Chapter 3

Open Source Hypervisors and Cloud Architectures

Chapter abstract Cloud computing has been emerging as one of the prominent technologies to the end users which offers pay-as-you-go model with the help of various underlying technologies. One of such technologies is virtualization that forms the foundation for Cloud environment. Virtualization makes it possible to run multiple computing resources (in form of Virtual Machines - VMs) on the same (physical) server at same time, concurrently. Hypervisors, also known as Virtual Machine Monitors (VMM), create, manage and run virtual machines on physical machines. There are many open source hypervisors available viz. KVM, Xen, OpenVZ, VirtualBox, Lguest, LXD etc. In addition, there are many Cloud-computing software platforms available that manage the provisioning of VMs for the Cloud provider. Compatibility issues of these VMM and Cloud providers are required to be handled for smooth functioning the Cloud. This chapter aims to compare and analyze the most frequently used open source hypervisors and cloud architectures.

3.1 Introduction

A revolution in grid computing introduced a new dimension in technology known as Cloud Computing over a last decade. It satisfies the end users’ need such as applications, software platforms or infrastructures over the Internet. It reduces the burden of purchasing
and maintaining software or infrastructures of the end users and industries as they are available in Cloud on pay-as-you-go model. Rather than considering the Cloud computing as a new technology, it should be treated as a new operational model that brings together a set of existing technologies to run business in a different way (Zhang et al. (2010)). It uses the existing technologies like grid computing, utility computing, virtualization and autonomic computing to facilitate the end users by combining the features of all-in-one to realize resource sharing and dynamic resource provisioning (Zhang et al. (2010)). All these attributes make the paradigm popular among the researchers, developers and the end users community. The services provided by Cloud technology based on subscriptions are divided into three categories viz. Infrastructure as a Service(IaaS), Platform as a Service(PaaS) and Software as a Service(SaaS) (Sosinsky (2010)). IaaS is one of the frequently and commonly used services among the Cloud users. It consists of a large number of server, disk, memory units to provide service to millions of end users (Jing et al. (2013)).

Cloud computing has a four layered architecture (Zhang et al. (2010)) as shown in fig 3.1. Each layer has clearly identified responsibilities. First is the hardware layer, a lowest layer responsible for managing hardware resources like server, switches, power, cooling system etc. Second is the infrastructure layer, which also known as virtualization layer responsible for creating a storage and computing resources by dividing the physical resources using virtualization technologies. The infrastructure layer plays an important role since many key features, such as dynamic resource assignments are only available through virtualization technologies (Zhang et al. (2010)). Virtulization of underlying hardware and software is done by creating a image/instance of physical machine called as virtual machine. Virtual Machine Monitor(VMM) is responsible for creation of virtual machine. Third layer is platform layer which consists of operating systems and application frameworks. The upper most forth layer is application layer which consists of the actual Cloud applications (Zhang et al. (2010)). All these layers are loosely coupled i.e this layered architecture handles the maintenance and management overhead of Cloud computing. The different services like infras-
Based on the idea of deployment models, the Cloud computing can further be categorized into four categories viz. public (Zhang et al. (2010)), private (Armbrust et al. (2010)), hybrid (Zhang et al. (2010)) and community Cloud (Sosinsky (2010)). When these deployment models are made available to general public in pay per use manner, it is called public Cloud (Zhang et al. (2010)). Amazon Elastic Compute Cloud (EC2) (EC2 (2015)), IBM’s Blue Cloud (IBM (2015)), Sun Cloud, Google AppEngine (Googleappengine (2015)) and Windows Azure Services (azure (2015)) are some of the example of public Cloud platforms. While an internal data-center of organization only used by a organization not available in general can be known as private Cloud (Armbrust et al. (2010)). A hybrid Cloud is a combination of both, public and private Cloud. In hybrid Cloud, the organizations rely on public Cloud as needed (Zhang et al. (2010)). Cloud organized/deployed to satisfy a com-
mon purpose/aim is called as community Cloud. Figure 3.2 shows the deployment location of different Clouds.

Virtualization is considered as a most important technology from the number of underlying technologies, services and infrastructure-level configurations that make the Cloud computing possible. In a simple term, virtualization can be defined as a mechanism which abstracts the hardware and the system resources of operating system (Zhang et al. (2010)). These abstraction and virtualization are carried out across a pool of servers by Hypervisor or VMM (Younge et al. (2011)), which provide multiple isolated execution environments. It is installed on server and it provides a virtual operating platform to the guest operating systems (guest OS) and observes execution of guest OS (Sabahi (2012b)). There are several ways to implement a virtualization. Most of the virtualization strategies fall into four main strategies like full virtualization, para-virtualization, Operating System-level Virtualization and Native Virtualization (Walters et al. (2008)).

Overview of all this virtualization techniques are as follow:

- Full virtualization: In full virtualization, VMM will give the image of physical machine to the each VM request. In this
way it executes the unmodified OS to execute the privileged instructions. It is successful as it provides isolation of users from each other and from the control program (Mishra & Jaiswal (2012)). Some examples of full virtualization are QEMU, Microsoft Hyper-V, Oracle VirtualBox, VMware Workstation, VMware Server, Microsoft Virtual PC.

