INFRASTRUCTURE-AS-A-SERVICE CLOUD MONITOR

Project report submitted in fulfilment of the requirement for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Abhinav Kumar (121225)

Sai Dinesh Tumu (123202)

Under the supervision of

Mr. Punit Gupta

to

Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Candidate's Declaration

I hereby declare that the work presented in this report entitled **"Infrastructure-as-a-Service Cloud Monitor"** in fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2015 to May 2016 under the supervision of Mr. Punit Gupta.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Abhinav Kumar, 121225

Sai Dinesh Tumu, 123202

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Mr Punit Gupta
Assistant Professor
Computer Science and Engineering
Dated:

ACKNOWLEDGEMENT

We would like to show our profound gratitude towards our project guide Mr. Punit Gupta, Assistant Professor, Department of Computer Science and Engineering for his guidance and constant supervision. He guided us in carrying out the project work with his able guidance, support and encouragement throughout in the entirety of semester. We consider his profound instructions, his concern and assistance with all things, invaluable.

We would also like to extend our sincerest thanks to the entire faculty members and especially the lab staff of the Department of Computer Science and Engineering, JUIT

The completion of the project would not have been possible without open source projects and the community. We would like to extend our sincere thanks to all of those involved.

Abhinav Kumar, 121225

Sai Dinesh Tumu, 123202

Table of Contents

List of Abbreviations

1.	VPS	Virtual Private Server
2.	QoS	Quality of Service
3.	SLA	Software License Agreement
4.	VM	Virtual Machine
5.	VEE	Virtual Execution Environment
6.	GPL	GNU Public License
7.	LAMP	Linux, Apache, MySQL, PHP

List of Figures

Abstract

With the advent of cloud computing applications, monitoring becomes a valid concern. Monitoring for various aspects in a cloud application is difficult because of multiple failure points spanning both hardware and software. Moreover the cluster nature of a cloud application increases the scope of failure and cost and it becomes even harder to measure and manage the same. In this project, Infrastructure-as-a-Service Cloud Monitor, we monitor the virtual machines and the hosts running the virtual machines for attributes including power usage. This Cloud Manager is a desktop application and generates daily and weekly reports.

Chapter-1

INTRODUCTION

1.1 Introduction

Although Cloud Computing is a popular trend, it is difficult to get a clear definition of it. Cloud Computing, often referred to in the mainstream media as simply "the cloud" is the delivery of on-demand computing resources to the clients on a pay-for-use basis. These resources can be anything – from applications to data-centers. That means, cloud computing relies on sharing computing resources rather than having local servers or personal devices to handle applications.

In cloud computing, the word "cloud" is used as a metaphor for "the Internet", so the term cloud-computing may be mean a type of Internet-based computing where all the different services such as platforms, servers, storage and applications etc are delivered to the client's computers and devices via the Internet.

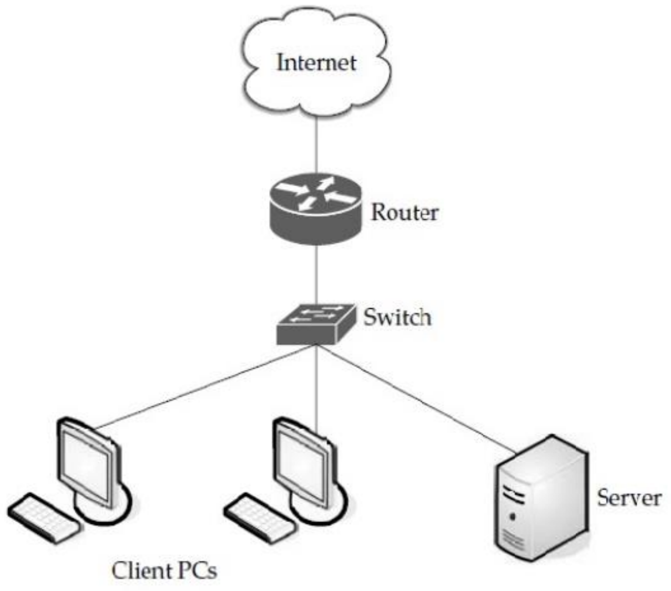


Fig (1-1): Cloud is used to refer to the Internet in network diagrams

1.2 Problem Statement

To design and develop an Infrastructure-as-a-Service (IaaS) Cloud monitor which is a dashboard for monitoring and managing running nodes, accessed remotely and on desktop, with a focus on power management and gauging and analysing power consumption.

1.3 Objectives

1. To build a cloud infrastructure on a local machine
2. To design and develop a desktop application which can be used to monitor and manage the cloud infrastructure locally or remotely
3. To generate automated performance reports, daily and weekly to analyse the resources used by a host machine or a virtual machine
4. A mechanism to monitor the power consumption patterns of virtual machines and visualize on the desktop application using graphs

1.4 Organization

The rest of the project report is organized as follows:

Chapter 1 gives a brief introduction of the concepts used in the project, its problem statement, objectives, methodology, report organization and the related work done in the particular domain.

Chapter 2 gives an overall and detailed idea of cloud computing, its models, functionalities and services provided by the cloud. It also brings light on the technologies involved in the attainment of the project goal.

Chapter 3 describes the system mode, how the local infrastructure was developed using all the software and hardware technologies involved.

Chapter 4 describes all milestones of the project and some snapshots of the Cloud Manager and the generated reports.

Chapter 5 gives the conclusion and summary of the whole project report.

Chapter-2

LITERATURE SURVEY

2.1 Cloud Computing

Cloud computing is a computing paradigm, where a large pool of systems is connected in a private or public network, to provide dynamically scalable infrastructure for application, data and file storage. With the advent of this technology, the cost of computation, application hosting, content storage and delivery is reduced significantly.

Cloud computing is a practical approach to experience direct cost benefits and it has the potential to transform a data-center from a capital-intensive set up to a variable priced environment.

The idea of cloud computing is based on a very fundamental principal of reusability of IT capabilities. The difference that cloud computing brings compared to traditional concepts of “grid computing”, “distributed computing”, “utility computing”, or “autonomic computing” is to broaden horizons across organizational boundaries.

There is no standard definition of Cloud Computing Forrester defines cloud computing as:

“A pool of abstracted, highly scalable, and managed compute infrastructure capable of hosting end customer applications and billed by consumption.”

The origin of the term *cloud computing* is unclear. The word "cloud" is commonly used in science to describe a large agglomeration of objects that visually appear from a distance as a cloud and describes any set of things whose details are not inspected further in a given context. Another explanation is that the old programs that drew network schematics surrounded the icons for servers with a circle, and a cluster of servers in a network diagram had several overlapping circles, which resembled a cloud.

In analogy to above usage the word *cloud* was used as a metaphor for the Internet and a standardized cloud-like shape was used to denote a network on telephony schematics and later to depict the Internet in computer network diagrams. With this simplification, the implication is that the specifics of how the end points of a network are connected are not relevant for the purposes of understanding the diagram. The cloud symbol was used to represent networks of computing equipment in the original ARPANET by as early as 1977, and the CSNET by 1981 both predecessors to the Internet itself.

The term *cloud* has been used to refer to platforms for distributed computing. In *Wired's* April 1994 feature "Bill and Andy's Excellent Adventure II" on the Apple spin-off General Magic, Andy Hertzfeld comments on General Magic's distributed programming language Telescript that:

"The beauty of Telescript ... is that now, instead of just having a device to program, we now have the entire Cloud out there, where a single program can go and travel to many different sources of information and create sort of a virtual service. No one had conceived that before. The example Jim White [the designer of Telescript, X.400 and ASN.1] uses now is a date-arranging service where a software agent goes to the flower store and orders flowers and then goes to the ticket shop and gets the tickets for the show, and everything is communicated to both parties."

