for a medium-scale University/Enterprise clouds is difficult. For this reason, we have chosen CloudStack as the IaaS platform to implement our solution.

2.6 IaaS Public Cloud Platforms

2.6.1 Amazon EC2

Amazon Elastic Compute Cloud (EC2) is a public cloud provider where it supports scalable cloud computing [14]. This is important for small-scale companies where they do not have to spend large enough capital for hardware resources. When they need more hardware resources they could scale up easily using Amazon EC2. Amazon is one of the large cloud providers in current and due to its popularity there are large number of tools implemented using Amazon EC2 API. Due to this reason, most IaaS cloud platforms including OpenStack, CloudStack and Eucalyptus also provides interfaces which implements EC2 API so that tools that are implemented to EC2 can also be used with them. Figure 2.6.1 is the architecture of Amazon EC2.

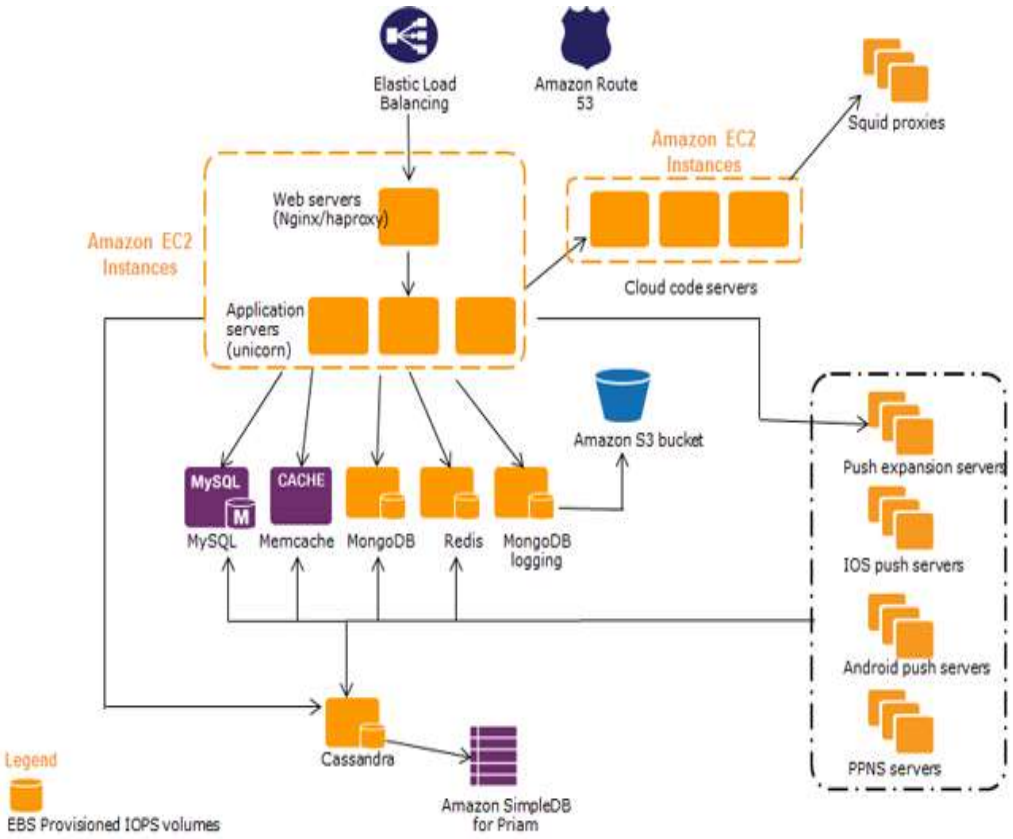

Figure 2.12 - Amazon EC2 Architecture [39].

Table 2.6.1 Types of EC2 instances [15].

Table 2.5 - EC2 Instances.

Instance Family	Current Generation Instance Types
General purpose	t2.micro | t2.small | t2.medium | m3.medium | m3.large | m3.xlarge | m3.2xlarge
Compute optimized	c3.large | c3.xlarge | c3.2xlarge | c3.4xlarge | c3.8xlarge
Memory optimized	r3.large | r3.xlarge | r3.2xlarge | r3.4xlarge | r3.8xlarge
Storage optimized	i2.xlarge | i2.2xlarge | i2.4xlarge | i2.8xlarge | hs1.8xlarge
GPU instances	g2.2xlarge

EC2 Instances are virtualized [15]. So when a physical host fail the VM is migrated to another physical host so the user will not lose information.

2.7 Resource Monitoring in Cloud

Before understanding what is resource monitoring, it is important to understand what resources are. Resources can be identified as CPU, Bandwidth, Memory and Storage. “At its simplest, resource monitoring is nothing more than obtaining information concerning the utilization of one or more system resources” [20]. Resource monitoring is important in many contexts of system administration because resources affect to the system performance such that careful monitoring of resources is needed.

In our project there is a requirement to monitor VM’s and also to discover resources in the cloud. Discovering resources in the cloud environment can be also done by using the Hypervisor API’s. But then our solution will limit to a specific hypervisor so we use a resource monitoring tool to monitor the cloud resource pool where we our solution will not depend on the hypervisor which is used by CloudStack.

2.7.1 Ganglia

Ganglia [21] is a resource monitoring system which allows monitoring high performance computing systems such as clusters and grids. For its data representation it uses XML, XDR (External Data Representation) for data transport and RRDtool for data storage and virtualization [22]. Ganglia considers on decreasing resource utilization overhead on hosts and high concurrency thus, provides various data structures and algorithms to achieve them [22].

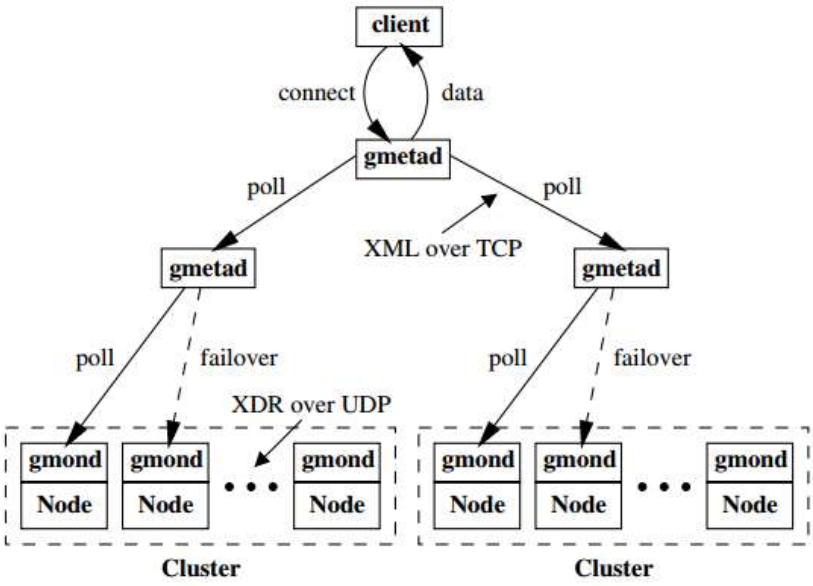

Figure 2.13 - Ganglia Architecture [22].

Ganglia relies on a multicast-based listen/announce protocol to monitor state within clusters [22]. To check the availability of resource it uses heartbeat messages on a well-known multicast address within a cluster. Each node monitors its local resources and multicast packets including monitoring data are sent to the multicast address mentioned above. Ganglia monitors each packet and federates multiple clusters together using a tree of point-to-point connections and aggregates their state.

The implementation of Ganglia consists of *gmond* (Ganglia Monitoring Daemon), *gmetad* (Ganglia Meta Daemon), *gmetric* and a client side library. *gmond* provides monitoring of a single cluster and *gmetd* provides federation of multiple clusters. *gmetric* is a command line program which can be used by applications to publish application-specific metrics and the Client side library provides access for specific Ganglia features.

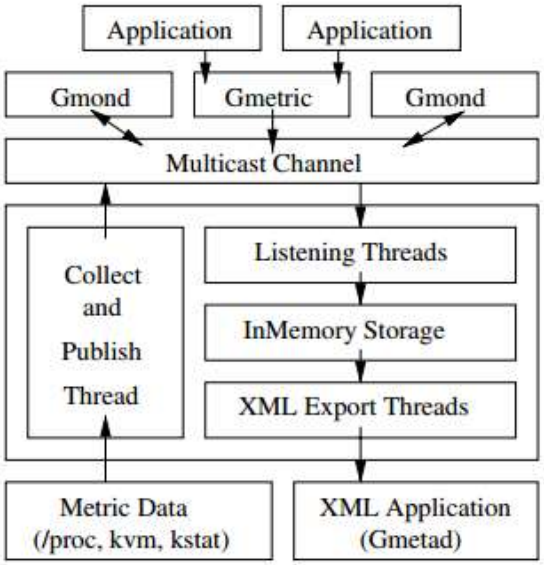

Figure 2.14 - Ganglia Implementation [22].

2.7.2 Nagios

Nagios is a resource monitoring tool where it helps to keep track of applications, services and infrastructure components. Nagios works with steps where it includes monitoring, alerting, response, reporting, maintenance and planning [23].

As a typical resource monitoring tool Nagios monitors resources such as network components, servers and other services. When a critical component fails it sends alerts to the system administrators so they could work on with the issue. These alerts could be grouped by the severity and then administrators could chose to work on to solve them. Reporting is also carried out by Nagios where it would be beneficial to check the availability of a particular resource and to assess SLA's. There is also a planning section where it specifies the necessary upgrades in an infrastructure where it predicts improvements.

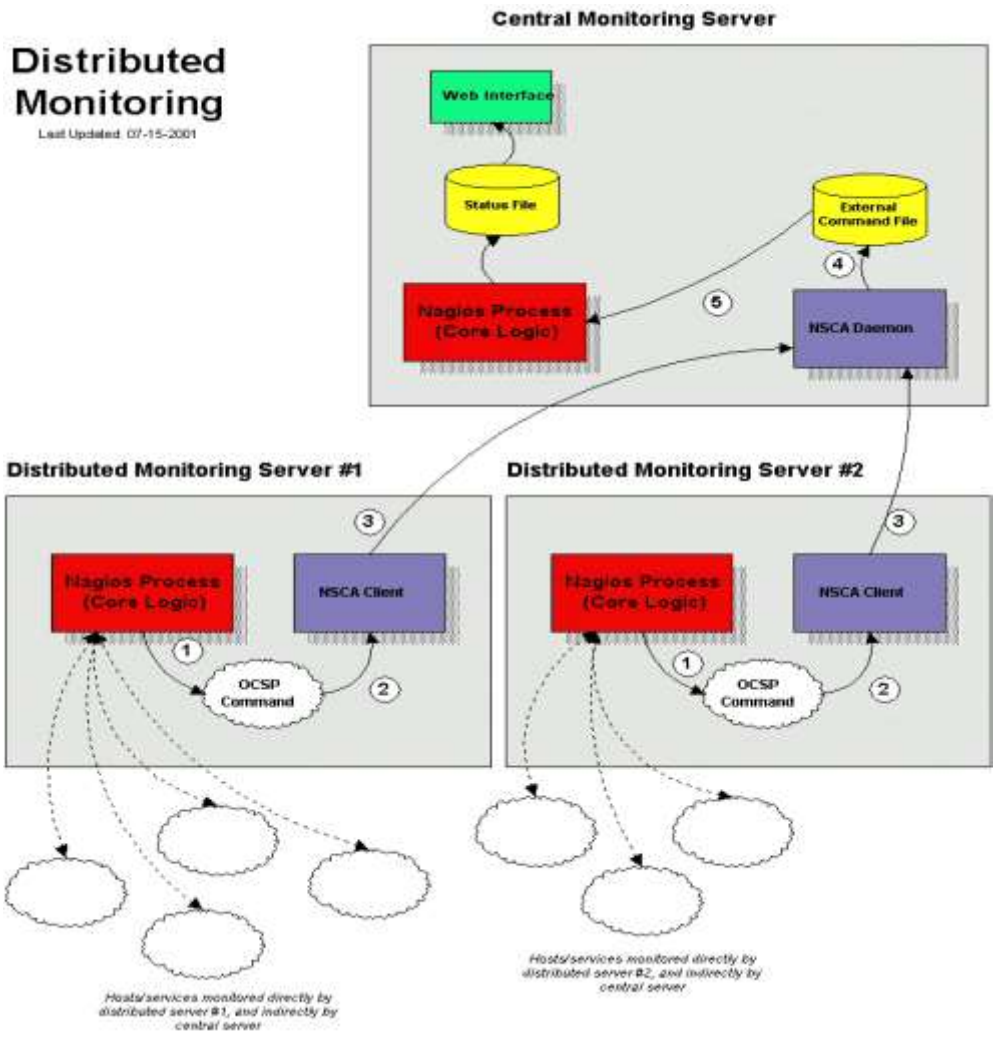

Figure 2.15 - Nagios Architecture [40].

Figure 2.7.3 presents the Nagios architecture. There are distributed monitoring servers will monitor services or resource pools and these servers will be monitored by a central server which keeps track of all the distributed servers.

Nagios periodically checks the nodes in the infrastructure. There are three states which can be identified as

- Critical
- Unknown
- Warning

When these states change notifications are sent so the administrators know that there is an emergency and they should react.

2.7.3 Zabbix

Zabbix is a free and open source resource monitoring tool that can be used to monitor resource utilization in hosts in a Datacenter. Its backend is developed using C and provides a PHP-based frontend for easy monitoring and configuration as well as a programmable interface for third-party tools integration. Zabbix includes an agent which can be installed on hosts in a Datacenter which needs to be monitored (Figure 2.7.4). Zabbix agents run in background on the hosts and provides monitored information about CPU, memory, storage, network utilization, etc., and provides an interface through which Zabbix server can access and collect those information. Zabbix server stores fetched information from Zabbix agents in a database (Figure 2.7.4). It supports number of databases including MySQL, PostgreSQL, SQLite, Oracle and IBM DB2.