- Para virtualization: In para-virtualization, each VM has an abstraction of the hardware that is similar but distinguishable to the underlying physical hardware. Guest operating systems are modified to execute VMs. As a result, the guest operating systems are executing on a VM to provide a near-native performance. Para-virtualization methods are still being developed and thus, have limitations including several security issues like the guest OS cache data, unauthenticated connections and “guest” OS needs to be modified (Abels et al. (2005)). Disaster recovery, migration, capacity management are the benefits of para-virtualization (Mishra & Jaiswal (2012)). Some examples of para-virtualization are Xen, KVM, UML VM.

- OS-level Virtualization: Unlike both para-virtualization and full virtualization, OS-level virtualization does not depend on a hypervisor. Instead, it modifies the operating system securely to isolate multiple instances of an operating system within a host machine. The advantage to OS-level virtualization is performance of near-native speeds. No hypervisor/instruction is required. Some examples of OS-level virtualization are OpenVZ and LXC.

- Native Virtualization: In native virtualization, multiple unmodified operating systems are allowed to run alongside one another. In this technique operating systems are capable of running on the host processor directly. That is, in native virtualization, it does not emulate a processor. Unlike the full virtualization technique where it is possible to run an operating system on a fictional processor, it gives full part of processor to unmodified OS, which leads to a poor performance. Some examples of native virtualization are Intel and AMD supports virtualization through the Intel-VT and AMD-V virtualization extensions (Walters et al. (2008)).
However, two leading ways two implement server virtualization by VMM/hypervisor are full virtualization (Walters et al. (2008)) and para-virtualization (Abels et al. (2005)).

Also, There are three VMM/hypervisor models as shown in fig 3.3.

- Type 2 VMM.
- Hybrid VMM
- Type 1 VMM

![Figure 3.3: Hypervisor](image)

The type 2 VMM runs within a host operating system. All guest environments are operated above the level of host OS. Examples of these guest environments include the Java Virtual Machine and Microsoft’s Common Language Runtime. The hybrid model is used to implement Virtual PC, Virtual Server and VM Ware GSX. These depend on a host operating system that shares control of the hardware with the VMM. The type 1 VMM controls the guest operating systems instance execute at a upper level of it with hardware. Xen and Windows Server virtualization are the examples of type 1 hypervisor. The vital component of any virtualization technique is the choice of hypervisor (Fenn et al. (2009)). The different VMMs provide a framework which allows VMs to run. Each of these Cloud frameworks with VMM relies on a library called libvirt, which is designed to control the start and stop of VMs. However, this abstraction is not fixed, as there is unique options for each VMM that needs to be set. For every different VMM and its version, there is a different inputs to libvirt library. For this and other reasons, the different Cloud frameworks support different subsets of the hypervisors (Sempolinski & Thain (2010)). VM performance is a crucial component of overall Cloud performance. Once a hypervisor has
added overhead a higher layer cannot remove it, so it reflects to the overall Cloud performance. An alternative to virtualization in the Cloud is provided by Container-based virtualization (Soltesz et al. (2007)). As Cloud can be considered as a pool of servers where the end user asks for the services with desired VM configuration and it is available on pay per use basis. If the Cloud paid services are used for the organization with the large number of users, it will be more cost effective then implementing a private Cloud. Open source frameworks are contemplate while selecting the open source which must be flexible enough to work with many underlying systems (Sempolinski & Thain (2010)). Whereas, commercial Clouds only need their system to work with the hardware that they have. Therefore, open source provides freedom to choose the best option. The role of Cloud hypervisors/VMM and Cloud architecture in Cloud environment can be viewed from the figure 3.4. In the figure Cloud environment is shown with open source hypervisors and Cloud architectures. Hypervisor creates VM in Cloud while Cloud architecture will allocates VM in Cloud. Here, the study is

Figure 3.4: Cloud environment

carried out with the motto of answering the following questions for selecting best option:
– Suitability of different VMM in Cloud computing.
– Compatibility of Cloud platforms with VMM.

Therefore, in the present chapter, firstly, we briefly address the basic architecture of open source VMM with different container based virtualization technologies and different open source Cloud platforms following comparisons among them.

The rest of the chapter is organized as follows. Section 2 describes various Hypervisor/VMM with comparative study followed by different open source Cloud architectures which enlightens the compatibility of hypervisors with Cloud architecture in Section 3, followed by conclusion and future work in section 4.