References to "cloud computing" in its modern sense appeared as early as 1996, with the earliest known mention in a Compaq internal document.

The popularization of the term can be traced to 2006 when Amazon.com introduced the Elastic Compute Cloud.

Components of Cloud Computing

A cloud system usually comprises of 3 major components, each of which plays a specific role.

- Clients
- Datacenters
- Distributed Servers

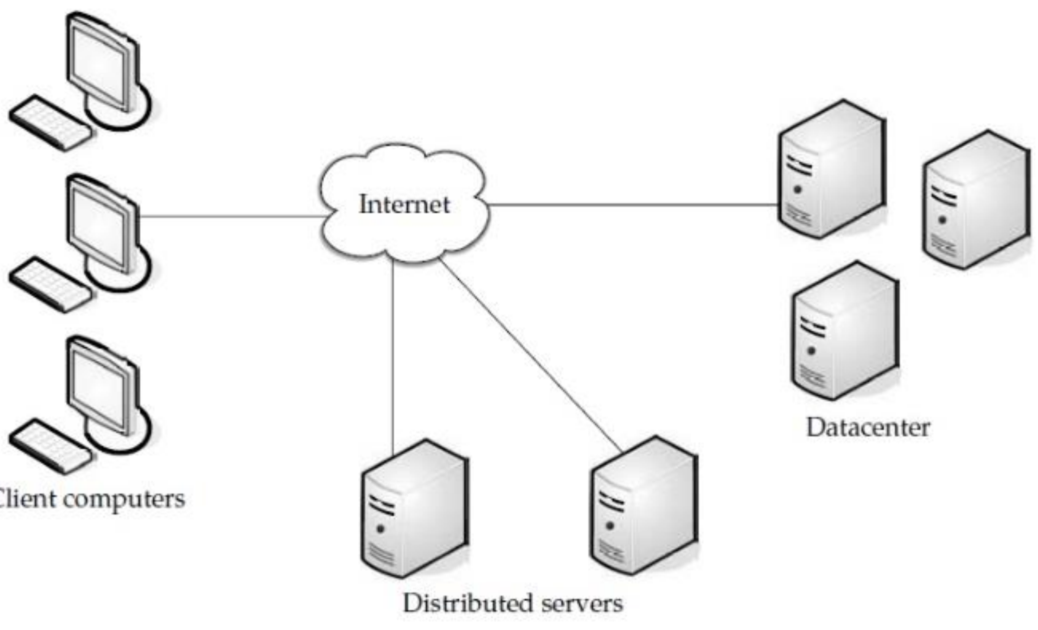


Fig (2-1) Cloud components

Clients

Cloud computing architecture consists of front-end platforms called cloud clients, or simple clients. The clients are what end users interact with to manage information related to the cloud. These clients comprise servers, fat or thick clients, thin clients, zero clients, tablets, smartphonesa and other personal mobile devices.

These cloud clients may interact with the cloud via a middleware application, via a web browser or through a remote session technology such as SecureShell (SSH), Telnet, RDP, VNC etc.

Thick client

A thick client (also known as fat client) is a computer that is capable of providing rich functionality independent of the cloud system. A fat client still requires at least periodic connection to a cloud network, but is often characterised by the ability to perform many functions without that connection.

Thin client

In contrast, a thin client generally does as little processing as possible and relies on accessing the cloud servers each time input data needs to be processed or validated.

Zero client

A zero client (also known as an ultra-thin client) initializes the network to gather required configuration files then then point it to where its operating system boot-strapper and binary files are located. A zero client runs entirely runs via the network. So if the network goes down, the ultra-thin client device cannot serve any purpose to the user.

These days, thin clients are favoured compared to other forms of clients due to their low cost, low power consumption requirement, easy reparability and replicability etc. Due to less surface of attack, thin clients also provide a greater security than other forms of cloud clients.

Characteristics of Cloud Computing:

Agility improves with users' ability to re-provision technological infrastructure resources.

Cost reductions claimed by cloud providers. A public-cloud delivery model converts capital expenditure to operational expenditure. This purportedly lowers barriers to entry, as infrastructure is typically provided by a third party and does not need to be purchased for one-time or infrequent intensive computing tasks.

Pricing on a utility computing basis is fine-grained, with usage-based options and fewer IT skills are required for implementation (in-house). The e-FISCAL project's state-of-the-art repository contains several articles looking into cost aspects in more detail, most of them concluding that costs savings depend on the type of activities supported and the type of infrastructure available in-house.

Device and location independence enable users to access systems using a web browser regardless of their location or what device they use (e.g., PC, mobile phone). As infrastructure is off-site (typically provided by a third-party) and accessed via the Internet, users can connect from anywhere.

Maintenance of cloud computing applications is easier, because they do not need to be installed on each user's computer and can be accessed from different places.

Multitenancy enables sharing of resources and costs across a large pool of users thus allowing for:

- *centralization* of infrastructure in locations with lower costs (such as real estate, electricity, etc.)
- *peak-load capacity* increases (users need not engineer for highest possible load-levels)
- *utilisation and efficiency* improvements for systems that are often only 10–20% utilised.

Performance is monitored, and consistent and loosely coupled architectures are constructed using web services as the system interface.

Productivity may be increased when multiple users can work on the same data simultaneously, rather than waiting for it to be saved and emailed. Time may be saved as information does not need to be re-entered when fields are matched, nor do users need to install application software upgrades to their computer.

Reliability improves with the use of multiple redundant sites, which makes well-designed cloud computing suitable for business continuity and disaster recovery.

Scalability and elasticity via dynamic ("on-demand") provisioning of resources on a fine-grained, self-service basis in near real-time (Note, the Virtual Machine start-up time varies by type of the virtual machine, location, OS and cloud providers), without users having to engineer for peak loads. This gives the ability to scale up when the usage need increases or down if resources are not being used.

Security can improve due to centralization of data, increased security-focused resources, etc., but concerns can persist about loss of control over certain sensitive data, and the lack of security for stored kernels. Security is often as good as or better than other traditional systems, in part because providers are able to devote resources to solving security issues that many customers cannot afford to tackle.

However, the complexity of security is greatly increased when data is distributed over a wider area or over a greater number of devices, as well as in multi-tenant systems shared by unrelated users. In addition, user access to security audit logs may be difficult or impossible.

Models of Cloud Computing:

Cloud Providers offer services that can be mainly grouped into three categories:

- Software-as-a-Service (SaaS)
- Platform-as-a-Service (PaaS)
- Infrastructure-as-a-Service (IaaS) or Hardware-as-a-Service (HaaS)

Others being:

- Storage-as-a-Service
- Database-as-a-Service
- Information-as-a-Service
- Process-as-a-Service
- Integration-as-a-Service
- Security-as-a-Service
- Management-as-a-Service
- Testing-as-a-Service
- Governance-as-a-Service
- Development-as-a-Service