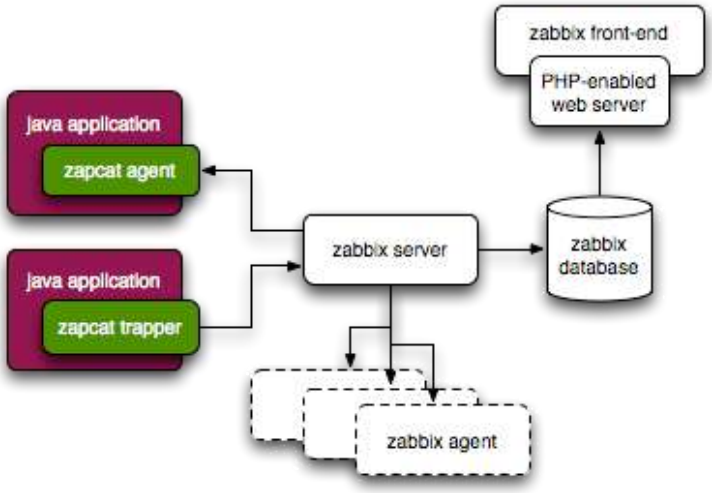

Figure 2.16 - Zabbix Architecture [43].

Figure 2.17 - Monitoring in Zabbix: CPU load (all processors).

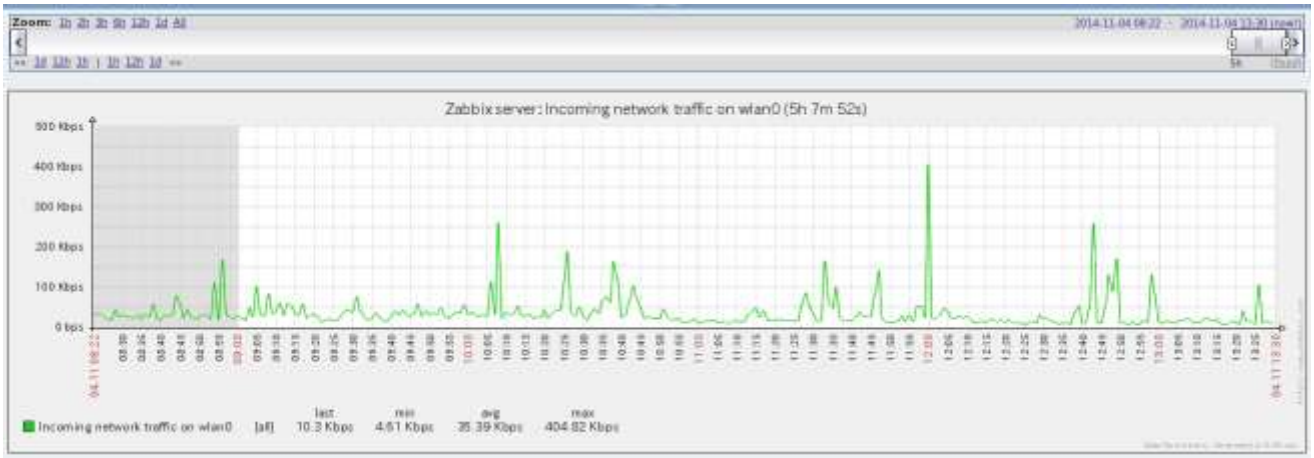

Figure 2.18 - Monitoring in Zabbix: Network utilization on wlan0 interface.

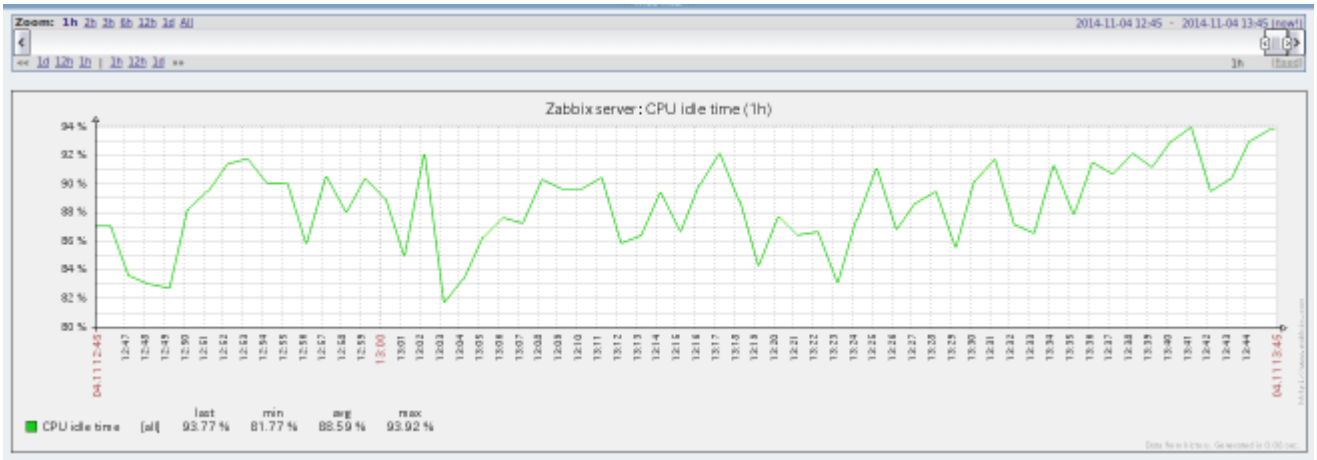

Figure 2.19 - Monitoring in Zabbix: CPU Idle time.

2.7.4 Comparison of Resource Monitoring Systems

According to the above described features Ganglia, Nagios and Zabbix are compared in Table 2.7.1.

Table 2.6 - Comparison among different resource monitoring tools.

Characteristics	Ganglia	Nagios	Zabbix
Type of statistics	Monitoring and trending statistics	Monitoring and health checking and statistics and alerting	Monitoring and health checking and statistics and alerting
Frequency of statistics collection	Can be configured	Can be configured	Can be configured
Flapping detection	No flapping detection	Yes	No flapping detection
Data storage	RRDtool	Flat files (relational database is not required)	MySQL database
Scalability	Highly scalable	Highly scalable	Highly scalable. Up to 100,000 monitored devices and 100,000 metrics [24]
UI	Read only Web UI	Read only interface	Interface provides full control
Commercial	Free	Yes. Only limited functionality is available free.	Free and Open Source

We have chosen Zabbix as the resource monitoring software because it is free. And it provides an easy to use REST API for third-party tools integration. Zabbix is also much easy to install and configure. Although Zabbix does not provide flapping detection while Nagios does, our purpose of using a resource monitor does not require flapping detection since we use average resource utilization over a period.

2.8 Resource Discovery and Scheduling

Cloud computing is a paradigm which supports vertical scaling of resources. When a demand of an application goes high user requests for more resources in a way it scales vertically but not horizontally. But still these resources may or may not use few or more physical nodes. Computational resources are scarce and should be handled appropriately. This is why resource allocation to virtual machines in an IaaS environment plays a major role and also a difficult problem to solve depending on the variety of the applications and the context.

"Resource Allocation Strategy (RAS) is all about integrating cloud provider activities for utilizing and allocating scarce resources within the limit of cloud environment so as to meet the needs of the cloud application." [25]

An optimal Resource Allocation strategy should avoid the following criteria

- Resource Contention: Two or more applications trying to access the same resource at the same time.
- Scarcity of Resources: Resources are limited such that a situation arise where there are no resources to allocate to an application.
- Resource Fragmentation: Resources are fragmented that it would not facilitate another application.
- Over Provisioning: Resources are allocated to an application where the resource demand is higher than actual usage.
- Lower Provisioning: Allocated resources are to the application is low than the actual usage of the application.

Resource allocation strategies can be performed in different methods. One such method is execution time of an application [25] which is erroneous because it is not easy to predict a time that an application finishes its execution. Using different policies user groups is another way of implementing strategies. Defining different user groups and prioritizing them while allocating resources, users having different authorization levels such as access to share file systems can be two perfect examples.

When we found a strategy of allocating resources we move on to find two important tools which are resource discoverer and resource allocator. Dialogic [26] Resource discoverer will be checking for resources where to fulfill a request of a virtual machine. Discoverer will scan through nodes and try to find a node with the attributes that the user will provide which is a criteria and a specification of the requirements that user seeks from his virtual machine.

```
<group>
 <name>Group1</name>
 <num_machines>10</num_machines>
 <oneminload>0.0,0.0,0.0,2.0,0.01</oneminload>
 <disk_free>0.1,0.2,max,max,0.005</disk_free>
 <latency>0.0,0.0,20.0,30.0,0.5</latency>
 <os_name>
  <value>Linux,  0.0</value>
 </os_name>
 <gnp>
  <value>0.0/0.0/0.0;50.0,  0.0</value>
 </gnp>
</group>
```

This is a typical structure of a request that user sends to the SWORD resource allocation tool which is used by PlanetLab [27]. So the user requests set of machines, which are inside a group which consists of 10 machines and some values of CPU load, network latency, and operating system. This is a simple query that user submits and the resource discover will answer whether or not it could provide the resources that user seeks.

There are different management objectives that cloud provider will seek when allocating resources. But optimize few or more objectives at the same time is an open research area [28]. There are frameworks where tries to minimize the power consumed by a cloud datacenter where it moves virtual machines to different physical servers. Load of the virtual servers' changes time to time such that they can be moved to different physical servers and shut down servers in order to achieve the management objective of the cost. But sometimes it may not be the ideal way of doing things if there are policy based models. Some user may want a network latency as such switching off a particular server will violate the policy.

We may or not be able to fulfill every requirement such that we need to preempt jobs. The states of the work should be saved and high priority jobs will take the resources and other lower priority jobs will wait till their return or till the system have resource free. So there are issues to solve such as when to release resources and when to start a job again which was preempted, how to prioritize jobs what policies that we could use, how to secure the data where a job is preempted in the middle of a computation.

Dealing with different constraints is the research problem that we have to solve in the resource allocation. Any system with a distributed architecture which has less resources to manage usually encounters this problem of allocating resources. Trying to find a more generic solution for the resource allocation is our main goal of the project where users could satisfy with the policies and also with the performance of their virtual machines which are dedicated to them in a profound manner.

In the following section we present a selected subset of related work on resource scheduling. This includes a VM consolidation framework for CloudStack and a product called VMware DRS and also several resource scheduling algorithms.

2.8.1 VM Consolidation Framework for CloudStack

Cloud computing based applications are emerging among Internet users and large-scale virtualized datacenters are established to meet with the increasing user demand. This demand produce high traffic volumes thus, the number of servers in cloud datacenters are increased. These additional computational power consume tremendous amounts of electricity but, this energy can be underutilized or sprawled. Virtual Machine Consolidation Framework [29] proposes an energy saving solution where the virtual machines can be migrated to the most appropriate place. That is VMs can be consolidate and de-consolidate as to balance CPU utilization among physical machines. VM consolidation framework proposed in paper [29] describes how VM consolidation should be performed in order to save power in a datacenter, and how VM deconsolidation should be done in order to balance load among datacenter servers. Architecture of this framework is a modular system consists of several modules such as, resource monitoring systems, power control mechanisms and VM packing algorithms.

As we know, a cloud system consists of number of physical servers and multiple virtual machines can be run on those servers. According to the resource demands these VMs should be allocated in to the minimum physical servers as possible avoiding underutilization and sprawl. For example, an underutilized server can be switched to sleep mode or can be turned off after it successfully migrates its VMs in to appropriate servers. VM consolidation framework [29] handles this merging and reallocation. Proposed consolidation framework in this paper [29] is an extension for CloudStack IaaS platform, allowing not only consolidation but also deconsolidation of VMs. De-consolidation is needed to avoid performance degradation due to overload.

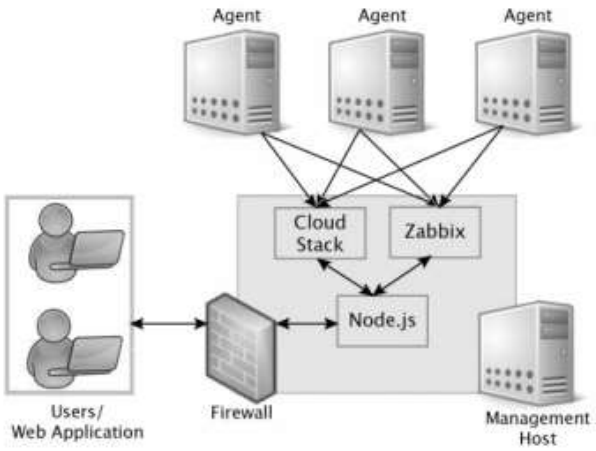

Figure 2.20 - VM Consolidation Framework Architecture [29].

As the Figure 2.8.1 shows this VM consolidation system combines three technologies, CloudStack, Node.js and Zabbix. System consists of controllers and agents. Each controller runs on a management host and each agent runs on a computing host. Main component of the framework is CloudStack platform. Live migration of VMs is achieved through CloudStack REST API. The policy for consolidation should consider CPU status, memory, storage, power consumption and temperature. All these information can be obtained by Zabbix, an open-source software which offers efficient resource and performance monitoring. As the main component of the Development Environment Node.js is chosen and therefore the entire framework is written in JavaScript language.

Proposed VM Consolidation Framework [29] consists of four main components. They are listed below with their functionalities:

1. CloudStack Operator - Interacts with CloudStack API and Enable utilization of CloudStack Services.

2. Monitoring Operator - Monitoring and Data Collection via Zabbix.

3. Mapping Operator - Switching hosts off.

4. Power Controller - Performing VM packing algorithm.

These four component communicates with each other via the REST API.

2.8.1.1 VM Packing Algorithm

Objective of the algorithm is to place maximum number of VMs in minimum number of physical machines. So that this problem is considered as a bin-packing problem. A bin represent a physical machine and items are the VMs that need to be allocated. Bin sized represent the available CPU capacities of the machines. Weight of an item represent the CPU consumption on each VM. Algorithm used to solve the problem is First-fit decreasing algorithm which is widely used to solve bin-packing problems.