### 3.2 Hypervisors

In the previous literature reviews (Deshane et al. (2008), Kivity et al. (2007), Barham et al. (2003), Cherkasova & Gardner (2005), Cherkasova et al. (2007), Barham et al. (2003), Kiszka et al. (2010), Abels et al. (2005)), some of the open source hypervisors (like Xen, KVM etc.) are discussed either separately or they are compared on some characteristics like performance, scalability etc. This section includes discussion on few open source hypervisors available viz. KVM, Xen, OpenVZ, VirtualBox, Lguest, LXD, with the container based virtualization technology like OpenVZ and LXC and then we compare all these virtualization technologies by introducing more attributes. Following section discussion covered on open source VMM called KVM, Xen, OpenVZ and VirtualBox, Lguest, LXD, with the container based virtualization technology like OpenVZ and LXC.

(a) **KVM (Kivity et al. (2007))**: KVM is fully integrated virtualized, simple and easy to use, solution for Linux. Its integration to Linux allows the usage of the large set of Linux features set with maximum advantage. It is attached in the mainstream Linux kernel since version 2.6.20. Device node (/dev/kvm) is open to create Virtual Machine in KVM (Franssens et al. (2013)). The default scheduler for KVM is the Linux priority-based scheduler (Gu & Zhao (2012)). As KVM is a kernel
module, Linux scheduler schedules VMs as regular processes. Also, KVM can be introduced as an extension of generic and open source machine emulator QEMU with support for the x86 VT extensions, in which virtual machines can make system calls without unnecessarily invoking the host kernel. KVM requires a recent Linux kernel with the KVM modules enabled. Virtualization system used by KVM differs from other virtualization technologies that require heavily modified kernels, whose development is not often current with the mainline kernel (Fenn et al. (2009)). It is a type 1 VMM (type 2015).

**KVM Architecture:**
As shown in KVM architecture in figure 3.5, processes create virtual machines. The integration of key virtualization technology at the processor level by both Intel (Intel VT) and AMD (AMD-V) have enabled virtualization to be deeply integrated at the Linux kernel level which creates many significant benefits of KVM in terms of performance, scalability, and security. It also incorporates Linux security features including SELinux (Security-Enhanced Linux) to add access controls, multi-level and multi-category security as well as policy enforcement.

(b) Xen (Barham et al. (2003)): Xen is an open source type 1 VMM
which uses paravirtualized techniques for virtualization (Abels et al. (2005)). It is being integrated by a number of users like RedHat, Suse, XenSource and Virtual Iron (Abels et al. (2005)).

**Xen Architecture:** Xen architecture is shown in figure 3.6. As it is shown, the architecture of Xen 3.0 hosting two VMs called Domain 0, VM 1. This architecture includes the Xen VMM, which virtualizes the underlying physical hardware to provides hardware access for the different virtual machines. The VM called Domain 0 has a special privileges. It can only access the control interface of the VMM, through which other VMs can be created, destroyed, and managed. Software for Management and control runs in Domain 0. Administrators can create virtual machines with different defined privileges such as direct access of hardware. Xen has a small memory as it uses a micro kernel design and restricted interface to guest, which make it
more secure and robust compared to other hypervisors.

(c) LXD (LXD (2015)): LXD is the next-generation of container hypervisor for Linux from Canonical. It aims to give rapid provisioning of resources and instant guest boot. It supports a full virtualization technology. It is compatible with OpenStack and is a most suitable pair for Linux-oriented private Cloud. It does not support Live migration. Architecture of LXD is same as LXC like it has containers, container snapshots and images. Refer figure 3.9 for the same.

(d) Lguest (Zaidenberg (2012)): Lguest is simplified Linux-on-Linux para-virtualization hypervisor that allows the execution of multiple Linux kernels on the top of a system that runs Linux kernel. It develops to serve as an educational hypervisor which supports only x86 environments. It allows Linux kernel to run internally in Lguest virtualization environment as a process runs in user space.

**Architecture:** High level architectural overview is shown in fig 3.7. As shown in figure, architecture can be defined with

---

**Figure 3.7: Lguest**

![Diagram of Lguest architecture](image)
two spaces: host kernel space and host user space. It has five major components: Launcher, /dev/lguest, guest kernel, host kernel, switcher. The role of each of this components are as follow:

- Launcher: It is a host user space program which sets up, runs and services the guest.
- /dev/lguest: Launcher uses a /dev/lguest as a read and write character device to perform read, write operations.
- Guest kernel: It handles the guest kernel implementation.
- Host kernel: It handles the different guest kernels.
- Switcher: Switcher is the program code required to run when the switch-over required between main memory and virtual memory.

Lguest has benefits of having simple, easy to understand and stable architecture which make it popular for educational purpose.

(e) VirtualBox (Vbox (2015)): VirtualBox was originally developed by Innotek, which later on acquired by Sun Microsystems in 2008. It is an X86 virtualization tools that implements full virtualization. VirtualBox runs on a large number of 32-bit and 64-bit host operating systems. It is a so-called “hosted” hypervisor which allows to run virtual machines created on one host and execute on another host with a different operating system. It allows executing an guest OS without modification (Fuertes & de Vergara (2007)). VirtualBox provides a group features that enables the user to organize and control virtual machines individually and in group also. It has different builds for different host operating systems like Windows, Mac OS X, Linux and Solaris hosts (Li (2010)). Thus, VirtualBox can be called as general purpose full virtualizer for server, desktop and embedded use.