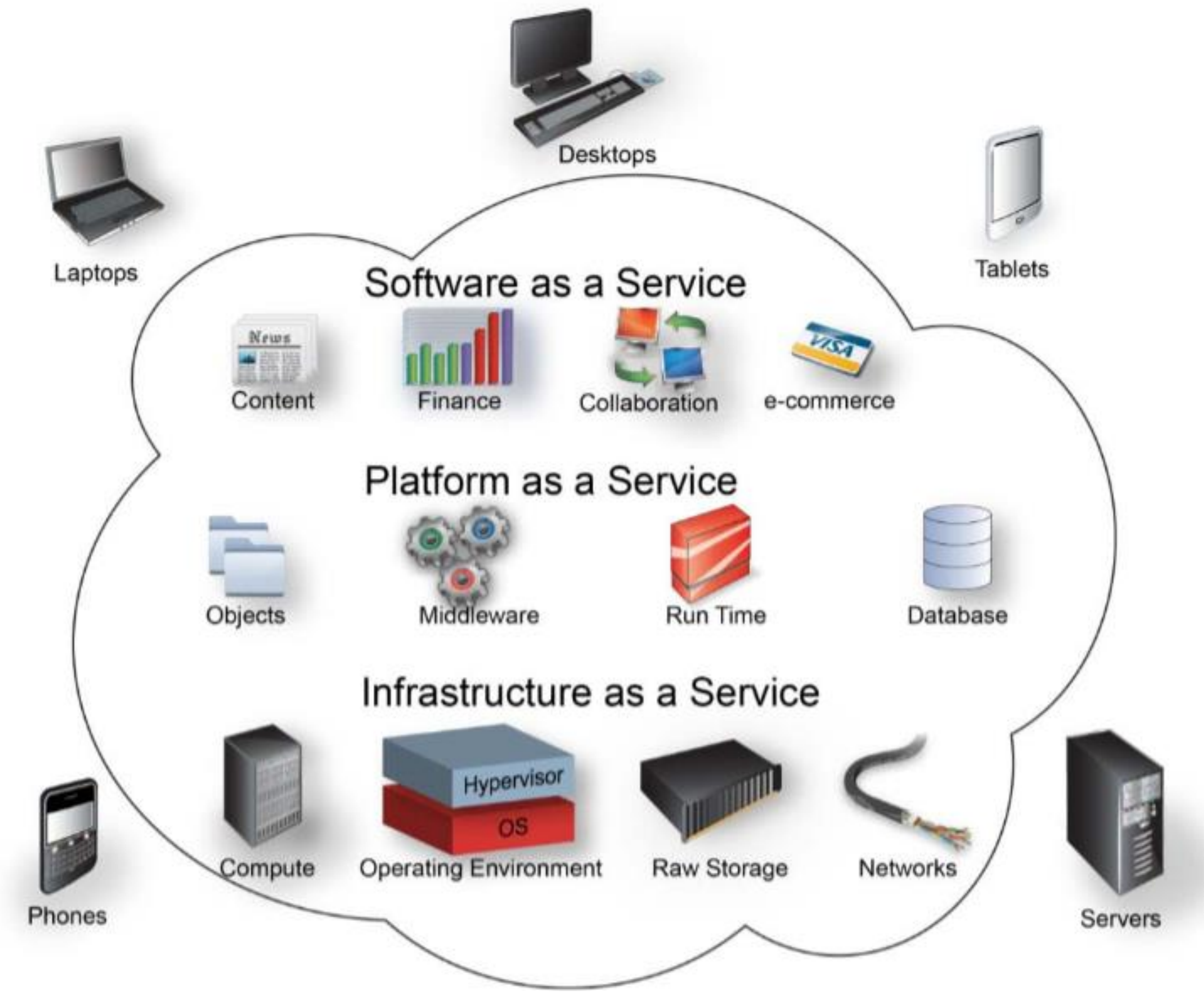


Fig (2-2) The three main models of cloud computing

Software-as-a-Service (SaaS): In this model, a complete application is offered to the customer, as a service on demand. A single instance of the service runs on the cloud & multiple end users are serviced. On the customers‟ side, there is no need for upfront investment in servers or software licenses, while for the provider, the costs are lowered, since only a single application needs to be hosted & maintained. Today SaaS is offered by companies such as Google, Salesforce, Microsoft, Zoho, etc.

Platform-as-a-Service (PaaS): Here, a layer of software, or development environment is encapsulated & offered as a service, upon which other higher levels of service can be built. The customer has the freedom to build his own applications, which run on the provider‟s infrastructure. To meet manageability and scalability requirements of the applications, PaaS providers offer a predefined combination of OS and application servers, such as LAMP platform (Linux, Apache, MySql and PHP), restricted J2EE, Ruby etc. Google's App Engine, Force.com, etc are some of the popular PaaS examples.

Infrastructure-as-a-Service (IaaS) or Hardware-as-a-Service (HaaS): IaaS provides basic storage and computing capabilities as standardized services over the network. Servers, storage systems, networking equipment, data centre space etc. are pooled and made available to handle workloads. The customer would typically deploy his own software on the infrastructure. Some common examples are Amazon, GoGrid, 3 Tera, etc.

<u>Understanding public and private clouds:</u>

Enterprises can choose to deploy applications on Public, Private or Hybrid clouds. Cloud Integrators can play a vital part in determining the right cloud path for each organization.

Public clouds

The most recognisable model of cloud computing to many consumers is the public cloud model, under which cloud services are provided in a virtualised environment, constructed using pooled shared physical resources, and accessible over a public network such as the internet. To some extent they can be defined in contrast to private clouds which ring-fence the pool of underlying computing resources, creating a distinct cloud platform to which only a single organisation has access. Public clouds, however, provide services to multiple clients using the same shared infrastructure.

They are owned and operated by third parties; they deliver superior economies of scale to customers, as the infrastructure costs are spread among a mix of users, giving each individual client an attractive low-cost, “Pay-as-you-go” model. All customers share the same infrastructure pool with limited configuration, security protections, and availability variances. These are managed and supported by the cloud provider. One of the advantages of a Public cloud is that they may be larger than an enterprises cloud, thus providing the ability to scale seamlessly, on demand.

The most salient examples of cloud computing tend to fall into the public cloud model because they are, by definition, publicly available. Software as a Service (SaaS) offerings such as cloud storage and online office applications are perhaps the most familiar, but widely available Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) offerings, including cloud based web hosting and development environments, can follow the model as well (although all can also exist within private clouds). Public clouds are used extensively in offerings for private individuals who are less likely to need the level of infrastructure and security offered by private clouds. However, enterprise can still utilise public clouds to make their operations significantly more efficient, for example, with the storage of non-sensitive content, online document collaboration and webmail.

The public model offers the following features and benefits:

- Ultimate scalability; cloud resources are available on demand from the public clouds' vast pools of resource so that the applications that run on them can respond seamlessly to fluctuations in activity
- Cost effective; public clouds bring together greater levels of resource and so can benefit from the largest economies of scale. The centralised operation and management of the underlying resources is shared across all of the subsequent cloud services whilst components, such as servers, require less bespoke configuration. Some mass market propositions can even be free to the client, relying on advertising for their revenue.
- Utility style costing; public cloud services often employ a pay-as-you-go charging model whereby the consumer will be able to access the resource they need, when they need it, and then only pay for what they use; therefore, avoiding wasted capacity
- Reliability; the sheer number of servers and networks involved in creating a public cloud and the redundancy configurations mean that should one physical component fail, the cloud service would still run unaffected on the remaining components. In some cases, where clouds draw resource from multiple data centres, an entire data centre could go offline and individual cloud services would suffer no ill effect. There is, in other words, no single point of failure which would make a public cloud service vulnerable
- Flexibility; there are a myriad of IaaS, PaaS and SaaS services available on the market which follow the public cloud model and that are ready to be accessed as a service from any internet enabled device. These services can fulfil most computing requirements and can deliver their benefits to private and enterprise clients alike. Businesses can even integrate their public cloud services with private clouds, where they need to perform sensitive business functions, to create hybrid clouds
- Location independence; the availability of public cloud services through an internet connection ensures that the services are available wherever the client is located. This provides invaluable opportunities to enterprise such as remote access to IT infrastructure (in case of emergencies etc) or online document collaboration from multiple locations.

Private clouds

A private cloud is a particular model of cloud computing that involves a distinct and secure cloud based environment in which only the specified client can operate. As with other cloud models, private clouds will provide computing power as a service within a virtualised environment using an underlying pool of physical computing resource. However, under the private cloud model, the cloud (the pool of resource) is only accessible by a single organisation providing that organisation with greater control and privacy.