Figures 2.8.2 and 2.8.3 below illustrate how the consolidation and deconsolidation of VMs occur.

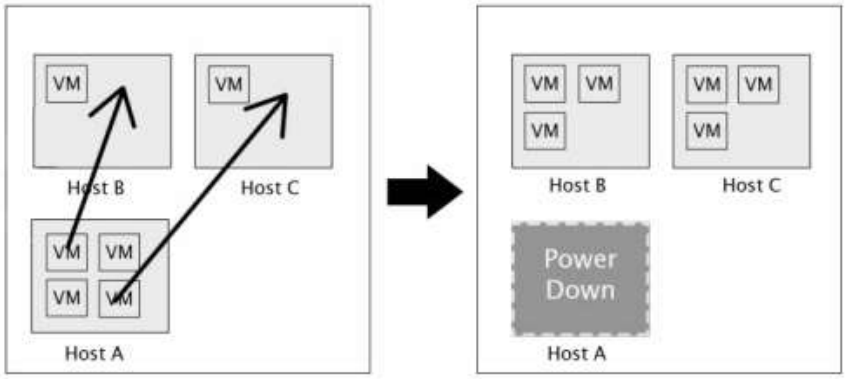

Figure 2.21 - VM Consolidation Framework: Consolidation [29].

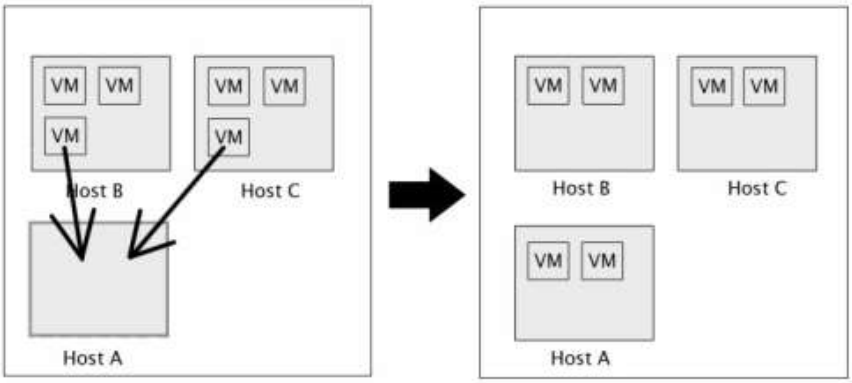

Figure 2.22 - VM Consolidation Framework: Deconsolidation [29].

This consolidation policy is aimed to minimize power utilization and thus reduce the operating cost. Paper [29] itself states that there is no one-fits-all solution and the policy has to be modified according to the application. For our implementation we are concerned on resource utilization rather than power. Besides that our policy also needs to be concerned on user priorities, resource allocation period, performance capabilities of the hosts and current usage of the hosts.

2.8.2 VMware DRS and VMware DPM

2.8.2.1 VMware Distributed Resource Scheduling (DRS)

VMware DRS [8] balances resources automatically among VMs based on predefined rules. These rules are defined by users indicating how they want VMs to share resources and what the priority levels for resources are for each VM. When a VM has faced with increasing of load, VMware DRS automatically allocates additional resources by redistribution of VMs among the physical servers according to these rules and policies. This Virtual machine can be reallocated in to a server which has more resources or the other VM on the current server can be migrated in to other servers to make required space for it. The live migration of VMs is done through VMware vMotion.

VMware DRS allows both automatic and manual modes. In the automatic mode VMware DRS finds the most suitable distribution for the VMs and apply it. In the manual mode VMware DRS suggest a suitable distribution and it is the system administrators decision to apply it or not.

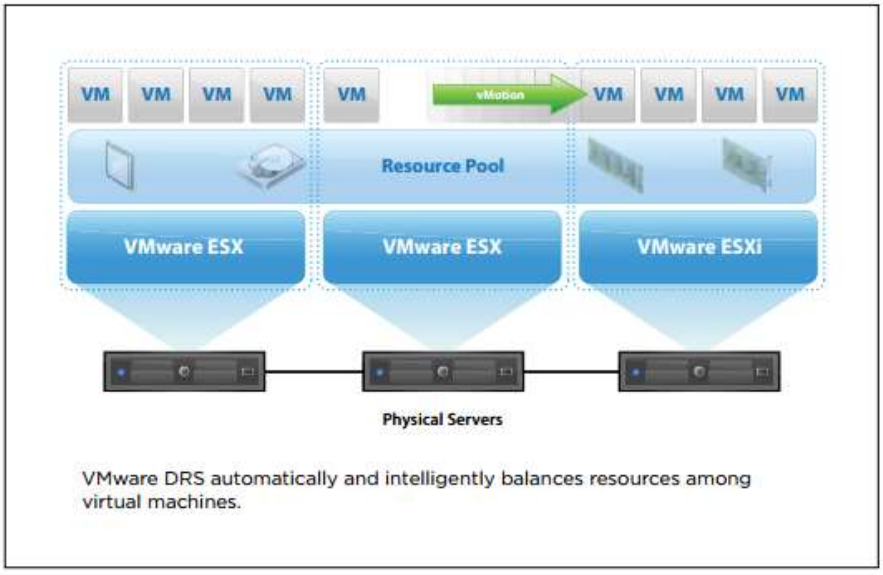

Figure 2.23 - VMware DRS overview [8].

2.8.2.2 VMware Distributed Power Management (DPM)

VMware DPM continuously monitors resource requirements of a VM cluster and if the requirement falls down it consolidates workloads to save power consumption. When resource requirement is increased, switched off hosts can again get into action and can deconsolidate workloads. Based on the capacity of the cluster versus the requirement for the resources, VMware DRS decides on consolidation or deconsolidation of hosts and switch off or on machines accordingly to save energy.

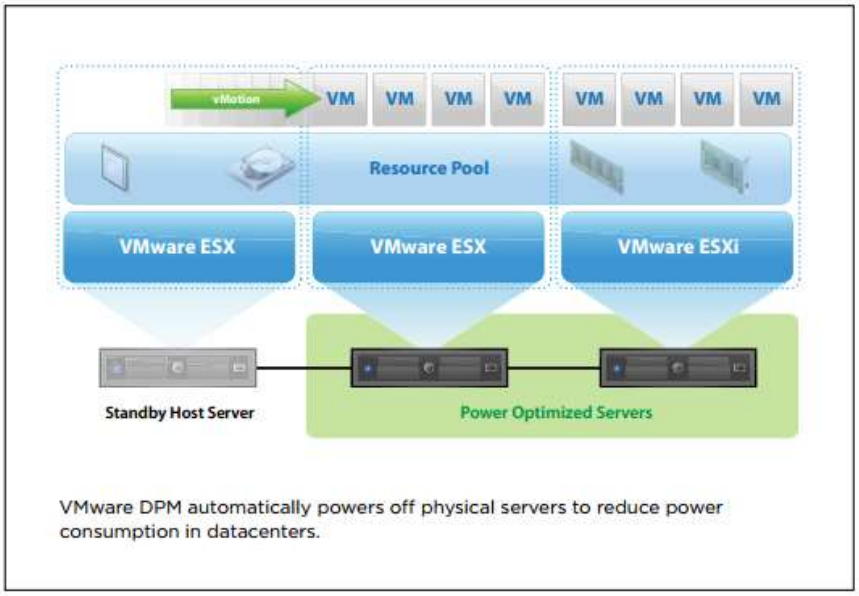

Figure 2.24 - VMware DPM overview [8].

2.8.3 Algorithms to Improve Scheduling Techniques in IaaS Cloud

As Optimal resource scheduling has been a great challenge to cloud platforms this paper [9] come up with a VM resource scheduling component named Haizea, which makes scheduling decisions upon user lease requests. It offers four type of leases. They are, immediate, BestEffort (BE), Advance Reservation (AR) and DeadLineSensitive (DLS) leases. Immediate leases requires immediate attention. Best Effort lease can be allocated when the resources are available. Advanced Reservation lease asks for the exact requested slot and DLS lease is same as BE lease but, it has a deadline.

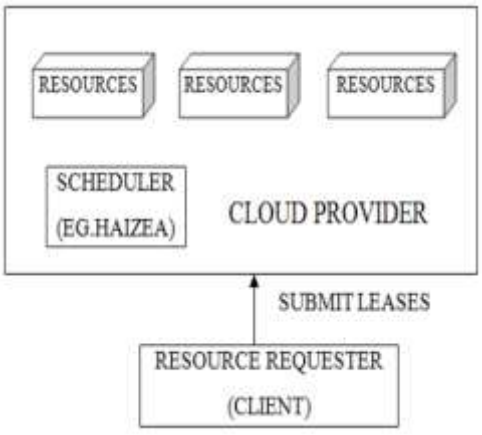

Figure 2.25 - Lease Requests and Heizea [9].

Heizea is simply a lease manager which is implemented in Python. It is doing the scheduling of resources according to user requests.

2.8.3.1 Problems faced by Heizea

- BE and DLS lease is considered as a low priority and a pre-emptible lease. Therefore starvation is possible.
- Immediate and AR leases will rejected if it cannot be scheduled for the requested time slot.

Following three algorithms are suggested in [9] to address these problems mentioned above.

2.8.3.2 Wait queue in Immediate Leases

A wait queue is introduced to minimize the rejection rate. Generally if an Immediate lease is rejected, that request should be again made for another scheduler and even then, there's no assurance in obtaining the resource. Submitting requests repetitively takes time. Therefore paper [9] suggest to offer the next available free slot for the user and enqueuing that into a wait queue. Then the user can make the decision to accept it or not as he or she desires.

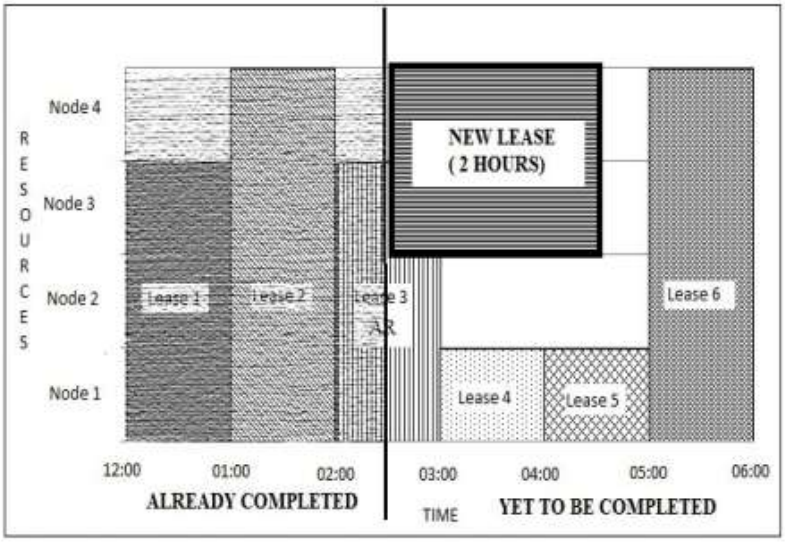

Figure 2.26 - Scenario for new Immediate Lease [9].

Figure 2.8.7 shows a scenario when a new lease comes at 2.50 but an AR lease is already allocated for that resources. So that in a typical implementation new lease will be rejected. But the proposing algorithm add this new lease in to wait queue and offer the next available timeslot 3.00 to 5.00 to it. If a user accepted the suggested slot that slot will be allocated, hence reducing the rejection rate.

2.8.3.3 Reservation queue in AR Leases

A reservation queue is introduced. If the exact timeslot requested is not available and if that slot can be scheduled in a slot time where it is the exact time plus threshold, that information is given to the user and

the lease is moved in to the reservation queue. This threshold value is 5 to 12 percent of the requested lease. User can either accept or not accept this slot. Thus the rejection rate is decreased.

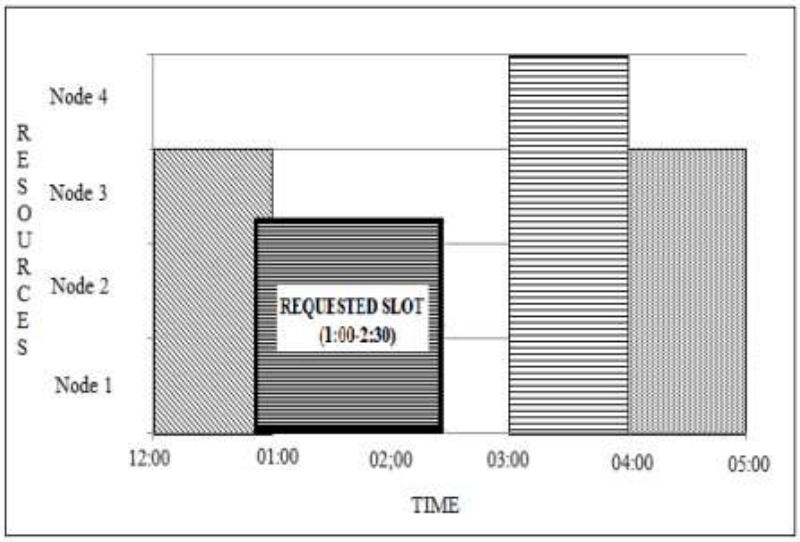

Figure 2.27 - Scenario for new AR Lease [9].

Here the AR lease is requested to 1.00 to 2.30. But that slot is not available and a time slot is available after 10 minutes. If the threshold value is assumed as 11, 10 minutes is within the threshold value and therefore that time slot is suggested to the user and the lease is moved to the reservation queue. User can utilize the slot if he or she desires.

2.8.3.4 Deadline queue in DLS Leases

A deadline queue is implemented. For a particular available slot if even a very small part of a work executes after the deadline, that work cannot be scheduled in that slot. So a new threshold value is introduced. If a work can be executed in its deadline plus the threshold value, that information is sent to the user and the lease is moved in to the deadline queue. This threshold is also can be 5 to 12 percentage of duration of work. This will considerably reduce the rejection rate.