**Container based Virtualization technologies:**

(a) OpenVZ(Kovári & Dukan (2012)): Based upon container technology, OpenVZ is not a true virtualization application. It uses OS-level virtualization technology in which multiple isolated execution environments are available within a single operating
system kernel. This technology does not allow to run different kernels from different OSs at the same time. OpenVZ is container-based virtualization for Linux. So, it has a modified Linux kernel. Also, it permit a physical server to execute and control multiple isolated operating system instances, known as containers, Virtual Private Servers (VPSs), or Virtual Environments (VEs). As it creates multiple secure and isolated container on single physical machine, each container executes like a stand alone server. OpenVZ has a limitation where it can requires both the host and guest OS to be Linux. OpenVZ manage the resources by handling user bean counters, I/O schedulers, disk schedulers and CPU schedulers. It do not requires a reboot when these resources needed to be change during run time, which is an attractive feature for developers and testers. OpenNode virtualization Platform can be used with OpenVZ. As shown in figure 3.8, OpenVZ creates many isolated, independent instance of operating system known as virtual environments or containers. It is more faster and efficient as it does not have any hypervisors overhead (Bazargan et al. (2012)).

(b) LXC (LXC (2015)): Like OpenVZ, LXC is a container technology, existing as a user space interface for the containment features in the Linux kernel. It does not provide a virtual machine, but rather provides a virtual environment that has its own CPU, memory, block I/O, network, etc. However, As LXC takes root in the host kernel, there is no need for a separate kernel. Like OpenVZ, LXC uses the resource management and check pointing of the host kernel. It composed of a variety of container templates, different containers managing tools, different bindings for the liblxc library (libvirt is considered an alternative library), and multiple languages (Ruby, Python, Go, Lua, etc). It achieve better performance than OpenVZ (Soltesz et al. (2007)). It has less I/O overhead and more storage capacity as it has operating system level virtualization (LXCL (2015)). As shown in figure 3.9 only one kernel execute on the host, not a different kernels for each virtual machines as Xen/KVM. It uses a chroot and linux control group at kernel
Discussion on VMM:

Firstly, we will discuss the widely used open source hypervisors- Xen and KVM and then focus on other too. During analyzing literature available (Younge et al. (2011)), it has been identified that experiments performed for HPC (High Performance Computing) shows that Xen is suitable and most widely used hypervisor, especially within academic Clouds and grids where, KVM is identified as optimal choice for general deployment in an HPC environment. The most striking difference between the two systems is in scalability. KVM had significant problems with guests crashing, as the number of guest increased (Deshane et al. (2008)). While, KVM does not face that problem of scalability. However, KVM had better performance isolation than Xen, but Xen’s isolation properties are quite good. Performance wise, Xen was found lacking as compared to KVM. It is not found as the best choice for enduring quality of service on infrastructure. However, as Xen is using para-virtualization technique, it does not depend on any type of CPU instructions for
virtualization support. As all supervisors calls are replaced by hypercalls into Xen hypervisor and thus, the guest operating system (OS) is modified. Still in comparison with Xen it has been found that KVM is a more promising hypervisor for grid computing. It is simple to maintain and easy to deploy (compared to Xen) and also gives excellent performance for many tedious jobs. Comparing Lguest with other hypervisors, it has been found that it is slower than other hypervisors, though sometimes only it is noticeably because of workload. In case of VirtualBox, it has a limitation that it allows only 16GB maximum memory allotment for individual guest VMs, which actually limit the use of VirtualBox in HPC environment. Also, OpenVZ and LXC can not use in heterogeneous environment as it require Linux as both guest and host OS. Summary of a discussion on VMM is shown in table3.1.
### Table 3.1: Comparison of Cloud Hypervisors

<table>
<thead>
<tr>
<th>Hypervisor Name</th>
<th>Xen</th>
<th>KVM</th>
<th>LXD</th>
<th>LXC</th>
<th>Lguest</th>
<th>VirtualBox</th>
<th>OpenVZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of VMM</td>
<td>Type-1</td>
<td>Type-1</td>
<td>Container</td>
<td>Container</td>
<td>Type-1</td>
<td>“hosted”</td>
<td>Container</td>
</tr>
<tr>
<td>Type of Server Virtualization</td>
<td>Full virtualization</td>
<td>Para virtualization</td>
<td>Full virtualization</td>
<td>Operating system level virtualization</td>
<td>Para virtualization</td>
<td>Full virtualization</td>
<td>OS- level virtualization</td>
</tr>
<tr>
<td>Required Host OS</td>
<td>Linux</td>
<td>NetBSD, Linux, Solaris</td>
<td>Linux</td>
<td>Linux</td>
<td>Linux</td>
<td>Windows, Linux</td>
<td>Linux</td>
</tr>
<tr>
<td>Guest OS Support</td>
<td>XBSD, Linux, Solaris, Windows</td>
<td>Linux, Windows</td>
<td>Linux</td>
<td>Linux</td>
<td>Linux</td>
<td>Linux and Windows</td>
<td>Linux</td>
</tr>
<tr>
<td>Suitable with heterogeneous Environment</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Service Provider/Cloud platform</td>
<td>Eucalyptus, OpenNebula, Nimbus, Amazon EC2 for on demand service, GoGreed, Flexiscale, Amazon Web service</td>
<td>OpenNebula, Amazon EC2 for on demand service, Enomaly</td>
<td>OpenStack</td>
<td>No hypervisor</td>
<td>Xen, KVM</td>
<td>OpenNebula, Enomaly</td>
<td>No hypervisor</td>
</tr>
</tbody>
</table>
3.3 Open source Cloud Architectures