The technical mechanisms used to provide the different services which can be classed as being private cloud services can vary considerably and so it is hard to define what constitutes a private cloud from a technical aspect. Instead such services are usually categorised by the features that they offer to their client. Traits that characterise private clouds include the ring fencing of a cloud for the sole use of one organisation and higher levels of network security. They can be defined in contrast to a public cloud which has multiple clients accessing virtualised services which all draw their resource from the same pool of servers across public networks. Private cloud services draw their resource from a distinct pool of physical computers but these may be hosted internally or externally and may be accessed across private leased lines or secure encrypted connections via public networks.

The additional security offered by the ring fenced cloud model is ideal for any organisation, including enterprise that needs to store and process private data or carry out sensitive tasks. For example, a private cloud service could be utilised by a financial company that is required by regulation to store sensitive data internally and who will still want to benefit from some of the advantages of cloud computing within their business infrastructure, such as on demand resource allocation.

The private cloud model is closer to the more traditional model of individual local access networks (LANs) used in the past by enterprise but with the added advantages of virtualisation. The features and benefits of private clouds therefore are:

- Higher security and privacy; public clouds services can implement a certain level of security but private clouds - using techniques such as distinct pools of resources with access restricted to connections made from behind one organisation’s firewall, dedicated leased lines and/or on-site internal hosting - can ensure that operations are kept out of the reach of prying eyes
- More control; as a private cloud is only accessible by a single organisation, that organisation will have the ability to configure and manage it in line with their needs to achieve a tailored network solution. However, this level of control removes some of the economies of scale generated in public clouds by having centralised management of the hardware

- Cost and energy efficiency; implementing a private cloud model can improve the allocation of resources within an organisation by ensuring that the availability of resources to individual departments/business functions can directly and flexibly respond to their demand. Therefore, although they are not as cost effective as a public cloud services due to smaller economies of scale and increased management costs, they do make more efficient use of the computing resource than traditional LANs as they minimise the investment into unused capacity. Not only does this provide a cost saving but it can reduce an organisation's carbon footprint too

- Improved reliability; even where resources (servers, networks etc.) are hosted internally, the creation of virtualised operating environments means that the network is more resilient to individual failures across the physical infrastructure. Virtual partitions can, for example, pull their resource from the remaining unaffected servers. In addition, where the cloud is hosted with a third party, the organisation can still benefit from the physical security afforded to infrastructure hosted within data centres

- Cloud bursting; some providers may offer the opportunity to employ cloud bursting, within a private cloud offering, in the event of spikes in demand. This service allows the provider to switch certain non-sensitive functions to a public cloud to free up more space in the private cloud for the sensitive functions that require it. Private clouds can even be integrated with public cloud services to form hybrid clouds where non-sensitive functions are always allocated to the public cloud to maximise the efficiencies on offer.

Additionally, a private cloud can be deployed in two ways:

- *On-premise Private Cloud:* On-premise private clouds, also known as internal clouds are hosted within one's own data-centre. This model provides a more standardized process and protection, but is limited in aspects of size and scalability. IT departments would also need to incur the capital and operational costs for the physical resources. This is best suited for applications which require complete control and configurability of the infrastructure and security.

- *Externally-hosted Private Cloud:* This type of private cloud is hosted externally with a cloud provider, where the provider facilitates an exclusive cloud environment with full guarantee of privacy. This is best suited for enterprises that don't prefer a public cloud due to sharing of physical resources.

Hybrid Clouds

They combine both public and private cloud models. With a Hybrid Cloud, service providers can utilize 3rd party Cloud Providers in a full or partial manner thus increasing the flexibility of computing. The Hybrid cloud environment is capable of providing on-demand, externally provisioned scale. The ability to augment a private cloud with the resources of a public cloud can be used to manage any unexpected surges in workload.

Virtualization

In computing, virtualization means to create a virtual version of a device or resource, such as a server, storage device, network or even an operating system where the framework divides the resource into one or more execution environments. Even something as simple as partitioning a hard drive is considered virtualization because you take one drive and partition it to create two separate hard drives. Devices, applications and human users are able to interact with the virtual resource as if it were a real single logical resource. The term virtualization has become somewhat of a buzzword, and as a result the term is now associated with a number of computing technologies including the following:

- Storage Virtualization
- Server Virtualization
- Operating System-Level Virtualization
- Network Virtualization
- Application Virtualization

Storage Virtualization

Storage Virtualization uses virtualization to enable better functionality and more advanced features in computer data storage systems.

Broadly speaking, a "storage system" is also known as a storage array or disk array or a filer. Storage systems typically use special hardware and software along with disk drives in order to provide very fast and reliable storage for computing and data processing. Storage systems are complex, and may be thought of as a special purpose computer designed to provide storage capacity along with advanced data protection features. Disk drives are only one element within a storage system, along with hardware and special purpose embedded software within the system.

Storage systems can provide either block accessed storage, or file accessed storage. Block access is typically delivered over Fibre Channel, iSCSI, SAS, FICON or other protocols. File access is often provided using NFS or CIFS protocols.

Within the context of a storage system, there are two primary types of virtualization that can occur:

- Block virtualization used in this context refers to the abstraction (separation) of logical storage (partition) from physical storage so that it may be accessed without regard to physical storage or heterogeneous structure. This separation allows the administrators of the storage system greater flexibility in how they manage storage for end users.[1]
- File virtualization addresses the NAS challenges by eliminating the dependencies between the data accessed at the file level and the location where the files are physically stored. This provides opportunities to optimize storage use and server consolidation and to perform non-disruptive file migrations.

Server Virtualization

Server Virtualization is similar to that which led to the development of time-sharing and multiprogramming in the past. Although the resources are still shared, as under the time-sharing model, virtualization provides a higher level of security, dependent on the type of virtualization used, as the individual virtual servers are mostly isolated from each other and may run their own full-fledged operating system which can be independently rebooted as a virtual instance.

Partitioning a single server to appear as multiple servers has been increasingly common on microcomputers since the launch of VMware ESX Server in 2001. The physical server typically runs a hypervisor which is tasked with creating, releasing, and managing the resources of "guest" operating systems, or virtual machines. These guest operating systems are allocated a share of resources of the physical server, typically in a manner in which the guest is not aware of any other physical resources save for those allocated to it by the hypervisor. As a VPS runs its own copy of its operating system, customers have superuser-level access to that operating system instance, and can install almost any software that runs on the OS, however due to the number of virtualization clients typically running on a single machine, a VPS generally has limited processor time, RAM, and disk space.

Although VMware and Hyper-V dominate in-house corporate virtualization, because of their cost and limitations they are less common for VPS providers, which instead typically use products such as OpenVZ, Virtuozzo, Xen or KVM.

Operating-System Level Virtualization

Operating-system-level virtualization is a virtualization method where the kernel of an operating system allows for multiple isolated user-space instances, instead of just one. Such instances (sometimes called containers, software containers, virtualization engines (VE), virtual private servers (VPS), or jails) may look and feel like a real server from the point of view of its owners and users.

On Unix-like operating systems, one can see this technology as an advanced implementation of the standard chroot mechanism. In addition to isolation mechanisms, the kernel often provides resource-management features to limit the impact of one container's activities on other containers.

Network Virtualization

Network Virtualization is the process of combining hardware and software network resources and network functionality into a single, software-based administrative entity, a virtual network. Network virtualization involves platform virtualization, often combined with resource virtualization.

Network virtualization is categorized as either external virtualization, combining many networks or parts of networks into a virtual unit, or internal virtualization, providing network-like functionality to software containers on a single network server.

In software testing, software developers use network virtualization to test software under development in a simulation of the network environments in which the software is intended to operate. As a component of application performance engineering, network virtualization enables developers to emulate connections between applications, services, dependencies, and end users in a test environment without having to physically test the software on all possible hardware or system software. Of course, the validity of the test depends on the accuracy of the network virtualization in emulating real hardware and operating systems.