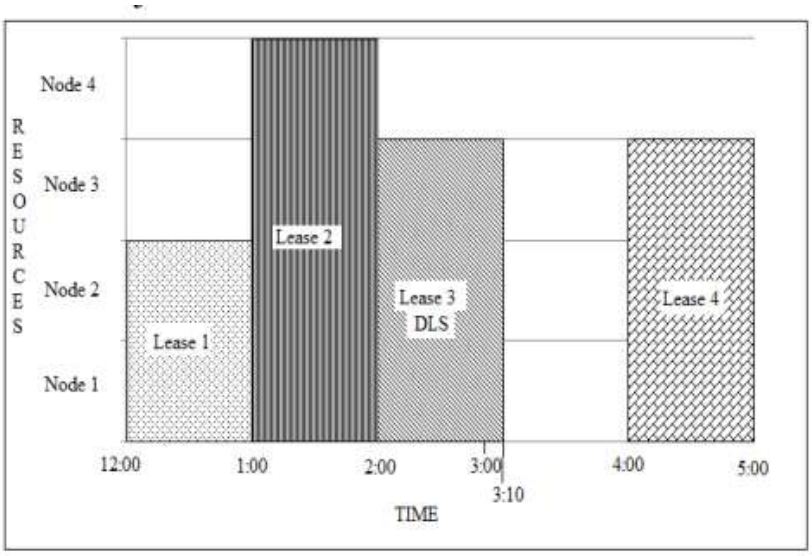

Figure 2.28 - Scenario for new DLS Lease [9].

In the Figure 2.8.9 above, Lease 3 is a DLS lease and its deadline is at 3.00. In the current system it will be immediately rejected as it can finish only by 3.10, which is 10 minutes after the deadline. When a threshold value of 10 minutes is used, this lease is accepted.

As these algorithms address the problems in typical implementations of schedulers, we can consider these algorithms when implementing resource scheduling techniques in our implementation. But necessary enhancements and adoption as per our implementation will be needed.

2.9 High Performance Computing in the Cloud

High Performance Computing (HPC) applications are good clients of cloud architecture. These applications can be categorized as CPU, memory, and data intensive applications. The applications used to run on supercomputers or clusters. Supercomputers and cluster computers very expensive and hard to maintain. In present organizations who wants run HPC application chose cloud computing because it has this infrastructure of pay per user basis. The resource are scalable such that when demand for the applications increase resources can be allocated dynamically. IaaS will provide virtual nodes for computations so that people could use computers and host their applications.

Benefits of running HPC in cloud:

1. Scalable applications
2. Parallel processing
3. Data sharing

High performance applications can be resource eaters. The resource demand of the applications could increase in a very rapid way that the user may want to scale the application in order to facilitate the growing need. Cloud infrastructure enables that need so user can increase the number of CPU's or the memory or else the capacity of the hard drive. Users can request resources which will enhance the parallel processing capabilities.

The derived data using applications can be shared among fellow scientists using service-level agreements. Previously evaluated data could be useful for other computations and it depends on the organizations and users who gets facilitated by the cloud infrastructure.

Apart from benefits as mentioned above, there are challenges that cloud need to overcome when running HPC applications. Real-time data processing can be an issue due to network latency e.g., seismic wave analysis. Cloud does support good infrastructure to higher CPU intensive applications [30] but there is still a void left to support high memory and data intensive applications.

2.10 Resource Description Languages

"Current Cloud Computing setups involve a huge amount of investments in datacenters, which are the underlying infrastructure of Clouds. These massive-size datacenters bring many well-known challenges such as the need for resource over-provisioning, the high cost of heat dissipation and temperature control, and power losses due to the distribution systems [31]. In contrast to this, Distributed Clouds, or just D-Clouds [32], can reduce these problems through the use of smaller datacenters sharing resources across geographic boundaries. Also, D-Clouds can reduce communication costs by provisioning servers and data close to end-users" [33].

Challenges to overcome in a distributed cloud environment:

- Resource modelling
- Resource offering and treatment
- Resource discovery and monitoring
- Resource selection

According to paper [32], it is important to model the resources and type of the service the cloud will offer. And resource modelling also contributes to resource allocation where the resource description language will provide insights of the user requirement.

In our project we need to model a resource description language where users can query and check whether they could get a machine with some resource specification e.g., 1 core, 4GB RAM, 20 GB Hard Disk Space.

So we should have a resource discovery tool where it will use the resource description language which is modelled in order to answer the user query.

```
<resource_request>
        <vm_request>
                <vm_count>10</vm_count>
                <os>
                <name>Ubuntu</name>
                        <kernel_version>3.6</kernel_version>
                </os>
                <cpu>
                        <cores>2</cores>
                        <architecture>x64</architecture>
                </cpu>
                <min_memory>4</min_memory>
                <min_storage>
                        <primary>4</primary>
                </min_storage>
                <network>
                        <min_bandwidth>8</min_bandwidth>
                </network>
                <preferred_priority>3</preferred_priority>
        </vm_request>
</resource_request>
```

The above mentioned resource description model is written in XML which captures some of our requirements but has a great potential of increasing the number of constraints e.g., time duration for resource allocation and storage type. So it is apparent that a resource description language plays a major role when stating resource requirements and allocating. This helps the dynamic management of resources and also satisfies developer goals.

Chapter 3

3 Design of the Resource Scheduler

This chapter discusses the design of the proposed solution. Section 3.1 presents an overview of the solution including the high-level architecture and component-based architecture. In Section 3.2, we describe components of our system in detail. It also describes about the infrastructure including CloudStack IaaS Framework [18], Zabbix monitoring system [24], and MongoDB [44] database. Architectural components of the system are also discussed. Section 3.3 further describes the steps that we followed in setting up infrastructure.

3.1 Solution Overview

Rather than extending the functionality of CloudStack IaaS framework by means of modifying its source code, we have developed the resource scheduling solution that resides outside the CloudStack. Resource Scheduler, which is implemented using Node.js [45], runs as a separate application which can access resource monitoring details of Zabbix using Zabbix REST API and execute VM scheduling decisions on CloudStack by accessing CloudStack REST API. Resource Scheduler itself provides a RESTful web service [46] which cloud users can use to send Resource Allocation Requests to the scheduler. Details of this RESTful web service will be described in Section 4 which discusses implementation details. As illustrated in Figure 3.1, a cloud user sends a Resource Allocation Request to the Cloud Scheduler using its REST API as the step 1. In step 2, Resource Scheduler will call Zabbix Monitoring System to fetch latest resource utilization information. In step 3, Zabbix will return latest monitoring data gathered by itself to the Resource Scheduler. Using these monitoring information, resource scheduler takes decision on which host(s) should this allocation be performed and use CloudStack REST API to perform VM deploying operation in step 4. Finally, in step 5, CloudStack will deploy VMs as the API request specifies.

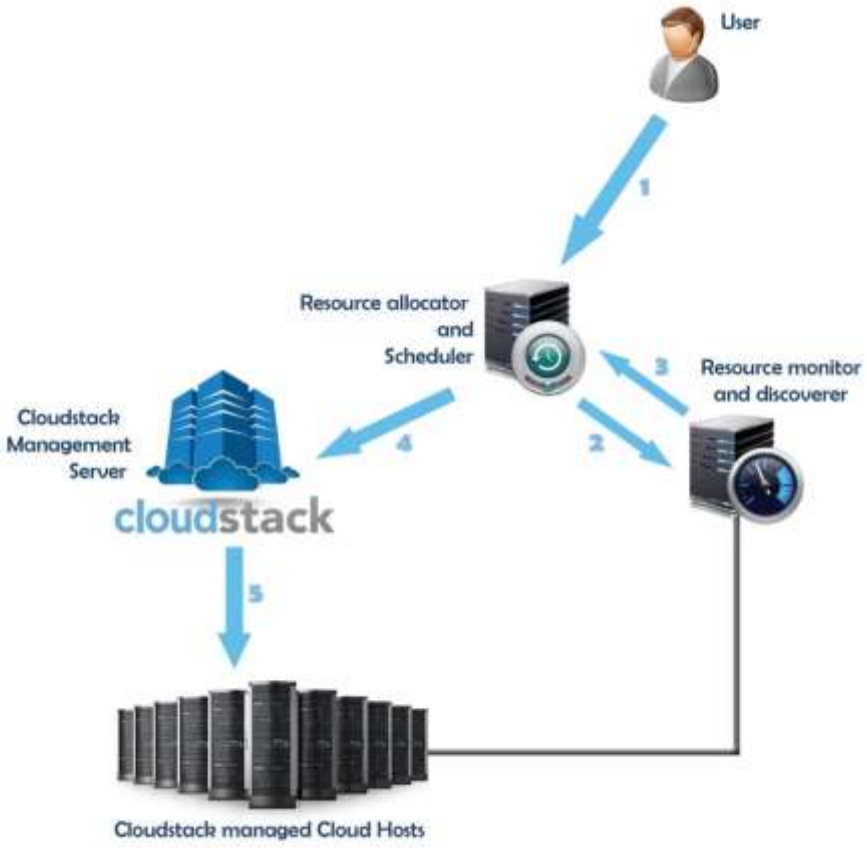

Figure 3.1 – Overview of the solution.

Next section will describe how Resource Scheduler performs taking decision on a resource allocation internally.

3.1.1 Request Flow

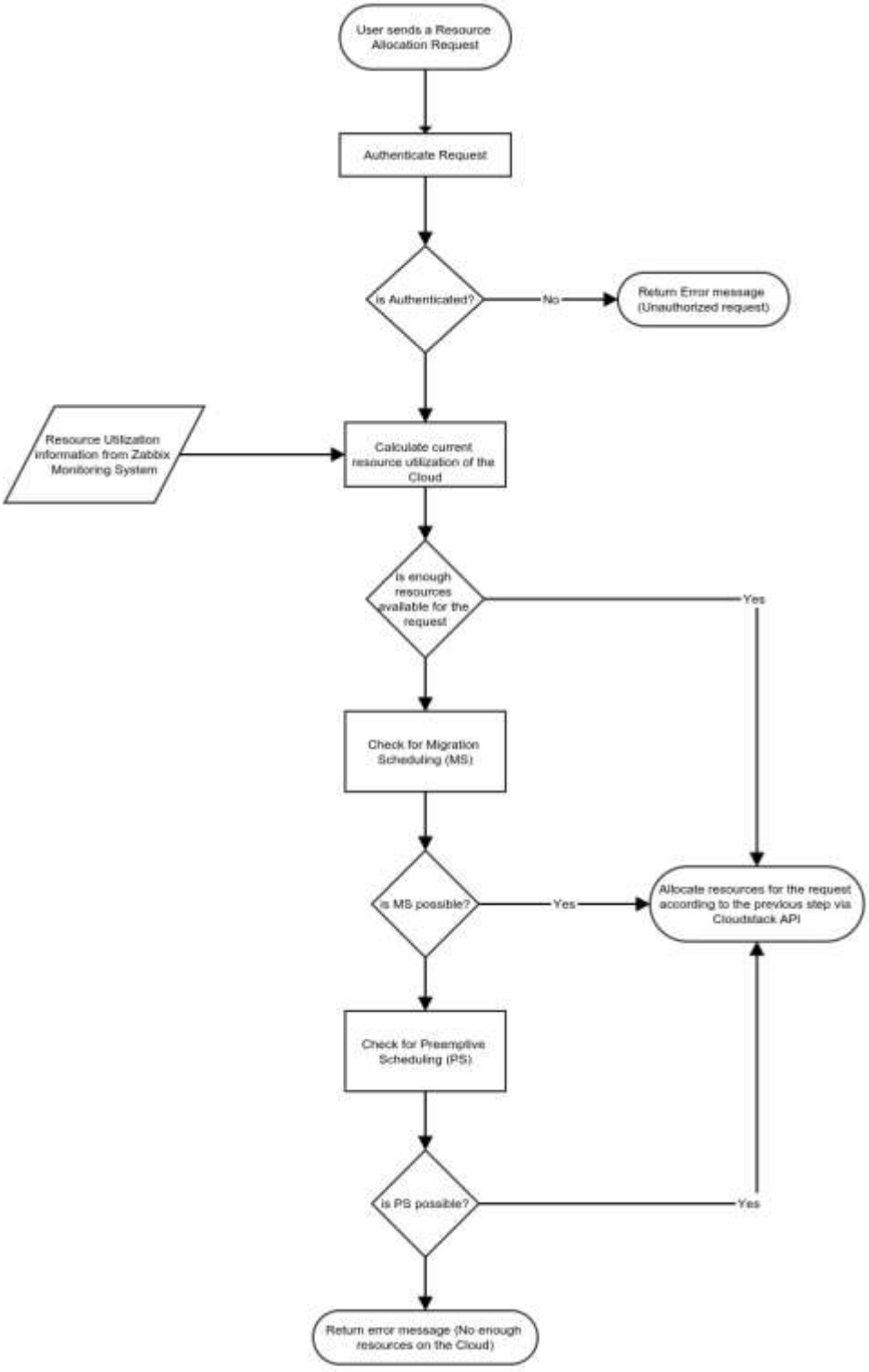

Figure 3.2 – Resource Scheduler Request Flow.

Figure 3.2 is flowchart diagram which explains the sequence how a request is served. It shows how a request is sent from one component in the system to another component system until it is allocated in the Cloud.