Open source Cloud computing option is one of the most important features for the Cloud to be seen as a dramatic rise in the academic and business world. We have referred and tried to analyze and compare various open source options (Sempolinski & Thain (2010), Gonçalves et al. (2011), Endo et al. (2010), Sefraoui et al. (2012), Keahey (2009), Nagar & Suman (2014), Kumar et al. (2014), Peng et al. (2009)) of Cloud. In this subsection, we discuss Eucalyptus (Nurmi et al. (2009)), OpenNebula (Sempolinski & Thain (2010)), nimbus (Keahey (2009)) and OpenStack (Sefraoui et al. (2012)). Enomaly (Gonçalves et al. (2011)), Apache VCL (Gonçalves et al. (2011)), TPlatform (Gonçalves et al. (2011)), XCP (Gonçalves et al. (2011)), cloudStack (Kranas et al. (2012)) and Ganeti (Kumar et al. (2014)) are also discussed in detail with its architecture.

(a) Eucalyptus (Nurmi et al. (2009)): Eucalyptus is an open-source software infrastructure for building a Cloud computing on clusters derived from the first letter of “Elastic Utility Computing Architecture for Linking Your Programs to Useful Systems”. It is used to build private, public, or hybrid Clouds. It provides an integrated set of application programming interfaces (APIs) that are compatible with Amazon Web Services, including Amazon’s Elastic Compute Cloud (EC2), Amazon Simple Storage Service (S3), and Amazon Elastic Block Store (EBS) (Hurwitz et al. (2010)).

Architecture: The five components of Eucalyptus are grouped into three separate levels. User can access the system via web interface or its own tool euc2ools for front end interaction (Sempolinski & Thain (2010)). Architecture of Eucalyptus is shown with its components in figure 3.10. The brief explanation of these levels are as follow:

- Cloud Level: It has two components: Cloud Controller (CLC) and Scalable Object Storage (SOS). CLC provides the web user interface (an http server on port 8445) and implements on Amazon EC2 APIs. The CLC accepts user
API requests from command-line interfaces like euca2ools or GUI-based tools like Management Console and manages the underlying computing, storage, and network resources. Only one CLC can exist per Cloud. It handles high-level activities like authentication, accounting, reporting and quota management. Scalable Object Storage (SOS) is equivalent to AWS Simple Storage Service (S3). It provides a basic storage implementation, known as Walrus, which is used to implement S3 APIs and is useful to store VM Images and user storage using S3 bucket put/get abstraction. EC2 provides a virtual computing environment that enables a user to run Linux-based applications (Nagar & Suman (2014)).

- Cluster Level (i.e., Availability Zone): There are two components: Cluster Controller (CC) and Storage Controller (SC). CC is used to manage the collection of resource nodes. The Cluster Controller (CC) acts as the front end for a cluster within an Eucalyptus Cloud and communicates with the Storage Controller (SC) and Node Controller (NC). The CC manages instance (i.e., virtual machines) execution and Service Level Agreements (SLAs) per cluster. DCM is provided by Ubuntu Eucalyptus CC package. The SC communi-
cates with the Cluster Controller (CC) and Node Controller (NC) within the distributed Cloud architecture and manages block volumes and snapshots to the instances within its specific cluster. If an instance requires writing persistent data to memory outside the cluster, and these data requires to write to the back end storage, which is available to any instance in any cluster (Sempolinski & Thain (2010)).

- Node Level: Node Controller (NC) hosts the virtual machine instances and manages the virtual network endpoints. The NC downloads and caches images from Scalable Object Storage as well as creates and caches instances (Sempolinski & Thain (2010)).

(b) OpenNebula: OpenNebula is a flexible tool that coordinate and manage storage, network and virtualization technologies to enable the dynamic placement of services on distributed infrastructures (Gonçalves et al. (2011)). OpenNebula is a pure private Cloud, where user can access Cloud functions by logging into the head node. This interface is a wrapper around an XML-RPC interface, which can also be used directly (Sempolinski & Thain (2010)).