Application Virtualization

Application virtualization is software technology that encapsulates computer programs from the underlying operating system on which it is executed. A fully virtualized application is not installed in the traditional sense, although it is still executed as if it were. The application behaves at runtime like it is directly interfacing with the original operating system and all the resources managed by it, but can be isolated or sandboxed to varying degrees.

In this context, the term "virtualization" refers to the artifact being encapsulated (application), which is quite different from its meaning in hardware virtualization, where it refers to the artifact being abstracted (physical hardware).

Full application virtualization requires a virtualization layer. Application virtualization layers replace part of the runtime environment normally provided by the operating system. The layer intercepts all disk operations of virtualized applications and transparently redirects them to a virtualized location, often a single file. The application remains unaware that it accesses a virtual resource instead of a physical one. Since the application is now working with one file instead of many files spread throughout the system, it becomes easy to run the application on a different computer and previously incompatible applications can be run side-by-side. Examples of this technology for the Windows platform include Citrix XenApp, Microsoft App-V, Oracle Secure Global Desktop, Sandboxie, Symantec Workspace Virtualization, VMware ThinApp etc.

In view of the cloud computing systems, two types of virtualization are available:

- Full Virtualization
- Paravirtualization

Full Virtualization

Full Virtualization is a virtualization technique used to provide a certain kind of virtual machine environment, namely, one that is a complete simulation of the underlying hardware. Full virtualization requires that every salient feature of the hardware be reflected into one of several virtual machines – including the full instruction set, input/output operations, interrupts, memory access, and whatever other elements are used by the software that runs on the bare machine, and that is intended to run in a virtual machine. In such an environment, any software capable of execution on the raw hardware can be run in the virtual machine and, in particular, any operating systems. The obvious test of full virtualization is whether an operating system intended for stand-alone use can successfully run inside a virtual machine.

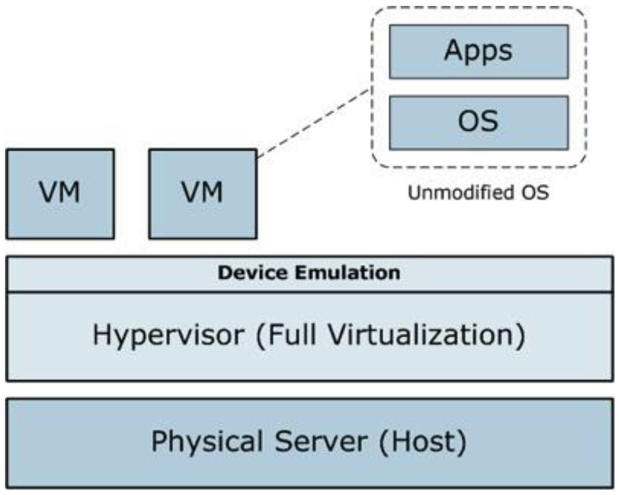


Fig (2-3) Full Virtualization Schematic

Other forms of platform virtualization allow only certain or modified software to run within a virtual machine. The concept of full virtualization is well established in the literature, but it is not always referred to by this specific term; see platform virtualization for terminology.

An important example of full virtualization was that provided by the control program of IBM's CP/CMS operating system. It was first demonstrated with IBM's CP-40 research system in 1967, then distributed via open source in CP/CMS in 1967-1972, and re-implemented in IBM's VM family from 1972 to the present. Each CP/CMS user was provided a simulated, stand-alone computer. Each such virtual machine had the complete capabilities of the underlying machine, and (for its user) the virtual machine was indistinguishable from a private system. This simulation was comprehensive, and was based on the Principles of Operation manual for the hardware. It thus included such elements as instruction set, main memory, interrupts, exceptions, and device access. The result was a single machine that could be multiplexed among many users.

Full virtualization is possible only with the right combination of hardware and software elements. For example, it was not possible with most of IBM's System/360 series with the exception being the IBM System/360-67; nor was it possible with IBM's early System/370 system until IBM added virtual memory hardware to the System/370 series in 1972.

Similarly, full virtualization was not quite possible with the x86 platform until the 2005-2006 addition of the AMD-V and Intel VT-x extensions (see x86 virtualization).[citation needed] Many platform hypervisors for the x86 platform came very close and claimed full virtualization even prior to the AMD-V and Intel VT-x additions. Examples include Adeos, Mac-on-Linux, Parallels Desktop for Mac, Parallels Workstation, VMware Workstation, VMware Server (formerly GSX Server), VirtualBox, Win4BSD, and Win4Lin Pro. VMware, for instance, employs a technique called binary translation to automatically modify x86 software on-the-fly to replace instructions that "pierce the virtual machine" with a different, virtual machine safe sequence of instructions; this technique provides the appearance of full virtualization.

A key challenge for full virtualization is the interception and simulation of privileged operations, such as I/O instructions. The effects of every operation performed within a given virtual machine must be kept within that virtual machine – virtual operations cannot be allowed to alter the state of any other virtual machine, the control program, or the hardware. Some machine instructions can be executed directly by the hardware, since their effects are entirely contained within the elements managed by the control program, such as memory locations and arithmetic registers. But other instructions that would "pierce the virtual machine" cannot be allowed to execute directly; they must instead be trapped and simulated. Such instructions either access or affect state information that is outside the virtual machine.

Full virtualization has proven highly successful for:

- Sharing a computer system among multiple users;
- Isolating users from each other (and from the control program);
- Emulating new hardware to achieve improved reliability, security and productivity.

Paravirtualization

Paravirtualization is a virtualization technique that presents a software interface to virtual machines that is similar, but not identical to that of the underlying hardware.

The intent of the modified interface is to reduce the portion of the guest's execution time spent performing operations which are substantially more difficult to run in a virtual environment compared to a non-virtualized environment. The paravirtualization provides specially defined 'hooks' to allow the guest(s) and host to request and acknowledge these tasks, which would otherwise be executed in the virtual domain (where execution performance is worse). A successful paravirtualized platform may allow the virtual machine monitor (VMM) to be simpler (by relocating execution of critical tasks from the virtual domain to the host domain), and/or reduce the overall performance degradation of machine-execution inside the virtual-guest.

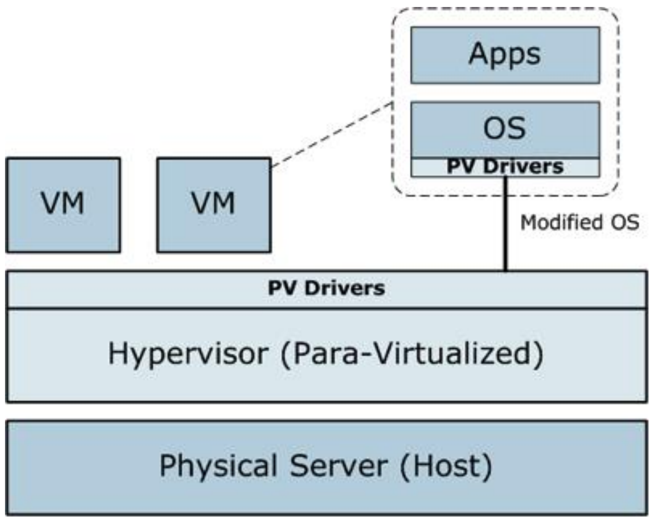


Fig (2-4) Paravirtualization

Paravirtualization requires the guest operating system to be explicitly ported for the para-API — a conventional OS distribution that is not paravirtualization-aware cannot be run on top of a paravirtualizing VMM. However, even in cases where the operating system cannot be modified, components may be available that enable many of the significant performance advantages of paravirtualization.

For example, the Xen Windows GPLPV project provides a kit of paravirtualization-aware device drivers, licensed under the terms of the GPL, that are intended to be installed into a Microsoft Windows virtual-guest running on the Xen hypervisor.