3.1.2 High Level Architecture

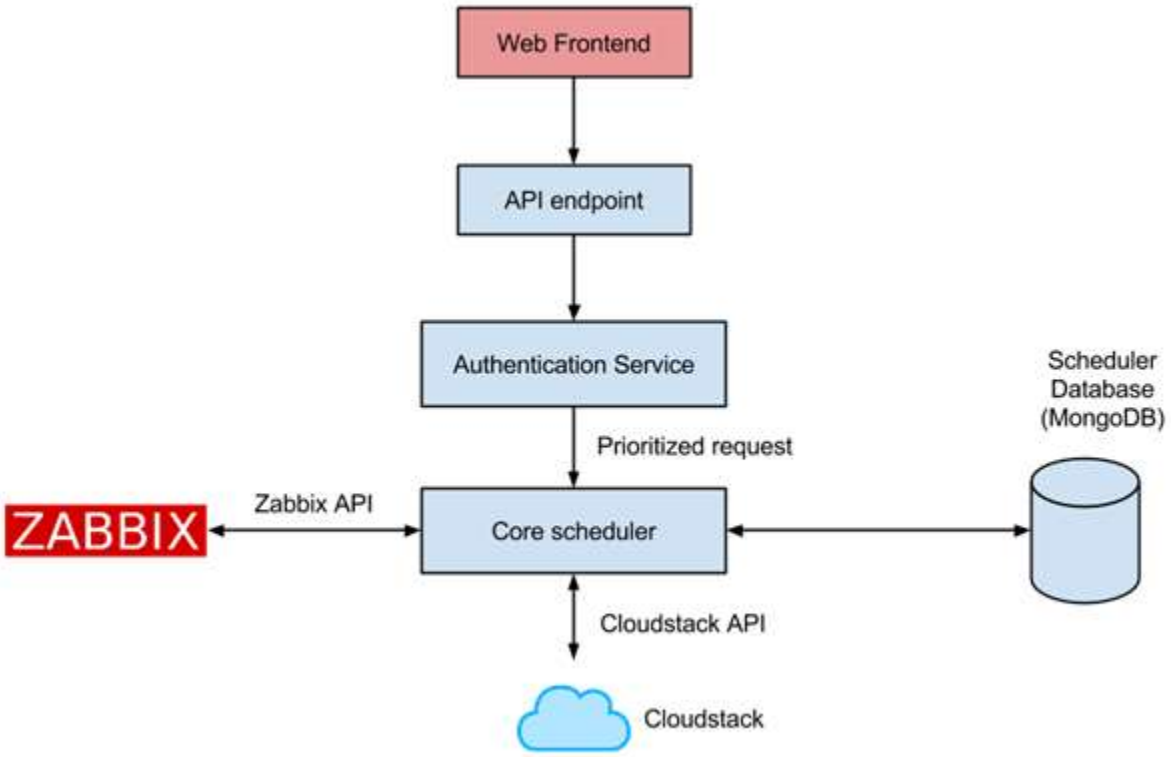

Figure 3.3 – Resource Scheduler High Level Architecture.

Figure 3.3 is a High Level architecture diagram of our system. This architecture diagram clearly shows how our resource scheduler works in the middle of Zabbix Resource Monitoring system and CloudStack IaaS framework and performs the coordination among them. A user can submit a query to the scheduler through Web Frontend or using the API endpoint provided by the scheduler. When the request is validated, authenticated and authorized, authentication service forwards an authorized and prioritized request to the core scheduler.

3.1.3 Component-Based Architecture

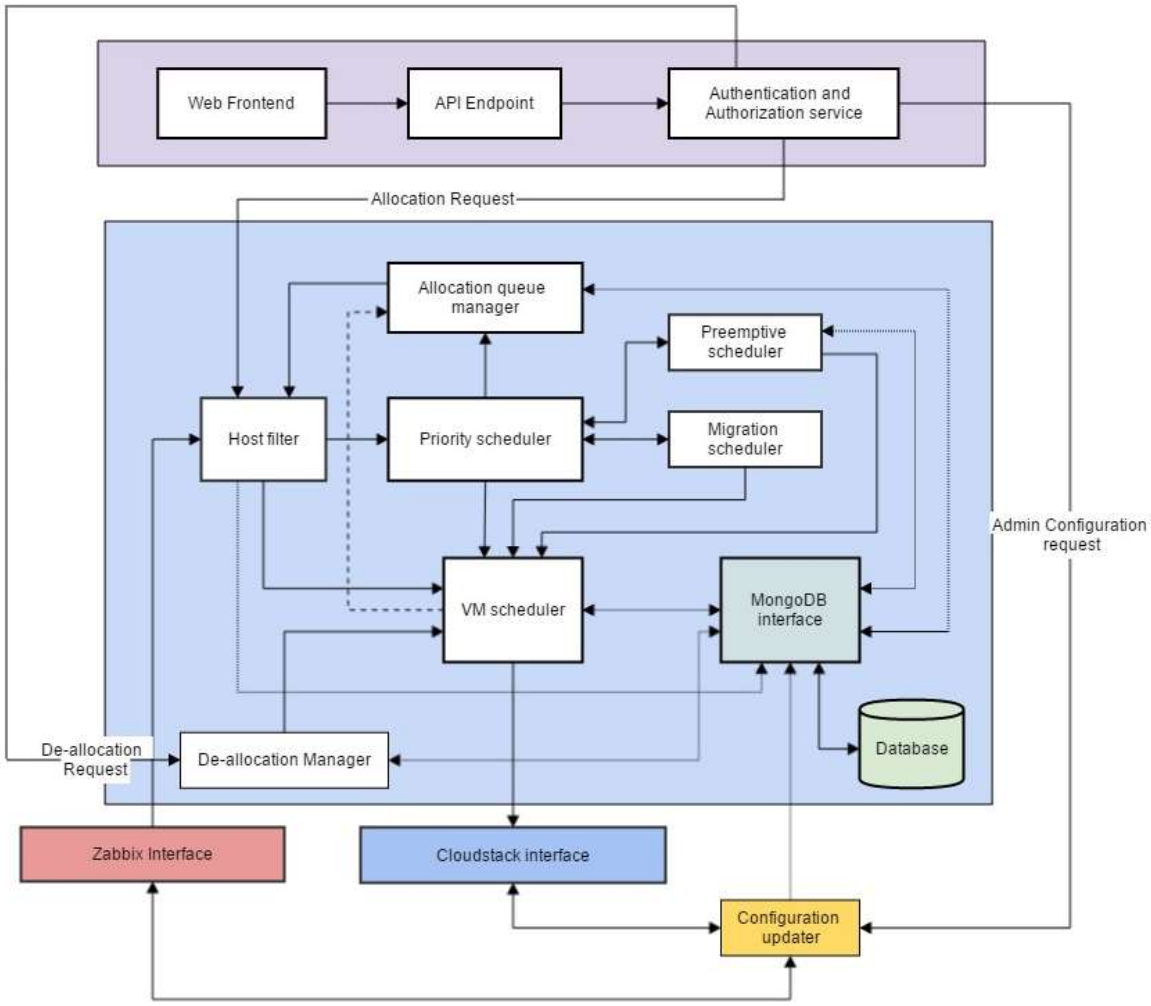

Figure 3.4 – Core Scheduler Component Architecture.

Figure 3.4 illustrates the internal component based architecture of the Core Scheduler. Functionality and purpose of each component will be described in the next section.

3.2 Components

3.2.1 Front-end

3.2.1.1 Web Interface

Our resource scheduler provides two access methods for the cloud users. One method is via the Web Interface. A user can compose a resource allocation request via the web interface and submit. Web interface internally access the API provided by the resource scheduler.

3.2.1.2 API Endpoint

API endpoint is a REST API to manage functionality of the resource scheduler. This REST API provides functions to issue resource allocation requests and perform admin/user functions. Following are the functions that are currently supported by the API.

Table 3.1 – Resource Scheduler API methods.

Description	**URL**	**Request Method**
Create User (*Admin only)*	/admin/createUser	POST
Login	/login	POST
Issue a request	/request	POST
Read Configuration (*Admin only)*	/configuration	GET
Update Configuration (*Admin only)*	/configuration	POST

- *Creating a User*

  Creating a User is only allowed for Administrator users. Create User request should be sent as a **POST** request to the URL **/admin/createUser**. Create User request body should have following format:

```
{
      username: “User 1”,
      password: “passw@rd”
      userinfo: {
            firstName: “User”,
            lastName: “One”
      },
      priority: 3,
      admin: true
}
```

SHA-256 [47] algorithm is used to hash passwords to store in the database and the timestamp at the point where the account was created is used as the salt for hashing. A successful creation of a new user account will return a response similar to the following:

```
{
      "status":"success",
      "code":200,
      "message":"User account was created successfully",
      "returnObject":"54d09cfc29da276c6e94cc40"
}
```

- *Login*

  Login function should be performed before sending any Resource Allocation requests to the API. Users should send a **POST** request to the **/login** with JSON [48] request body including the information about the user account to be created. Login function accepts username and password in a JSON request and a successful login will return a response including the session ID to be used in future requests.

  Request body format:

```
{
      "username":"adminuser",
      "password":"adminpassw@rd"
}
```

  Response body format:

```
{
      "status":"success",
      "code":200,
      "message":"Login successful",
      "returnObject":{        "sessionID":"f192aa719b98bb6dc9725f
9db0e4fa7ee31a9d177ca8a740e6297d36b6d4ea2a"
   }
}
```

  Login function will create a session ID and store it in the database for future session validation and it will be sent back to the user in the response as above. User should include above session ID in future Resource Allocation Requests in order to get authenticated.

- *Issue a request*

  Once a user is logged in, he/she can issue Resource Allocation Requests to the Resource Scheduler using this method in API. Resource allocation Request should be composed as a XML document which describes the reources the user requires. This request should be sent as a **POST** request to

the URL **/request** with ***Content-Type*** header set to ***application/xml***. We are currently using XML as the Resource Description Language. But, to be consistent with other API methods, we are currently implementing this API method to support requests with ***Content-Type*** set to ***application/json***. To differentiate these two types, ***Content-Type*** header is required in the **POST** request for Resource Allocation. Internally, Resource Scheduler selects appropriate parser (XML or JSON) according to the ***Content-Type*** header.

Following is an example Resource Allocation Request:

```
<resource_request>
        <group>
                <vm_count>10</vm_count>
                <image>
                        <type>iso</type>
                        <id>280b40d0-6644-4e47-ac7c-074e2fa40cd4</id>
                </image>
                <cpu>
                        <cores>1</cores>
                        <frequency>1</frequency>
                        <unit>GHz</unit>
                        <architecture>x86</architecture>
                </cpu>
                <min_memory>
                        <size>2</size>
                        <unit>GB</unit>
                </min_memory>
                <min_storage>
                        <primary>5</primary>
                        <unit>GB</unit>
                </min_storage>
                <priority>3</priority>
        </group>
        <session_id>3deb1bb861f34b527e6709c655fff139b36c2dc43d8b3e29e3914bf8b23ce069</s ess
ion_id>
</resource_request>
```

- *Read Configuration*

  Configuration information includes the settings related to all components in the resource scheduler. To access configuration information, user should be an administrator. This API method can be accessed by sending a **GET** request to the URL **/request**. Configuration information can be retrieved as a JSON document.

- *Write Configuration*

  Writing a new configuration can be done by sending a **POST** request to the URL **/request** with a document containing configuration information as a JSON document. This method is also protected and only administrator users are allowed to access the method.

3.2.2 Authentication Service

Authentication service provides authentication and authorization services for the resource scheduler. API authentication is provided by a session key validation mechanism. Users need to login into the resource scheduler before issuing any request. Once logged in, Authentication service generates and returns a persistent session key which should be re-sent in following requests. Generated session key is stored in MongoDB and when a user issues a request, validity of the session key is evaluated by the Authentication service and identify the user's privileges.

Passwords are stored in database as salted **SHA-256** hashes where timestamp at the creation of user account is used as the salt. In the first implementation of the Authentication Service, we used MD5 as the hashing method but then updated to **SHA-256** for better security. Only the ***login()*** and ***createUser()*** methods are provided by Authentication service to outside via the REST API, and ***authorizeResourceRequest()*** function is used internally by VM scheduler to authenticate and authorize an incoming resource request. Following diagram is a graphical representation of Authentication and Authorization services provided by the module.

3.2.3 Core Scheduler

3.2.3.1 Host Filter

Host Filter performs the filtration of hosts according to their monitored statistic data through Zabbix. That filtered hosts are then sent to VM scheduler as the candidate hosts for scheduling of a particular job.
Complete steps taken by the Host Filer are illustrated below.

1. Host Filter receives the resource request and all host information. This host information is monitored through Zabbix and it contains all host information in the infrastructure. For every host, an ID is created in Zabbix.

2. Statistics are collected for specified data items in each host. Monitored items includes, memory information, CPU load and CPU utilization. For each item, history values are also collected apart from the current value. In our application we only take the previous value and the current value as we are calculating the EWMA [49] (exponentially weighted moving average) using the last EWMA which is stored in the database.

   Equation for calculating EWMA:

$$EWMA(new) = * EWMA(last) + (1 - \alpha) * Current\ value$$

   α- predefined constant

3. As mentioned above EWMA is calculated for each item and using that values all the candidate hosts which fulfill the resource requirements for the request are found.
4. All those candidate hosts information are then sent to VM scheduler so that the scheduling decision can be made in VM scheduler.
5. Other than the candidate hosts, Host Filter also finds the hosts that fulfill memory requirements for the request and that is also passed to VM scheduler. That information is sent for the use of Migration and Preemption Scheduler.

3.2.3.2 VM Scheduler

VM Scheduler provides orchestration for all components in the Resource Scheduler including Core Scheduler components and Authentication Service.

Following is the flow of how a request is handled by the VM scheduler:

1. VM Scheduler receives a Resource Allocation Request via REST API
2. VM Scheduler passes the request to the Authentication service for authentication and authorization
3. If the request is authenticated and authorized, authorized request is taken by the VM Scheduler and forwards it to Priority Scheduler which is the coordinator of Migration Scheduler and Pre-emption scheduler.
4. Either the Priority Scheduler or Migration Scheduler will return a selected host where the request can be allocated. VM Scheduler then performs request allocation on the selected host via CloudStack API using the CloudStack Interface.

In addition to resource allocation, resource de-allocation is also performed by VM Scheduler with the support of De-allocation manager.

3.2.3.3 Priority Scheduler

Priority Scheduler acts as the coordinator of Migration Scheduler and Pre-emption scheduler. Priority Scheduler first forwards the authorized resource request to Migration Scheduler to find host(s) to allocate the incoming request. If Migration Scheduler does not return any host information, Priority Scheduler then checks whether there are previous allocations with priority less than the incoming requests, and then sends those allocation information along with the request to the Pre-emption scheduler. If Pre-emption scheduler selected some hosts, those host information is sent back to the VM Scheduler by the Priority Scheduler.