- Architecture: As shown in figure 3.11, the architecture of OpenNebula (Borja et al. (2009)) includes many components to handle the virtualized environment, which are: OpenNebula core, scheduler and pluggable drivers that handle the external Cloud. To manage the life cycle of VM, the OpenNebula core synchronizes storage, network and underlying a hypervisor. The Core performs specific storage, network or virtualization operation through pluggable drivers. The Core is also designed to support deployment of services, which includes a set of inter related components (e.g. web server, DB backend, etc.) requiring several VMs. The core also provide different context information like IP address of the web server, digital certificates and software licenses to the VMs (Borja et al. (2009)). The second important part is scheduler, which is very important as it schedules the VM. The OpenNebula default scheduler uses a rank scheduling policy that places VMs on physical resources according to a
ranking algorithm that is highly configurable by the administrator, and uses a real-time data from both the running VMs and available physical resources to schedule VM. It offers Management Interfaces to integrate the Core functionality such as accounting or monitoring frameworks within other data center management tools. OpenNebula implements the libvirt API and a command line interface (CLI) as an interface for VM management. OpenNebula can also support a hybrid Cloud model by using Cloud drivers to interface with external Clouds.

(c) Nimbus (Keahey (2009)): The Nimbus advertises itself as a “science” Cloud solution (Sempolinski & Thain (2010)). It is introduced as a combination of open source tools, which provides IaaS Cloud computing solutions to users, which provides a remote resources on lease by deploying VMs and configured VM according to user’s requirement. It is formally known as Virtual Workspace Service (VMS). Generally workspace service is a technical component in the software collection. In nimbus, most of the customization is done by the administrator and is not allowed to the users. The programming framework used for nimbus is Java and Python.
Architecture: As shown in figure 3.12

The architecture of Nimbus includes various components. The role of each of these components are as follows:

– Workspace service: Workspace service is the most important component of the architecture, which is responsible for management of virtual machines in data center. Moreover, it receives developers’ requests through different interfaces. It uses the WSRF-based (Web Services Resource Framework) interface and the Amazon EC2-compatible as an interface.

– The workspace control (or VMM): The workspace control performs basic management and control operations of the VMs. So, it needs to be installed on each server.

– Workspace Resource Manager and Workspace Pilot: This components handle automatic management of VMs including selection of appropriate server for each VM.

– The ContextBroker: It allows clients to coordinate large virtual cluster launches automatically and repeatably.

– The ContextAgent: It lives on VMs and interacts with the Context Broker at VM boot.

(d) OpenStack (Sefraoui et al. (2012)): OpenStack is scalable, compatible, flexible (i.e. OpenStack supports most virtualization so-
olutions of the market like ESX, Hyper-V, KVM, LXC, QEMU, UML, Xen and XenServer). It is written in python and implements EC2 and Rackspace as a control APIs. To provide an interface with a maximum number of hypervisors like Xen, KVM, HyperV and Qemu, it uses different drivers. Linux container technology such as LXC is also supported for situations where users wants to minimize virtualization overhead to achieve efficiency and sufficient performance. Moreover, to different hypervisors, OpenStack also supports ARM and many alternative hardware architectures (OpenStack (2015-09-19)).

– Architecture: As shown in figure 3.13

![OpenStack Architecture](image)

Figure 3.13: OpenStack

The architecture of the OpenStack includes major three components known as OpenStack Compute, Image and Object. The role of each of these components are as follows:

– OpenStack Compute: It is also known as Nova. It works as a management platform that controls the infrastructure. It provides an administrative interface and API for the proper coordination in Cloud. It does not require any prerequisite hardware/installation and it is independent to any hypervisor. It has a seven main components included:

  i. API Server: It acts as a web front and service control of the hypervisor.
ii. The Message Queue: Message Queue implements the mechanism for sending the exchanged instructions for smooth communication.

iii. A Compute Controller: It controls the life cycle of instances and responsible of creating and managing virtual servers.

iv. The Component Object Store: It provides storage services.

v. A Volume Controller Component: It controls the volumes.

vi. The network Controller: It is responsible for controlling the network.

vii. Scheduler: It schedules tasks and allocate the virtual server.

- OpenStack Imaging Service: Imaging service provides storage services, also manages the distribution of the images to virtual machine disks.

- OpenStack Object Storage: It is used to create a redundant and scalable storage space for storing multiple petabytes of data. It is used for long term storage of large volumes.

(e) CloudStack (Kranas et al. (2012)): CloudStack is completely open source that offers Cloud including public, private and hybrid. It has Java based implementation. It is compatible with Xen, KVM and VMWare. CloudStack provides two interfaces: a Web interface and a RESTful API.

Architecture:

CloudStack has five types of components as shown in figure 3.14: Computer Nodes, Clusters, Pods, Availability Zones, and the Management Service. Explanation of these components are as follows (Muñoz et al. (2012)):

- Computer Nodes: The Computer Nodes (CNs) are the resources that have the CloudStack agent and any of the hypervisors supported by the platform like KVM, XenServer or VMware vSphere are installed. These nodes can be identified by the machine that allows the execution of VMs.

- Clusters: A group of CNs is a Cluster. CNs of the same
Cluster have the same hypervisor installed and share the same primary storage.