Other than these two types of virtualization, following types of virtualization technologies can also be used:

Hardware-assisted Virtualization

In hardware-assisted virtualization, the hardware provides architectural support that facilitates building a virtual machine monitor and allows guest OSes to be run in isolation. Hardware-assisted virtualization was first introduced on the IBM System/370 in 1972, for use with VM/370, the first virtual machine operating system.

In 2005 and 2006, Intel and AMD provided additional hardware to support virtualization on their shipped microprocessor products. Sun Microsystems (now Oracle Corporation) added similar features in their UltraSPARC T-Series processors in 2005. Examples of virtualization platforms adapted to such hardware include KVM, VMware Workstation, VMware Fusion, Microsoft Hyper-V, Windows Virtual PC, Xen, Parallels Desktop, Oracle VM Server, Oracle VM VirtualBox and Parallels Workstation.

Partial Virtualization

In partial virtualization, including address space virtualization, the virtual machine simulates multiple instances of much of an underlying hardware environment, particularly address spaces. Usually, this means that entire operating systems cannot run in the virtual machine—which would be the sign of full virtualization—but that many applications can run. A key form of partial virtualization is address space virtualization, in which each virtual machine consists of an independent address space. This capability requires address relocation hardware, and has been present in most practical examples of partial virtualization.

Cloud Computing Benefits:

Enterprises would need to align their applications, so as to exploit the architecture models that Cloud Computing offers. Some of the typical benefits are listed in the next page:

Reduced Cost

There are a number of reasons to attribute Cloud technology with lower costs. The billing model is pay as per usage; the infrastructure is not purchased thus lowering maintenance. Initial expense and recurring expenses are much lower than traditional computing.

Increased Storage

With the massive Infrastructure that is offered by Cloud providers today, storage & maintenance of large volumes of data is a reality. Sudden workload spikes are also managed effectively & efficiently, since the cloud can scale dynamically.

Flexibility

This is an extremely important characteristic. With enterprises having to adapt, even more rapidly, to changing business conditions, speed to deliver is critical. Cloud computing stresses on getting applications to market very quickly, by using the most appropriate building blocks necessary for deployment.

Some other benefits are:

- One can access applications as utilities, over the Internet.
- Manipulate and configure the application online at any time.
- It does not require to install a specific piece of software to access or manipulate cloud application.
- Cloud Computing offers online development and deployment tools, programming runtime environment through Platform as a Service model
- Cloud resources are available over the network in a manner that provides platform independent access to any type of clients.
- Cloud Computing offers on - demand self – service. The resources can be used without interaction with cloud service provider.
- Cloud Computing is highly cost effective because it operates at higher efficiencies with greater utilization.
- It just requires an Internet connection.
- Cloud Computing offers load balancing that makes it more reliable.

Cloud Computing Challenges:

Despite its growing influence, concerns regarding cloud computing still remain. In our opinion, the benefits outweigh the drawbacks and the model is worth exploring. Some common challenges are:

Data Protection

Data Security is a crucial element that warrants scrutiny. Enterprises are reluctant to buy an assurance of business data security from vendors. They fear losing data to competition and the data confidentiality of consumers. In many instances, the actual storage location is not disclosed, adding onto the security concerns of enterprises. In the existing models, firewalls across data centers (owned by enterprises) protect this sensitive information. In the cloud model, Service providers are responsible for maintaining data security and enterprises would have to rely on them.

Data Recovery and Availability

All business applications have Service level agreements that are stringently followed. Operational teams play a key role in management of service level agreements and runtime governance of applications. In production environments, operational teams support appropriate clustering and Fail over, Data Replication, System monitoring (Transactions monitoring, logs monitoring and others), Maintenance (Runtime Governance), Disaster recovery, and Capacity and performance management

If, any of the above mentioned services is under-served by a cloud provider, the damage & impact could be severe.

Management Capabilities

Despite there being multiple cloud providers, the management of platform and infrastructure is still in its infancy. Features like „Auto-scaling" for example, are a crucial requirement for many enterprises. There is huge potential to improve on the scalability and load balancing features provided today.

Regulatory and Compliance Restrictions

In some of the European countries, Government regulations do not allow customer's personal information and other sensitive information to be physically located outside the state or country. In order to meet such requirements, cloud providers need to setup a data-center or a storage site exclusively within the country to comply with regulations. Having such an infrastructure may not always be feasible and is a big challenge for cloud providers.

With cloud computing, the action moves to the interface that is, to the interface between service suppliers and multiple groups of service consumers. Cloud services will demand expertise in distributed services, procurement, risk assessment and service negotiation areas that many enterprises are only modestly equipped to handle.

2.2 Cloud Monitoring

Monitoring of Cloud is a task of paramount importance for both Providers and Consumers. On the one side, it is a key tool for controlling and managing hardware and software infrastructures; on the other side, it provides information and Key Performance Indicators (KPIs) for both platforms and applications. The continuous monitoring of the Cloud and of its SLAs (for example, in terms of availability, delay, etc.) supplies both the Providers and the Consumers with information such as the workload generated by the latter and the performance and QoS offered through the Cloud, also allowing to implement mechanisms to prevent or recover violations (for both the Provider and Consumers). Monitoring is clearly instrumental for all the activities covered by the role of Cloud Auditor. In more general terms, Cloud Computing involves many activities for which monitoring is an essential task. In this Section we carefully analyze such activities, underlining the role of monitoring for each of them.

<u>Need for Cloud Monitoring:</u>

Capacity and resource planning

One of the most challenging tasks for application and service developers, before the large scale adoption of Cloud Computing, has always been resource and capacity planning (e.g. Web Services). In order to guarantee the performance required by applications and services, developers have to (i) quantify capacity and resources (e.g. CPU, memory, storage, etc.) to be purchased, depending on how such applications and services are designed and implemented, and (ii) determine the estimated workload. However, while an estimation can be obtained through static analysis, testing and monitoring, the real values are unpredictable and highly variable. Cloud Service Providers usually offer guarantees in terms of QoS and thus of resources and capacity for their services as specified in SLAs, and they are in charge of their resource and capacity planning so that service and application developers do not have to worry about them. To this end, monitoring becomes essential for Cloud Service Providers to predict and keep track of the evolution of all the parameters involved in the process of QoS assurance in order to properly plan their infrastructure and resources for respecting the SLAs.

Capacity and resource management

The first step to manage a complex system like a Cloud consists in having a monitoring system able to accurately capture its state. Over the years, virtualization has become a key component to implement Cloud Computing. Hiding the high heterogeneity of resources of the physical infrastructure, virtualization technologies introduced another complexity level for the infrastructure provider, which has to manage both physical and virtualized resources. Virtualized resources may migrate from a physical machine to another at any time. Hence, in Cloud Computing scenarios (specially in mobile ones) monitoring is necessary to cope with volatility of resources and fast-changing network conditions (which may lead to faults). In the context of public critical services (e.g., healthcare or other strategic applications), when using IaaS, concerns about QoS and QoP (Quality of Protection) become very critical. Indeed, when adopting Cloud infrastructures, companies and people expect such services to have 100% up time. Thus, a resilient and trustworthy monitoring of the entire Cloud infrastructures is needed to provide availability.

Data center management

Cloud services are provided through large scale data centers, whose management is a very important activity. Actually, this activity is part of resource management and we reported it here because of its importance and of its peculiar requirements. Data center management activities (e.g. data center control) imply two fundamental tasks: (i) monitoring, that keeps track of desired hardware and software metrics; and (ii) data analysis, which processes such metrics to infer system or application states for resource provisioning, troubleshooting, or other management actions. In order to properly manage such data centers, both monitoring and data analysis tasks must support real-time operation and scale up to tens of thousands of heterogeneous nodes, dealing with complex network topologies and I/O structures. In this context energy-efficiency is a major driver of monitoring data analysis for planning, provisioning and management of resources.