3.2.3.4 Migration Scheduler

Migration Scheduling is the second step of VM Scheduler if there appears to be no resources to serve the incoming Resource Request. When Priority Scheduler forwards incoming request to the Migration Scheduler, it checks whether the enough resources for the incoming request can be made available in any of the hosts by migrating some of its virtual machines to another hosts. If this is possible, Migration scheduler algorithm is as follows:

3.2.3.5 Pre-emption Scheduler

When the Migration Scheduler is unable to find a host for an incoming request by reshuffling VMs on the cloud, the request is then passed to the Pre-emption scheduler. In pre-emption step, Pre-emption scheduler checks the priority of incoming request; compare it with current allocations and takes decision on pre-empting a running allocation in order to gain resources on the incoming resource request. When a request is preempted, preempted request is saved in the database in order to be rescheduled later when resources are available. When preemption fails, it means that there are no enough resources to be allocated for the incoming request and then the Resource Scheduler returns an error message to the user via API stating that there are no enough resources to handle his request. When there are specific set of VMs to be preempted, Preemption scheduler calls an additional web service implemented in Java, called 'JVirshService'. In this external call, Preemption Scheduler will pass the list of VMs to be preempted by a RESTful API call in JSON format and JVirshService will perform the preemption.

3.2.3.6 'JVirshService' RESTful Java Web Service

JVirshService is a java RESTful web service which runs separately from the main Node.js application. This web service internally uses 'virsh' [50] command line tool to issue VM snapshot commands to the hypervisor via libvirt API. When preemption scheduler sends a set of virtual machines to be preempted to the JVirshService in JSON format, it internally calls libvirt and performs taking VM snapshots in hypervisor level. Once the snapshot command gets executed successfully, JVirshService will return the IP address of

the host on which the VMs were preempted. This IP address will be received by Preemption Scheduler and it will inform VM Scheduler that preemption complete.

3.2.3.7 Allocation Queue Manager

Allocation Queue Manager is the component which stores the Resource Requests which cannot be served with currently available resources in MongoDB. When a request arrives, there are currently no enough resources to be allocated for the request, Resource Scheduler then tries Migration Scheduler and Preemption Scheduler to find enough resources for the incoming request. In the Preemptive scheduling phase, if there are no allocations with priority less than the incoming request, Preemption Scheduler returns no hosts for the incoming request. At this point, the request is queued by Allocation Queue Manager and will be allocated when enough resources is available for the request in the Cloud.

3.2.3.8 De-allocation manager

De-allocation manager is to perform resource release and re-allocate preempted requests/queued requests in the Allocation Queue Manager.

3.2.3.9 Configuration updater

Configuration updater is the module which provides methods to change the configuration of the Resource Scheduler. Configuration information of the all components of the Resource Scheduler can be changed using this module. Access to this module is protected and only administrator users are allowed to make configuration updates.

3.2.3.10 Database Interface

Database interface is the component we use to store and retrieve information from MongoDB storage. Accessing MongoDB is provided by a 3rd party Node.js module called Mongoose. We use Mongoose module to perform queries and updates on MongoDB database.

3.2.3.11 CloudStack Interface

CloudStack interface is implemented as a Node.js module called ‘csclient’. ‘csclient’ is also a 3rd party module which provides easy access to the CloudStack API with Node.js where complex functionality including request signing is performed inside.

3.2.3.12 Zabbix Interface

Zabbix Interface is implemented as another Node.js module which provides access to Zabbix Monitoring System via Zabbix API. It has the functions to login into the monitoring system and issue API requests in Node.js.

3.3 Infrastructure Requirements

3.3.1 Setting up CloudStack

Setting up the necessary infrastructure where we could run the solution was a high priority requirement. CloudStack was the IaaS framework which was used to setup the cloud. This wasn't an easy task because deploying CloudStack required a special skill set where users need to have a sound understanding on Networks, NFS and client and server architectures.

3.3.1.1 Objectives

1. Setting up the CloudStack management server
2. Setting up the Database
3. Setting up Host machines with necessary modifications (Configuring the KVM hypervisor)
4. Creating an Internal DNS server
5. Setting up a Network File system for primary and secondary storage
6. Setting up System VM Templates
7. Creating a Virtual Machine

The cloud consists of four physical nodes, where one physical host was used to deploy CloudStack management server. Other three nodes were used as resource providers where the actual Virtual Machines will be hosted in these machines. All four nodes were identical where a cluster was created which was used to add other three physical nodes.

CloudStack needs to have an internal DNS server where it will resolve the hostnames of the physical nodes. And it also needs a network file system where it will be a mount point to the host machines. The NFS server stores necessary templates and ISO images which will be used to create virtual machines.

3.3.1.2 Install Openntpd

1. apt-get install openntpd

3.3.1.3 Configure IP Tables

1. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p tcp --dport 22 -j ACCEPT
2. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p tcp --dport 80 -j ACCEPT
3. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p tcp --dport 8080 -j ACCEPT
4. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p udp --dport 111 -j ACCEPT
5. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p tcp --dport 111 -j ACCEPT
6. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p tcp --dport 2049 -j ACCEPT
7. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p tcp --dport 32803 -j ACCEPT

8. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p udp --dport 32769 -j ACCEPT
9. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p tcp --dport 892 -j ACCEPT
10. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p udp --dport 892 -j ACCEPT
11. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p tcp --dport 875 -j ACCEPT
12. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p udp --dport 875 -j ACCEPT
13. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p tcp --dport 662 -j ACCEPT
14. iptables -A INPUT -s 10.1.0.0/16 -m state --state NEW -p udp --dport 662 -j ACCEPT
15. apt-get install iptables-persistent

3.3.1.4 Install CloudStack

1. echo "deb http://Cloudstack.apt-get.eu/ubuntu precise 4.4" > /etc/apt/sources.list.d/Cloudstack.list
2. wget -O - http://Cloudstack.apt-get.eu/release.asc | sudo apt-key add -
3. apt-get update
4. apt-get install Cloudstack-management

3.3.1.5 Preparing the NFS

1. apt-get install nfs-kernel-server quota
2. mkdir /mnt/primary -p
3. mkdir /mnt/secondary -p
4. echo "/mnt *(rw,async,no_root_squash,no_subtree_check)" >> /etc/exports
5. exportfs -a

3.3.1.6 Install MySQL server

1. apt-get install mysql-server
2. mysql_secure_installation
3. echo "[mysqld]
4. innodb_rollback_on_timeout=1
5. innodb_lock_wait_timeout=600
6. max_connections=350
7. log-bin=mysql-bin
8. binlog-format = 'ROW'" > /etc/mysql/conf.d/Cloudstack.cnf
9. service mysql restart
10. ufw allow mysql

3.3.1.7 Deploy the CloudStack Database

1. Cloudstack-setup-databases cloud:SECURE_PASSWORD_FOR_CLOUDSTACK@localhost --deploy-as=root:MY_ROOT_DATABASE_PASSWORD

These commands will finish up installing the Cloudstack management server.

3.3.1.8 Install CloudStack Agent in the Host Machine

1. echo "deb http://Cloudstack.apt-get.eu/ubuntu precise 4.4" > /etc/apt/sources.list.d/Cloudstack.list
2. wget -O - http://Cloudstack.apt-get.eu/release.asc | sudo apt-key add -
3. apt-get update
4. apt-get install Cloudstack-agent

3.3.1.9 Setting up the Network

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet manual

# Public network
auto cloudbr0
iface cloudbr0 inet static
  address 10.8.100.202
  netmask 255.255.255.0
  gateway 10.8.100.254
  dns-nameservers 192.248.8.97
  bridge_ports eth0
  bridge_fd 5
  bridge_stp off
  bridge_maxwait 1