- **Pods**: A collection of Clusters is called as a Pod.
- **Availability Zones**: Collection of Pod is called as Availability Zones.
- **Management Service**: Management Service allows to manage the entire Cloud. It controls and manages the services of Cloud.

![CloudStack Diagram](image)

**Figure 3.14: CloudStack**

The CloudStack platform has three different users with different roles:

i. **Root Administrator**: The root administrator has the highest access privileges. It can manage the entire Cloud, including the hosts, clusters, Pod, user accounts, services offered and many more.

ii. **Domain Administrator**: The domain Administrator is responsible to perform administrative operations of only one
domain, but cannot access physical servers or other domains.

iii. CloudStack user: The CloudStack user has no privileges. It can only manage and handle its own virtual resources like VMs.

(f) Ganeti: Ganeti (Kumar et al. (2014)) is a virtual machine cluster management tool developed by Google. It is lightweight, easy to implement for a small cluster for organizations. It is compatible with Xen, KVM, VMWare. It doesn’t provide object/image storage by default and does not provide any direct interface to users.

Architecture: As shown in figure 3.15, architecture of ganeti has two main components: cluster and node group.

- Cluster: Collection of node is called cluster. Instances (guests) run on nodes. A cluster has one master node that can handle and control the all other nodes in cluster. It control and handle VM by VM migration in case of failure or overload in same cluster.

- Node group: Collection of cluster is known as Node group which actually is a split up of nodes into logical groups.

(g) Enomaly: Enomaly is an open source tool providing IaaS service. It focuses on small Cloud implementation (enomaly (2015)). It can be deployed with the Xen and KVM. Open source edition of Enomaly suffers from many restrictions like limited scalability and no mechanism for capacity control.

(h) Apache VCL (Gonçalves et al. (2011)): Apache VCL is an open source tool for providing SaaS. It focuses on applications of the internet. It can deploy with VMware. Apache VCL has a simple architecture consisting of three tiers: Web server, Database server and Management nodes

- Web server: This Components represents the VCL portal and uses Linux/Apache/PHP solution. This portal uses to enable the requesting and management of VCL resources through user interface.

- Database server: It stores the information like VCL reservations, different access controls, machine and environment inventory. It uses Linux/SQL solution for server.
Management nodes: It is the processing engine, which controls a subset of VCL resources like physical blade servers, traditional rack, or virtual machines. It uses Linux/VCLD (perl)/image library solution. VCLD is a middleware responsible to process reservations or jobs assigned by the VCL web portal. According to type of environment requested, VCLD provides a service (computational environment) to available user. Figure 3.16 shows a conceptual overview of the VCL, where to access its resources through a web interface, the user has to connect firstly to the VCL scheduling application. Users may request a reservation to use the environment immediately or schedule to use it in the future.

(i) TPlatform: TPlatform is used for providing services of PaaS services. It focuses on web mining. Infrastructure of TPlatform is supported by three technologies: a scalable file system called Tianwang File System (TFS) (which is similar to the Google File System (GFS)), the BigTable data storage mechanism, and the MapReduce programming model (Gonçalves et al. (2011)).

Architecture (Endo et al. (2010)): As shown in figure 3.17,
architecture of TPlatform has two main components: PC cluster and Infrastructure and Data processing applications.

- **PC Cluster:** This layer is responsible for controlling the hardware infrastructure for data processing.

- **Infrastructure:** This layer includes different components of infrastructure like of file system (TFS), programming model (MapReduce) and distributed data storage mechanism (BigTable).

- **Data Processing Applications:** This layer facilitates the services for users to develop their application like web data analysis and language processing and many more.

(j) **XCP:** It provides IaaS. It focuses on Automatic management in Cloud. It can be deployed in Xen. It can not provide overall architecture of Cloud as it can not provide any interface to end users for interaction (Gonçalves et al. (2011)).

**Discussion:** The Eucalyptus Cloud is mainly used for private Clouds, while, OpenNebula can be considered as a standard-based open-source toolkit to build private, public, and hybrid Clouds and can be used with the Amazon EC2 service. VMM configuration setup is easy in Eucalyptus, as the details of VM are set by administrator for all users while its tedious in OpenNebula where all users must be made aware of some underlying configuration in order to properly configure VMs, as it requires users to provide more details of the VM. It has been analyzed that Nimbus emphasizes
more towards science community, which may be less interested in the internal technique of the system. But, nimbus needs broad customization requirements. So, this science community needs to be more familiar with the Globus Toolkit. While XCP is intended to manage a virtualized infrastructure and does not offer a solution for resource’s negotiation. OpenStack is excellent for large deployment, it has medium complexity for setup. It uses python only for coding. CloudStack has lots of features. It also has medium complexity for set up as it has monolithic component architecture. Ganeti has built-in fault tolerance. It is ideal for small clusters. It is less complex, but is having less features. It has no EC2 compatibility. Summary of a discussion on Cloud architecture is shown in table 3.2.