SLA management

The unprecedented flexibility in terms of resource management provided by Cloud Computing calls for new programming models in which Cloud applications can take advantage of such new feature, whose underlying premise is monitoring. Moreover, monitoring is mandatory and instrumental in certifying SLA compliance when auditing activities are performed to respect regulation (e.g. when government data or services are involved). Finally, monitoring may allow Cloud Providers to formulate more realistic and dynamic SLAs and better pricing models by exploiting the knowledge of user-perceived performance

Billing

One of the essential characteristics of Cloud Computing is the offer of ‘‘measured services’’, allowing the Consumer to pay proportionally to the use of the service with different metrics and different granularity, according to the type of service and the price model adopted.

With reference to the service models reported in Section 2, examples of billing criteria are: for SaaS, the number of contemporary users, or the total user base, or application-specific performance levels and functions; in PaaS services, the CPU utilization, or the task completion time; for

IaaS, the number of VMs, possibly varying with different CPU/Memory setups; we refer the interested reader to for a review of theoretical pricing models. For each of the reported pricing models and service models, monitoring is necessary both from the Provider side for billing, and from the Consumer side for verifying his own usage and to compare different Providers, a non-trivial process requiring monitoring functionalities and tools. When the billing granularity is coarse e.g. per VM in IaaS, or up to a maximum database size for a SaaS the pricing is considered a ‘‘flat rate’’, depending on the subscription duration, and the required monitoring is relatively basic. A significantly more complex scenario is the presence of a Cloud Service Broker in this case advanced monitoring is of paramount importance for the resource provisioning and charge-back strategies at the base of the Cloud Broker’s business.

Troubleshooting

The complex infrastructure of a Cloud represents a big challenge for troubleshooting (e.g. root cause analysis), as the cause of the problem has to be searched in several possible components (e.g. network, host, etc.), each of them made of several layers (e.g. real and virtual hardware, host and guest OS, etc.). A comprehensive, reliable and timely monitoring platform is therefore needed for Providers to understand where to locate the problem inside their complex infrastructure and for Consumers to understand if any occurring performance issue or failure is caused by the Provider, network infrastructure, or by the application itself.

Performance management

Being the hardware infrastructure maintenance delegated to the Providers, the Cloud Computing model is attractive for most Consumers (primarily medium sized enterprises and research groups). However, despite the attention paid by Providers, some Cloud nodes may attain performance orders of magnitude worse than other nodes. If a Consumer adopts a public Cloud to host a mission-critical service or for a scientific application, performance variability and availability become a concern. Therefore, from a Consumer’s perspective, monitoring the perceived performance is necessary to adapt to the changes or to apply corrective measures.

For instance, a Consumer may decide to host applications at multiple Clouds to ensure high-availability, switching between Clouds depending on the measured performance. Monitoring is then necessary since it may considerably improve the performance of real applications and affect activity planning and repeatability of experiments.

Security management

Cloud security is very important for a number of reasons. Security is considered as one of the most significant obstacles to the spread of Cloud Computing, especially considering certain kinds of applications (e.g. business-critical ones) and Consumers (e.g. governments). Different works in literature have provided reviews and recommendations for Cloud security (e.g. the references therein). For managing the security in Cloud infrastructures and services, proper monitoring systems are needed. Moreover, for hosting critical services for public agencies, Clouds have to satisfy strict regulations and prove it. And this can be done through a monitoring system that enables auditing (e.g. to certify the compliance to regulations and obligations, such as keeping data of a user inside country borders).

<u>Basic concepts of cloud computing:</u>

Cloud monitoring is needed to continuously measure and assess infrastructure or application behaviors in terms of performance, reliability, power usage, ability to meet SLAs, security, etc., to perform business analytics, for improving the operation of systems and applications, and for several other activities. In this section we introduce a number of concepts at the base of Cloud monitoring that are used to set the context for the following sections, we report these concepts in a taxonomy we propose for main aspects of Cloud monitoring we consider in this paper.

Layers

According to the work of the Cloud Security Alliance, a Cloud can be modeled in seven layers: facility, network, hardware, OS, middleware, application, and the user. Considering the roles defined in Section 2, these layers can be controlled by either a Cloud Service Provider or a Cloud Service Consumer. They are detailed in the following: Facility: at this layer we consider the physical infrastructure comprising the data centers that host the computing and networking equipment. Network: at this layer we consider the network links and paths both in the Cloud and between the Cloud and the user. Hardware: at this layer we consider the physical components of the computing and networking equipment. Operating System (OS): at this layer we consider the software components forming the operating system of both the host (the OS running on the physical machine) and the user (the OS running in the virtual machine). Middleware: at this layer we consider the software layer between the OS and the user application. It is typically present only in the

Cloud systems offering SaaS and PaaS service models. Application: at this layer we consider the application run by the user of the Cloud system. User: at this layer we consider the final user of the Cloud system and the applications that run outside the Cloud (e.g. a web browser running on a host at the user’s premise). In the context of Cloud monitoring, these layers can be seen as where to put the probes of the monitoring system. In fact, the layer at which the probes are located has direct consequences on the phenomena that can be monitored and observed. Orthogonally to these layers, system-wide and guest wide measurements, as proposed by Du et al. in the context of profiling virtual machines, can be defined when discussing what can be monitored inside and what can be monitored outside a Cloud system. Besides, due to the very high complexity of Cloud systems, it not possible to be sure that certain phenomena are actually observed or not. For example, if we put a probe into an application that runs into the Cloud, to collect information on the rate at which it exchanges information with other applications running in the same Cloud, we do not necessarily know if this rate comprises also the transfer rate of the network. It depends on if the two applications run on the same physical host or not, and this information is not always exposed by the Provider. Similar issues arise for evaluating the performance of computation: the time required for a task completion can depend on the actual hardware that is executing the instructions (usually exposed only as a CPU model – equivalent) and on the workload due to other virtualized environments running on the same physical server (which are not exposed at all to the Consumer).

Abstraction levels

In Cloud Computing, we can have both high and low level monitoring, and both are required. High-level monitoring is related to information on the status of the virtual platform. This information is collected at the middleware, application and user layers by Providers or Consumers through platforms and services operated by themselves or by third parties. In the case of SaaS, high-level monitoring information is generally of more interest for the Consumer than for the Provider (being closely related to the QoS experienced by the former). On the other hand, low-level monitoring is related to information collected by the Provider and usually not exposed to the Consumer, and it is more concerned with the status of the physical infrastructure of the whole Cloud (e.g. servers and storage areas, etc.). In the context of IaaS, both levels are of interest for both Consumers and Providers. More precisely, for low-level monitoring specific utilities collect information at the hardware layer (e.g., in terms of CPU, memory, temperature, voltage, workload, etc.), at the operating system layer and at middleware layer (e.g., bug and software vulnerabilities), at the network layer (e.g., on the security of the entire infrastructure through firewall, IDS and IPS), and at the facility layer (e.g. on the physical security of involved facilities through monitoring of data center rooms using video surveillance and authentication systems).

Tests and metrics

Monitoring tests can be divided in two main categories: Computation-based and Network-based. Computation-based tests are related to monitoring activities aimed at gaining knowledge about and at inferring the status of real or virtualized platforms running Cloud applications.