# Private network
auto cloudbr1
iface cloudbr1 inet manual
  bridge_ports none
  bridge_fd 5
```

bridge_stp off

bridge_maxwait 1

3.3.1.10 Configuring the KVM hypervisor

1. sudo nano /etc/libvirt/libvirtd.conf
2. listen_tls = 0
3. listen_tcp = 1
4. tcp_port = "16509"
5. mdns_adv = 0
6. auth_tcp = "none"
7. sudo nano /etc/default/libvirt-bin
8. libvirtd_ops="-d -l"
9. sudo nano /etc/libvirt/qemu.conf
10. vnc_listen = "0.0.0.0"

3.3.1.11 Disable apparmor in Ubuntu

1. ln -s /etc/apparmor.d/usr.sbin.libvirtd /etc/apparmor.d/disable/
2. ln -s /etc/apparmor.d/usr.lib.libvirt.virt-aa-helper /etc/apparmor.d/disable/
3. apparmor_parser -R /etc/apparmor.d/usr.sbin.libvirtd
4. apparmor_parser -R /etc/apparmor.d/usr.lib.libvirt.virt-aa-helper

3.3.1.12 Setup CloudStack Agent

1. sudo Cloudstack-setup-agent

This guide basically finishes the CloudStack installation. CloudStack Management server service will be started and also CloudStack agent should be started in the host machines.

3.3.2 Zabbix Monitoring System

Zabbix Monitoring System is a free and open source Resource Monitoring System which can be used for agent based as well as agentless monitoring. It is a highly scalable resource monitoring system which can span up to 100,000 monitored devices and up to 1,000,000 metrics. We use Zabbix Monitoring System to gather information about the resource utilization of CloudStack cloud when a resource request is received and perform scheduling according to resource availability.

Deployment of our Zabbix Monitoring System includes two components which are, Zabbix Server and Zabbix Agents. Each host in CloudStack managed cloud is installed a Zabbix Agent and registered in

the Zabbix Server. Periodically Zabbix Server pulls resource utilization information from the hosts via Zabbix Agents.

3.3.3 Database Storage

We are using *MongoDB* as our database storage to store information including configuration information, user information, resource allocation requests, etc. Since we are using Node.js as our development language for the resource scheduler, and also we need to store queued Resource Allocation Requests easily, MongoDB was easier than MySQL. If we use MySQL, we would need to use a proper Object Relational Mapping (ORM) or convert JSON documents into a String and store in the database. In that case, information retrieval would also add additional overhead to create objects from relational data or parse string into JSON. Using MongoDB, we can directly store Resource Allocation Requests in JSON format and retrieve them as JavaScript objects easily.

Chapter 4

4 Implementation

This Chapter will discuss implementation details of our project. In this chapter, we will discuss how we performed version controlling of the implementation, project management and tools that we used to perform unit testing in our project.

4.1 Version Control

In our project, we used Git [51] as our Version Control System (VCS). Compared to SVN, using Git is easy for our project since Git is a distributed VCS. We have created separate branches for team members and later merged our work into the master branch to avoid conflicts.

Our project repository can be located at:

https://github.com/dpjayasekara/VirtualOps

4.2 Project Management

We used 'Zoho' online project management portal for project management. 'Zoho' provides various functionality such as creating tasks, assigning those tasks to specific people, monitoring project progress according to task completion and also provides storage capabilities for sharing documents and keep track of online resources.

4.3 Testing

Testing of the Project was done using Mocha Testing Framework which is a Node.js Testing Framework. In addition to Mocha, we used "Should" Node.js module for assertions. Mocha Testing Framework is a widely used JavaScript Testing Framework which makes Asynchronous Testing easier than other frameworks.

Chapter 5

5 Results and Evaluation

In this Chapter, we describe the results that we obtained and evaluate them against the goals that we tried to achieve at the start of our project.

5.1 Resource-Aware Virtual Machine Scheduling

One requirement of our project was to improve the default VM scheduling mechanism of CloudStack in order to improve the availability for users. By default, CloudStack provides four VM allocation algorithms. Allocation algorithm can be changed by updating the Global Configuration parameter named *vm.allocation.algorithm*. Following are the four allocation algorithms supported by CloudStack:

Table 5.1 – CloudStack VM Allocation Algorithms.

Parameter value	Algorithm
random	Pick a random host available across a zone for allocation.
firstfit	Pick the first available host across a zone for allocation
userdispersing	Host which has the least amount of VMs allocated for a given account is selected. This provides load balancing up to a certain extent for a given user. But running VM count is not considered between different user accounts.
userconcentratedpod_random	This algorithm is similar to 'random' but, this considers hosts within a given pod rather than across the zone.
userconcentratedpod_firstfit	This algorithm is similar to 'firstfit' but, only considers hosts within a given pod rather than across the zone.

By default, *random* algorithm is selected as the VM allocation algorithm. None of these algorithms provide resource-aware scheduling. Although *userdispersing* algorithm considers running VM count which belong to given user, it does not consider the resource utilization of VMs and resource availability in the host. Figure 5.1 snapshot from Zabbix Monitoring System shows how VM Allocation is performed using random method in CloudStack. Diagram shows amount of memory available in all hosts.

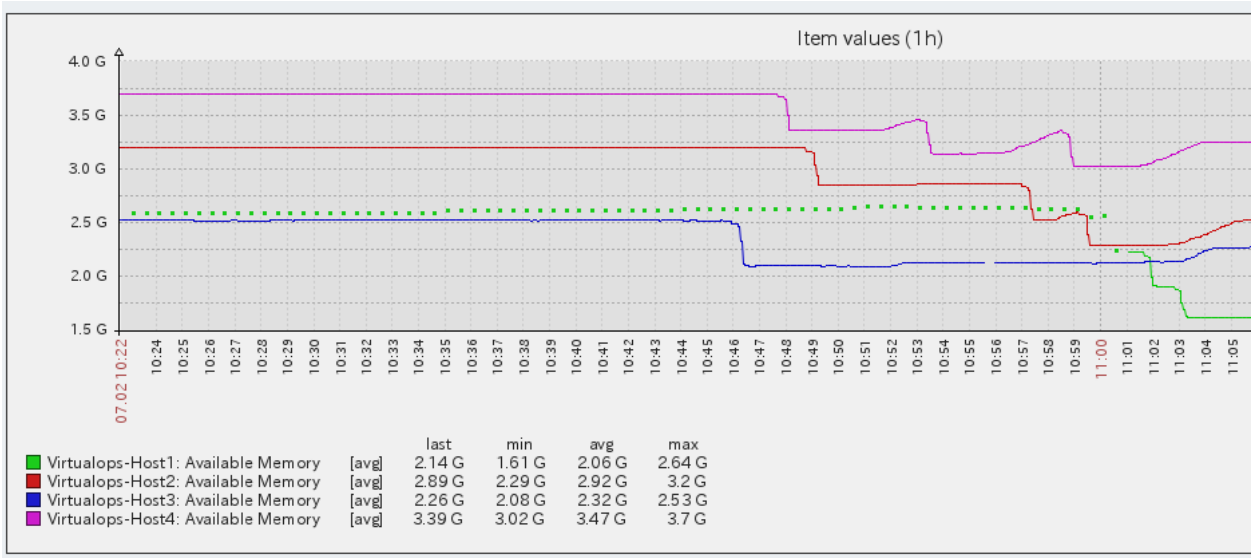

Figure 5.1 – Zabbix Graph for CloudStack Default VM Scheduling Algorithm.

We can identify a VM deploying as a drop of a line in the graph. We can see than the sequence of host selection is Host 3, Host 4, Host 2, Host 4, Host 2, Host 4, Host 2, Host1, Host 1, etc. which shows a random host selection. Using resource utilization information fetched from the Zabbix server, we have implemented mechanism to deploy VM on the host which has least available memory to deploy a given VM. This algorithm is known as *best fit*. The reason for using this algorithm is to keep as much as free memory as possible in a given host, so that those memory can be allocated to future requests asking for more memory. An alternative algorithm would be a load balancing algorithm which would allocate a VM on the host which has maximum available memory to allocate the VM. But, that algorithm could cause frequent shuffling of VMs (perform Migration Scheduling) when memory intensive VM requests come.

Figure 5.2 snapshot from Zabbix Monitoring System shows how Resource Aware VM scheduling is performed with our scheduling algorithm on CloudStack:

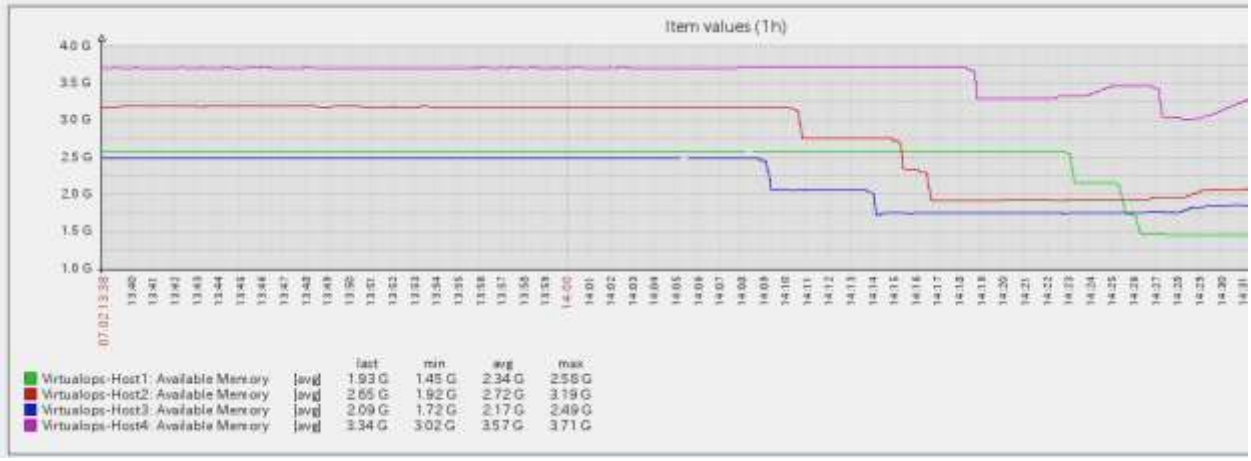

Figure 5.2 – Zabbix Diagram for Resource Aware VM Scheduling.

Above diagram shows available amount of memory in each host in CloudStack. Note that the green line in Zabbix Diagram shows a host which was later added to CloudStack. Figure 5.2 shows change in available memory in four hosts when a list of VMs are deployed with following memory requirements:

VM 1 - 2 GB
VM 2 - 2.5 GB
VM 3 - 1 GB
VM 4 - 2 GB
VM 5 - 2 GB
VM 6 - 2 GB

After these six VM deployments, we added the new host to CloudStack which is represented in the diagram with green line. After that, we can see three next deployments were performed on the newly added host since it has the minimum memory available for the requirements. After these three allocations, its available amount of memory drops than 2GB where final VM in our list cannot be deployed on that host. Therefore the final VM which requires 2GB of memory can only be deployed on Host 4 and it gets allocated on Host 4.

VM 7 - 2 GB
VM 8 - 1 GB
VM 9 - 1 GB
VM 10 - 2 GB

5.2 Preemptive Scheduling

When there are low priority requests are allocated on the cloud and there are no space for another resource allocation on the cloud, we need Preemptive Scheduling to service high priority requests coming. Since high priority requests need to be immediately allocated, we need to preempt suitable number of currently allocated low priority requests and gain space for the incoming high priority request. When there is no space on the cloud for an incoming request, we move into Migration Scheduling phase in which we re-shuffle all VMs in the cloud using VM live migration to gain space on a specific host for the incoming request. If Migration Scheduling is not possible, we then move to Preemptive Scheduling phase, in which we check the priority in incoming request, compare it with currently allocated request and then take preempting decision based on availability of enough low priority requests which can be preempted to gain space for the incoming request. Figure 5.3 graph is a graph on memory availability against time on all hosts in the cloud.

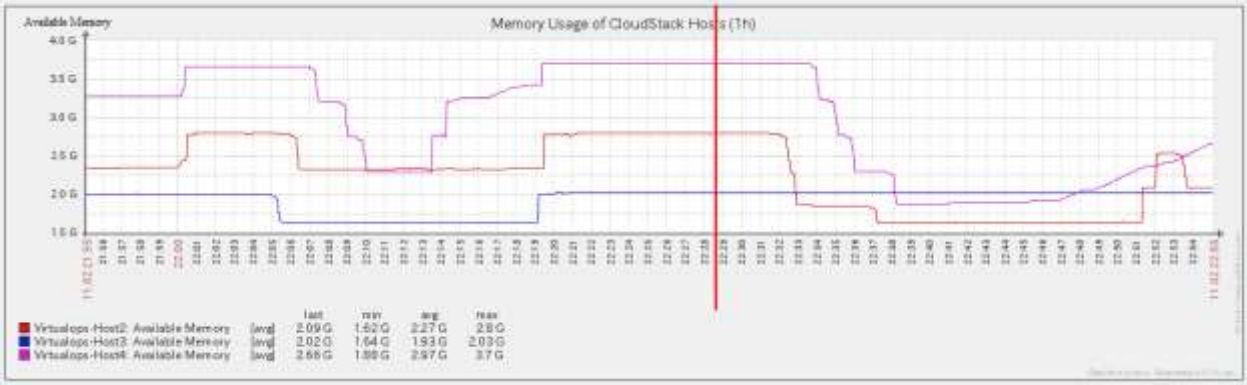

Figure 5.3 – Zabbix Host Memory Graph for Preemptive Scheduling.

When we consider the section after the red vertical line, resource scheduler has first created two VMs on host 2. Then it has created 3 VMs on host 4. Following those, it has further created one VM on host 2 and another one on host 4. Now we are in a situation on where we do not have sufficient memory capacity for more requests asking more than 2GB of memory. Then we get a request asking 2GB of memory. At this point resource scheduler has taken decision to preempt two VMs from host 2 which are consuming 2GB of memory. And then the same host is then used to allocate the incoming request. We can identify that before the preemption, there are 3 VMs running on host 2 and 4 VMs running on host 4. Resource Scheduler has chosen host 2 for preemption because there are less number of VMs running. This algorithm can further be improved to consider the resource utilization by each VM. This is currently not possible because Zabbix agent based resource monitoring requires each VM to run Zabbix Agent to achieve this.

5.3 Migration Scheduling

We have then implemented Migration Scheduling and integrated into the system. After adding migration scheduling, when sufficient resources are not available for an incoming request, request is then sent to the Migration Scheduler. In Migration Scheduler, it checks whether sufficient resources can be freed on a certain host for the incoming request by migrating some of its virtual machines to another hosts. If sufficient space can be gained, we perform migration on selected victim VMs and allocate new request on the host in which we gained resources. Figure 5.4 graph from Zabbix Monitoring System shows how migration scheduling works. It shows available memory on three physical cloud hosts.

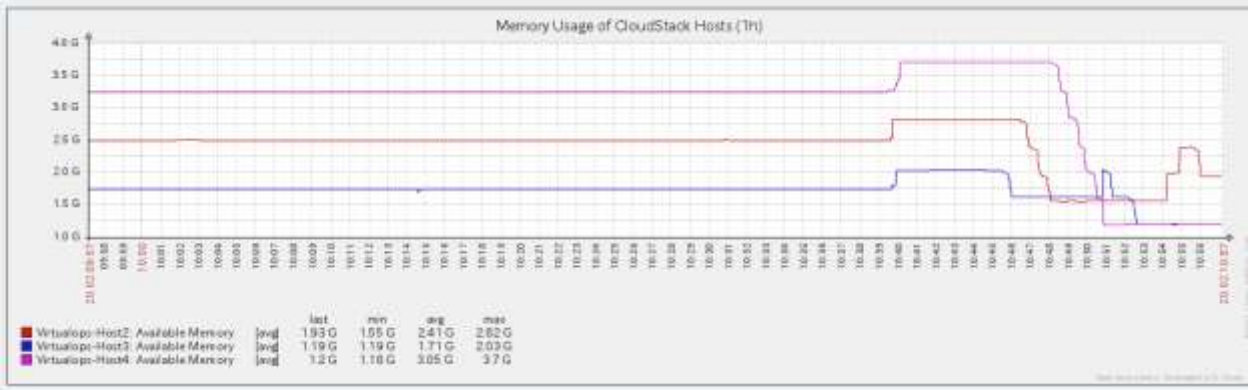

Figure 5.4 – Zabbix Graph for Migration Scheduling.

In this scenario, we have created 1 VMs with memory 1.5GB and 8 VMs with 2GB of memory. Then available memory of all hosts dropped below 2GB. Then we have issues another request asking for a VM with 2GB of memory. At this point, there are no enough resources to perform an allocation for the current request, and the request is passed to the Migration Scheduler. Migration scheduler checks for the host which has maximum memory available and have selected to perform migration scheduling. To allocate the VM with 2GB of memory, it has migrated the VM with 1.5GB of memory to host 4. This frees memory on host 3 and the available memory on host 3 has increased beyond 2GB. Then the allocation is performed on host 2. This scenario can be seen in the graph as a sudden increase of available memory on host 3 and sudden drop of available memory on host 4 at the same time.

Chapter 6

6 Discussion

In this section, we will discuss the problems that we encountered while researching on a solution and during the implementation of the solution and possible future improvements.

6.1 Challenges

6.1.1 No support for KVM VM memory snapshots in CloudStack

We went through several online resources including documentation related to CloudStack and they have clearly stated that CloudStack provides VM memory snapshot support for KVM hypervisor. But once we have completed installation and configuration of CloudStack with KVM, we experienced that CloudStack does not support VM memory snapshots, but only the VM disk snapshots. Also in CloudStack 4.4, they have even dropped the KVM disk snapshot support. We had previously moved from CloudStack 4.3 to 4.4 due to a bug in VM deployment. But with lack of support for memory snapshot in VMs in version 4.4, we had to devise a different mechanism to achieve this since storing VM state to be restored later is a key goal in our project.

To overcome this problem and achieve our goal in a different way, we had to directly call KVM hypervisor using libvirt API. To access KVM through libvirt, we are using *virsh* command line tool. For this purpose, we have implemented a separate Java REST web service which performs taking VM snapshots with memory when a list of VM instance names were given. CloudStack monitors changes in VM state in hypervisor layer and changes VM state in CloudStack and its database. This feature was very important for us since we noticed that after a VM snapshot is taken, VM is stopped by KVM hypervisor. Change of state of VM from running state to stopped state is immediately detected by CloudStack and it shows the change of state in UI and updates the database as well. Since VM is stopped after taking snapshot, resources utilized by VM are immediately released and we could achieve preemption in a single command to KVM.

6.1.2 Issues in setting up cloud infrastructure

Hardware Virtualization was switched off by default on physical hosts. Kernel-based Virtual Machines (KVM) needs hardware assisted virtualization. This created a problem, still we could set up the Cloudstack agent. Cloudstack agent was complaining that hardware virtualization is switched off so we needed to access the server physically and then enabled the hardware virtualization in BIOS.

There were issues with the Hypervisor where Cloudstack-agent duplicating the libvirt package. This made the CloudStack agent to diminish the connection with the Cloudstack management server. Solution to the problem came after five days of research and most of the google searches claimed that this is a bug. There was one such machine which didn't have Libvirt previously where the connection was successful where it leads in to more research and find the problem.

Our infrastructure consist of four physical hosts where one physical host running the Cloudstack management server and other three machines running the Cloudstack agent. There was time skew between these physical hosts and the Cloudstack clients could not connect to the server. This could only identified by reading Cloudstack agent and Cloudstack management server logs and software like open ntpd solved the issue.