Figure 3.17: Tplatform
Table 3.2: Comparison of Cloud architecture

<table>
<thead>
<tr>
<th>Cloud architecture</th>
<th>Cloud type</th>
<th>Hypervisor support</th>
<th>OS supported</th>
<th>Supported languages</th>
<th>Main Characteristics</th>
<th>Service</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eucalyptus</td>
<td>Private</td>
<td>VMWare, KVM, Xen, ESX</td>
<td>Linux and Windows VMs</td>
<td>Java and C</td>
<td>Hierarchical Architecture</td>
<td>IaaS</td>
<td>Large and Small Organization</td>
</tr>
<tr>
<td>OpenNebula</td>
<td>Public, Private and Hybrid</td>
<td>VMware, KVM, VirtualBox, Xen and libvirt</td>
<td>Linux</td>
<td>C++, C, Ruby, Java, Shell script, lex and yacc (compiler tools)</td>
<td>Policy-driven resource allocation</td>
<td>IaaS</td>
<td>Large and Small Organization.</td>
</tr>
<tr>
<td>CloudStack</td>
<td>Public, Private and Hybrid</td>
<td>Xen, KVM, VMware</td>
<td>Linux</td>
<td>Java</td>
<td>Deploy and manage large networks of VM</td>
<td>IaaS</td>
<td>large and small organizations.</td>
</tr>
<tr>
<td>Nimbus</td>
<td>Private and Public</td>
<td>Xen 3.x or KVM and bash, ebtables, libvirt</td>
<td>Linux</td>
<td>Java and Python</td>
<td>Legacy clusters for Cloud</td>
<td>IaaS</td>
<td>Small organization.</td>
</tr>
<tr>
<td>OpenStack</td>
<td>Public, Private and Hybrid</td>
<td>Xen, KVM, Hyper-VQEMU, UML (User Mode Linux), XenServer, LXC</td>
<td>Linux</td>
<td>Python</td>
<td>Supports ARM and many alternative hardware architectures</td>
<td>IaaS</td>
<td>Best option for Private and public Cloud</td>
</tr>
</tbody>
</table>
Table: Comparison of Cloud architecture continue

<table>
<thead>
<tr>
<th>Cloud architecture</th>
<th>Cloud type</th>
<th>Hypervisor support</th>
<th>OS supported</th>
<th>Supported languages</th>
<th>Main Characteristics</th>
<th>Service Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ganeti</td>
<td>Private</td>
<td>Xen, KVM, VMWare</td>
<td>Linux</td>
<td>Python, Haskell, Shell</td>
<td>Light weight, easy to implement</td>
<td>IaaS</td>
</tr>
<tr>
<td>Enomaly</td>
<td>Private</td>
<td>Xen, KVM and VirtualBox</td>
<td>Linux</td>
<td>Python 2.5</td>
<td>Targeting small Cloud</td>
<td>IaaS</td>
</tr>
<tr>
<td>Apache VCL</td>
<td>Private</td>
<td>VMware</td>
<td>Linux</td>
<td>PHP</td>
<td>Applications on the Internet</td>
<td>SaaS</td>
</tr>
<tr>
<td>TPlatform</td>
<td>Private</td>
<td>TFS and MapReduce</td>
<td>Linux</td>
<td>Python, C, Java, C++, Ruby</td>
<td>Development platform for web mining applications</td>
<td>PaaS</td>
</tr>
<tr>
<td>XCP</td>
<td>Private</td>
<td>Xen</td>
<td>Windows and Linux</td>
<td>—</td>
<td>Only tool for automatic management</td>
<td>IaaS</td>
</tr>
</tbody>
</table>
3.4 Conclusion and Future work

Cloud computing is identified as a new and most promising paradigm delivering IT services as computing utilities. As Clouds are designed to provide services on pay-per-use bases to end users where providers need to be recompense for sharing their resources and capabilities. In this chapter, we have discussed various open source Cloud architectures and hypervisors. In particular, we have discussed hypervisors like KVM, Xen, LXD, Lguest and VirtualBox with container technologies like OpenVz and LXC. In the section of open-source Cloud architectures/software platforms, we covered OpenNebula, Eucalyptus, Nimbus, OpenStack, CloudStack, TPlatform, ApacheVCL, Ganeti and XCP. Moreover, we have discussed some representative platforms for Cloud computing covering the state-of-the-art. In analyzing these various open-source hypervisors and Cloud computing architecture/Cloud software, we found that there are salient differences between them in the overall scheme of their design and function. They identified to be useful based on the Cloud requirements for deployment. As the chapter provides brief introduction of many open source IaaS platforms and architectures, it may be helpful to researchers/users to answer the question which open source IaaS platforms and architectures is right for them. Also, it gives a huge space to start a further work by selecting a best option for hypervisor and Cloud architectures/software available in open source. However, detail working of hypervisors/Cloud platforms and architectures are not explained/covered in present chapter that can be covered in future work. Moreover, these Cloud architectures and platforms are not compared by performing any experiments on them. Only theoretical analysis is done. So, comparison of all this open source IaaS frameworks based on experimental results can be targeted in future work.