System VM's are essential piece in CloudStack where it holds the core logic in CloudStack and also it is responsible for downloading ISO images. The template which was used before was not compatible with the CloudStack installation. The problem continued for a week and we had to update our Cloudstack installation to a higher version, which solved the issue but introduced new bugs like freezing of User interfaces where we could not create virtual machines through the user interface. The compatible version resolved the issue and system VM's started running.

CloudStack agent could not mount the mount point provided in the NFS server. So the agent could not access the ISO images where it needed to create Virtual machines. This issue was tagged as a bug in Cloudstack threads and didn't provide any help to solve the issue. System VM's were running properly ISO images getting downloaded so it was quite an issue and hard to infer that the NFS server had an issue. But client logs had issues mounting the ISO image to the local machine where this forced to change the configurations of the NFS server where ultimately it solved the issues. There were different network configurations supported by Cloudstack and it was important to identify which will work to create the infrastructure with a single cluster. It needed trial and error where different network configurations broke the connection with the physical host where SSH connection stopped. This stopped the progress of setting up the infrastructure completely because server room was closed on weekends. Power failures were also forced system re-deployment where the system wasn't fault tolerant.

6.2 Future work

6.2.1 Support for Advanced Reservation

Currently we have only supported for immediate and best effort resource reservations. Immediate reservation is for the users with higher priority and best effort service will be provided to low priority users such as students. In best effort reservations, Resource Allocation Requests will be queued and will only be allocated if there's enough space on the cloud for them. In contrast, immediate reservations will be served as soon as possible by using VM preemption. Besides these two methods, Advanced Reservation is another resource reservation strategy in which a user can specify future time slot when the resource allocation should be performed. For an example, a student may need a specified VM during a specific time period in the future. To support this kind of advanced reservations, we need to store resource requests in the database and trigger resource allocation requests when those need to be allocated. In our implementation, we currently do not support for advanced reservations and it will be a future improvement.

6.2.2 VM Cloning

When there is a lab to be scheduled, we need to deploy multiple Virtual Machines with the same configuration. To make this easier than multiple VM deployments via CloudStack, VM cloning can be used in the hypervisor layer to deploy multiple virtual machines as a binary tree type deployment. KVM itself support VM cloning with libvirt API and we can use it to deploy VMs without calling CloudStack API. This would be a future improvement in our solution.

6.2.3 Migration support for VMs with local storage

CloudStack support two types of storages, local storage and shared storage. In local storage VM disk is stored in the same host attached to the VM. In shared storage, VM disk will be created in shared primary storage pool but attached to the VM with NFS or ISCSI. Live Migration of VMs with shared storage is feasible within CloudStack itself. But with local storage, CloudStack does not support live migration for VMs since VM need to be stopped before migrating. To handle this scenario, we can take a VM snapshot with memory using KVM hypervisor itself, then restore the snapshot on a different host as we decide.

Chapter 7

Summary

The objective of our project is to build a resource aware, policy-based virtual machine scheduling solution for a medium-scale private cloud. The proposed solution targets an IaaS service model. …

We have explained open source cloud platform technologies such as CloudStack, OpenStack and Eucalyptus. We concluded that CloudStack is more appropriate to be integrated with our solution because its monolithic architecture is more suitable for medium-scale clouds. Also, CloudStack's compatibility with Amazon EC2 [14] API supports numerous third-party tools which were developed to support EC2 can be also integrated into CloudStack. We have also discussed several hypervisors to work with IaaS frameworks. Among them we discussed KVM hypervisor's compatibility with CloudStack and advantages of using it including live snapshot support.

Resource monitoring is one of the key problems that a resource scheduling scheme should address in order to understand current resource utilization in a resource pool. To achieve resource monitoring problem, we have studied several resource monitoring systems including Zabbix, Ganglia and Nagios. When we compare these three resource monitoring systems, we saw that Zabbix is more preferable for our solution as it is easy to set up and configure. Also, Zabbix API provides a friendly programmable interface supporting third party tools to be integrated. This API provides ability to configure Zabbix and monitor resource utilization of a set of hosts programmatically.

We discussed about the current solutions which are bit similar to the problem that we are trying to solve. VM consolidation framework [29] tries to minimize the power used by the datacenter and it consolidates VM's into physical hosts where the power to operate the datacenter at a given time is minimized. We have also discussed about VMware DRS and VMware DPM which are commercial products which respectively balance resource allocations among physical hosts and manages power utilization. Paper [9] discusses about the algorithms where it could increase the acceptance rate of a resource request.

A programmable method to describe cloud resources is important because, resource describing creates the interface between a resource scheduler and a cloud user. To achieve this, we have discussed the need of an appropriate resource description language such as SWORD in PlanetLab. This resource description language needs to be understood both by the cloud user and the resource scheduler.

References

[1] Dialogic Inc., “Introduction to Cloud Computing”. [Online]. Available: http://www.dialogic.com/~/media/products/docs/whitepapers/12023-cloud-computing-wp.pdf. [Accessed: 04- Nov- 2014]

[2] Avnet Technologies Solutions, “Storage, Data Management, Backup & Recovery | Avnet ATS”. [Online]. Available: http://www.ats.avnet.com.hk/storage-data-management-backup-recovery. [Accessed: 05- Nov- 2014].

[3] Citrix Inc., 'Paravirtualization (PV) - Xen', 2013. [Online]. Available: http://wiki.xenproject.org/wiki/Paravirtualization_%28PV%29. [Accessed: 04- Nov- 2014].

[4] Citrix Inc., 'Hypervisor'. [Online]. Available: http://www.xenproject.org/developers/teams/hypervisor.html. [Accessed: 05- Nov- 2014].

[5] "Hardware-assisted Virtualization." Wikipedia. Wikimedia Foundation, 24 Oct. 2014. Web. 30 Oct. 2014. http://en.wikipedia.org/wiki/Hardware-assisted_virtualization

[6] Linux-kvm.org, 'Main Page - KVM'. [Online]. Available: http://www.linux-kvm.org/page/Main_Page. [Accessed: 05- Nov- 2014].

[7] "VMware ESX." Wikipedia. Wikimedia Foundation, 30 Oct. 2014. Web. 30 Oct. 2014. http://en.wikipedia.org/wiki/VMware_ESX

[8] “VMware Distributed Resource Scheduler (DRS): Dynamic Load Balancing and Resource Allocation for Virtual Machines”. VMware, Inc. 2009

[9] M. K. Nivodhini, K. Kousalya, and S. Malliga, S., “Algorithms to improve scheduling techniques in IaaS cloud,” Int. Conf. on Information Communication and Embedded Systems (ICICES), Feb. 2013, pp. 246-250.

[10] OpenStack.org, 'OpenStack Open Source Cloud Computing Software'. [Online]. Available: http://www.OpenStack.org/. [Accessed: 04- Nov- 2014].

[11] Rackspace Hosting, 'Rackspace: We manage your cloud services. You run your business.’ [Online]. Available: http://www.rackspace.com/. [Accessed: 04- Nov- 2014].

[12] NASA, 'NASA'. [Online]. Available: http://www.nasa.gov/. [Accessed: 04- Nov- 2014].

[13] Docs.OpenStack.org, 'Conceptual architecture - OpenStack Cloud Administrator Guide - current'. [Online]. Available: http://docs.OpenStack.org/admin-guide-cloud/content/conceptual-architecture.html. [Accessed: 07- Nov- 2014].

[14] Docs.aws.amazon.com, 'What Is Amazon EC2? - Amazon Elastic Compute Cloud'. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html. [Accessed: 05- Nov- 2014].

[15] Docs.aws.amazon.com, 'Instance Types - Amazon Elastic Compute Cloud'. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html. [Accessed: 05- Nov- 2014].

[16] D. Nurmi *et al*, "The Eucalyptus Open-Source Cloud-Computing System," Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on, vol., no., pp.124, 131, May 2009

[17] von Laszewski *et al.*, "Comparison of Multiple Cloud Frameworks," 5th IEEE Int. Conf. on Cloud Computing (CLOUD 2012), pp.734, 741, June 2012.

[18] Docs.CloudStack.apache.org, "Welcome to CloudStack Documentation! — Apache CloudStack 4.3.0 documentation". [Online]. Available: http://docs.CloudStack.apache.org/en/master/. [Accessed: 05- Nov- 2014].

[19] Docs.OpenStack.org, 'Swift Rings', 2014. [Online]. Available: http://docs.OpenStack.org/developer/swift/overview_ring.html. [Accessed: 04- Nov- 2014].

[20] Redhat, 'Resource Monitoring'. [Online]. Available: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Introduction_To_System_Administration/ch-resource.html. [Accessed: 04- Nov- 2014].

[21] Ganglia, 'Ganglia Monitoring System'. [Online]. Available: http://ganglia.sourceforge.net/. [Accessed: 04- Nov- 2014].

[22] Matthew L. Massie, Brent N. Chun, David E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience". 15 Jun. 2004

[23] Nagios, 'Nagios - Nagios Overview'. [Online]. Available: http://www.nagios.org/about/overview/. [Accessed: 04- Nov- 2014].

[24] Zabbix SIA, 'Zabbix:: An Enterprise-Class Open Source Distributed Monitoring Solution', Zabbix.com. [Online]. Available: http://www.zabbix.com/. [Accessed: 07- Nov- 2014].

[25] V.Vinothina, Dr.R.Sridaran, Dr.R.Sridaran, A Survey on Resource Allocation Strategies in Cloud Computing, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol.3, No.6, June2012

[26] Robert Ricci *et al.*, Lessons from Resource Allocators for Large-Scale Multiuser Testbeds, News Letter ACM SIGOPS Operating system review, Volume 40 Issue 1, January 2006 Pages 25 - 32

[27] Planet Labs Inc., 'SWORD on PlanetLab: Scalable Wide-Area Resource Discovery'. [Online]. Available: http://sword.cs.williams.edu/. [Accessed: 04- Nov- 2014].

[28] Fetahi Wuhib, Rolf Stadler and Hans Lindgren, Dynamic Resource Allocation with Management Objectives—Implementation for an OpenStack Cloud, Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualization management (svm), 22-26 Oct. 2012

[29] T. Janpan, V. Visoottiviseth, R. Takano, "A virtual machine consolidation framework for CloudStack platforms," Information Networking (ICOIN), 2014 International Conference on, vol., no., pp.28, 33, 10-12 Feb. 2014

[30] J. Napper,and P. Bientinesi, Can Cloud Computing Reach the TOP500?, Proceedings of the Workshop on UnConventional High Performance Computing, in conjunction with The ACM International Conference on Computing Frontiers, 18-20 May 2009

[31] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in datacenter networks". SIGCOMM Comput. Commun. Rev., v. 39, n. 1, pp. 68-73, 2008

[32] P. T. Endo, A. V. A. Palhares, N. N. Pereira, G. E. Gonçalves, D. Sadok, J. Kelner, B. Melander, J. E. Mangs, "Resource allocation for distributed cloud: concepts and research challenges", IEEE Network Magazine, vol.25, pp. 42-46, July 2011

[33] G. Goncalves *et al.*, "CloudML: An Integrated Language for Resource, Service and Request Description for D-Clouds," Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on , vol., no., pp.399,406, Nov. 29 2011-Dec. 1 2011

[34] D. Kranowski, 'CloudStack vs OpenStack vs Eucalyptus', JavaOne Conference, 2012.

[35] Novell Inc., (n.d.). Novell Doc: Virtualization: Getting Started - Xen Virtualization Architecture. [online] Available at: https://www.novell.com/documentation/vmserver/virtualization_basics/data/ba0khrq.html [Accessed 7 Nov. 2014]

[36] IBM Inc., (n.d.). Hypervisors, virtualization, and the cloud: Dive into the KVM hypervisor. [Online] Available at: http://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare-kvm/ [Accessed 7 Nov. 2014]

[37] Red Hat Inc., (n.d.). 2.2. Red Hat Virtualization Hypervisor. [Online] Available at: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Virtualization/3.0/html/Technical_Reference_Guide/sect-Technical_Reference_Guide-Architecture-Red_Hat_Virtualization_Hypervisor.html [Accessed 7 Nov. 2014]

[38] IT 2.0, (n.d.). A brief architecture overview of VMware ESX, XEN and MS Viridian. [Online] Available at: http://it20.info/2007/06/a-brief-architecture-overview-of-vmware-esx-xen-and-ms-viridian/ [Accessed 7 Nov. 2014]

[39] Amazon Web Services, Inc., (n.d.). AWS Case Study: Parse. [Online] Available at: http://aws.amazon.com/solutions/case-studies/parse/ [Accessed 7 Nov. 2014]

[40] Doc.monitoring-fr.org, (n.d.). Chapitre 41. Supervision distribuée. [Online] Available at: http://doc.monitoring-fr.org/3_0/html/advancedtopics-distributed.html [Accessed 7 Nov. 2014]

[41] J. Ramsaran, 'Cloud Computing: Benefits and Challenges', Transform Customers. [Online]. Available: http://transformcustomers.com/cloud-computing-benefits-and-challenges/. [Accessed: 07- Nov- 2014].

[42] Philippe Scoffoni, 'CloudStack 3, an open source cloud orchestration solution by Citrix @pscoffoni - Philippe Scoffoni'. [Online]. Available: http://philippe.scoffoni.net/CloudStack3-solution-open-source-orchestration-cloud-citrix/. [Accessed: 07- Nov- 2014].

[43] kjkoster.org, 'Zabbix Architecture'. [Online]. Available: http://www.kjkoster.org/zapcat/Architecture.html. [Accessed: 07- Nov- 2014].

[44] Mongodb.org, 'MongoDB', 2015. [Online]. Available: http://Mongodb.org. [Accessed: 12- Feb- 2015].

[45] Nodejs.org, 'Node.js', 2015. [Online]. Available: http://Nodejs.org. [Accessed: 12- Feb- 2015].

[46] Fielding, Roy T., and Richard N. Taylor. 'Principled Design Of The Modern Web Architecture'.*Proceedings of the 22nd international conference on Software engineering - ICSE '00* (2000): n. pag. Web. 11 Feb. 2015.

[47] Wikipedia, 'Secure Hash Algorithm', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Secure_Hash_Algorithm. [Accessed: 12- Feb- 2015]

[48] Json.org, 'JSON', 2015. [Online]. Available: http://Json.org. [Accessed: 12- Feb- 2015]. [49] Wikipedia,. 'Moving Average'. N.p., 2015. Web. 11 Feb. 2015.

[50] Libvirt.org, 'libvirt: The virtualization API', 2015. [Online]. Available: http://Libvirt.org. [Accessed: 12- Feb- 2015].

[51] Git-scm.com, 'Git', 2015. [Online]. Available: http://Git-scm.com. [Accessed: 12- Feb- 2015].