Computation-based

Tests are related to the following metrics: server throughput, defined as the number of requests (e.g. web page retrieval) per second; CPU Speed; CPU time per execution, defined as the CPU time of a single execution; CPU utilization, defined as the CPU occupation of each virtual machine (useful to monitor the concurrent use of a single machine by several VMs); memory page exchanges per second, defined as the number of memory pages per second exchanged through the I/O; memory page exchanges per execution, defined as the number of memory pages used during an execution; disk/memory throughput; throughput/delay of message passing between processes; duration of specific predefined tasks; response time; VM startup time; VM acquisition/release time; execution/access time, up-time. All of them can be evaluated in terms of classical statistical indicators (mean, median, etc.) as well as in terms of temporal characterization and therefore stability/variability/predictability. Computation-based tests are operated by the provider or sometimes demanded to third parties. For example, in the case of EC2 and Google App Engine, Hyperic Inc publishes results of these test on CloudStatus.

Network-based

Tests are related to the monitoring of network-layer metrics. This set includes round-trip time (RTT), jitter, throughput, packet/data loss, available bandwidth, capacity, traffic volume, etc. Using these metrics, several experimental studies in literature compared legacy web-hosting and Cloud-based hosting.

A note on Cluster vs Grid vs Cloud monitoring

Similarities and overlapping of properties among Cloud Computing and previous distributed paradigms have led to deep discussion on the definition of Cloud Computing and its peculiar characteristics here we consider the differences from the point of view of monitoring. Compared with the case of Grid Computing, the monitoring of a Cloud is more complex because of the differences in both the trust model and the view on resource/services presented to the user. In fact the main objective of a Grid is the sharing of resources across multiple organizations, implying simpler accounting criteria and limited resource abstraction, which creates a simple relation between monitoring parameters and physical resource status.

On the other hand, for the Cloud, the presence of multiple layers and service paradigms leads to high abstraction of resources, resulting in more opaque relationship between the layer- or service-specific observables and the underlying resources. Moreover we expressly note that in Cloud Computing, even if the abstract interfaces offered to a Consumer could apparently require a reduced necessity for monitoring with respect to Grid, in reality such need is pushed on the Provider of the service, that has to cope with promised or expected performance and with optimization of resources in a highly dynamic and heterogeneous scenario. This gap of objectives and transparency has to be filled when adopting for a Cloud a monitoring system coming from the Grid Computing field. Finally, as noted in previous sections, the ‘‘on demand’’ service paradigm poses additional challenges to monitoring systems not designed for high churning of both users and resources. Most of the monitoring approaches and platforms proposed for the Grid case have been customized for Cloud systems. Zanikolas et al. surveyed the Grid monitoring research field by introducing the involved concepts, requirements, phases, and related standardization activities (e.g. Global Grid Forum’s Grid Monitoring Architecture). Furthermore, they proposed a taxonomy – built by considering scope, scalability, generality and flexibility – of Grid monitoring systems aiming at classifying a wide range of projects and frameworks. In the next section we thoroughly discuss the issues and the proposed solutions regarding the adoption in the Cloud scenario of systems designed for slowly changing fixed infrastructure. These aspects have to be taken into account when considering Ganglia, Nagios, MonaLisa, R-GMA and GridICE and similar systems to monitor a Cloud. All those differences are even more stressed when comparing Cloud paradigm to Cluster Computing in this case the relatively rigid architecture, the limited possibilities of service negotiation and the low automation of resource provisioning make Clusters comparable to a base technology for Cloud IaaS Providers, and lead to requirements in terms of monitoring that are a limited subset of the ones of a Cloud. Therefore, most characterizing properties for Cloud monitoring systems either do not apply for Clusters or Grids (namely Elasticity, Adaptability, Automaticity) or are not vital for their purpose (Comprehensiveness, Extensibility and Intrusiveness).

Cloud monitoring: properties and related issues:

In order to operate properly, a distributed monitoring system is required to have several properties that, when considered in the Cloud Computing scenario, introduce new issues. In this Section we define and motivate such properties, analyze the issues arising from them, and discuss how these issues have been addressed in literature. We report these properties in a taxonomy of main aspects regarding Cloud monitoring considered in this paper. We illustrate the research issues associated with each of the properties considered.

Scalability

A monitoring system is scalable if it can cope with a large number of probes. Such property is very important in Cloud Computing scenarios due to the large number of parameters to be monitored about a huge number of resources. This importance is amplified by the adoption of virtualization technologies, which allow to allocate many virtual resources on top of a single physical resource. The measurements required to obtain a comprehensive view on the status of the Cloud lead to the generation of a very large volume of data coming from multiple distributed locations. Hence, a scalable monitoring system should be able to efficiently collect, transfer and analyze such volume of data without impairing the normal operations of the Cloud. In literature such issue has been mainly addressed by proposing architectures in which monitoring data and events are propagated to the control application after their aggregation and filtering, in order to reduce their volume: aggregation combines multiple metrics into a synthetic one that is inferred or not directly monitored; filtering avoids useless data to be propagated to the control application. Most of the proposed architectures, regardless of the specific low-level or high-level monitored parameters, adopt a subsystem to propagate event announcements or rely on agents, which are responsible for performing data collection, filtering and aggregation. In this context, different aggregation strategies have been proposed: extraction of high-level performance metrics by means of machine learning algorithms extraction of predicted parameters by combining metrics from different layers (hardware, OS, application and user) and by applying Kalman filters linear combination of OS-layer metrics. Some architectures further improve scalability by adopting additional optimizations: efficient algorithms for agent deployment and interconnection. Content Based Routing (CBR) and Complex Event Processing (CEP) facilities lightweight analysis close to the data source, adjustable sampling, time-based filtering, and ad hoc collection and aggregation strategies applied to different partitions of the monitored system.

Elasticity

A monitoring system is elastic if it can cope with dynamic changes of monitored entities, so that virtual resources created and destroyed by expansion and contraction are monitored correctly. Such property, also referred to as dynamism, implies scalability and adds to it the requirement of supporting on-line upsizing or downsizing of the pool of monitored resources. As opposed to the static nature of previous computing paradigms (e.g. Grid computing), Cloud Computing requires its resources to be dynamic, thus making elasticity an essential property for its monitoring system, as derived from three main drivers: varying assignment of resources to users, varying monitoring requirements for the user, and varying presence of users (multi-tenant scenarios). A challenge in providing elasticity is related with the fact that it is a new fundamental property introduced by Cloud monitoring and not previously considered as a requirement for monitoring generic distributed systems.

Therefore, many different monitoring systems proposed for large distributed systems (e.g. Ganglia, Nagios) have been designed for a relatively slowly changing physical infrastructure. Thus, they do not assume or support a rapidly changing dynamic infrastructure and they are not suitable for as-is adoption in Cloud scenarios. In literature, a number of extensions to traditional monitoring systems have been proposed to address this challenge. They basically added the support for monitoring virtualized resources and the condition-triggered reporting in a push fashion, often exploiting a publish-subscribe paradigm to decouple communication ends and to support dynamism. In order to cope with migration of virtual resources, in the Lattice platform a hypervisor controller is responsible of tracking the presence of virtual execution environments (VEEs) by obtaining a list of running VEEs from the hypervisor, on a regular basis. Analyzing such list the controller determines (i) if there is a new VEE, in which case it adds a new probe for that VEE, or (ii) if a VEE has been shutdown, in which case it deletes the probe for that VEE.A more comprehensive solution has been provided by Carvalho and Granville, which makes Nagios aware of machine virtualization using both active checks (a remote execution feature that realizes a pull communication paradigm) and passive checks (physical hosts notifying the Nagios server about the VMs that are currently running, implementing a push communication paradigm). An analogous extension to Nagiosis provided with a RESTful Event Brokering module, which allows the monitoring of both physical and virtual infrastructures; elasticity is obtained by exploiting the design patterns of a traditional service-oriented architecture to realize a twofold push–pull model: monitoring information is pushed by agents towards the management layer and information consumers can pull data fromit. More complex solutions are possible when the Cloud is taken into account in the design of the monitoring system. For example, Monalytics has been designed for scalability and effectiveness in heavily dynamic scenarios, providing among the other features: runtime discovery of the monitored resources and runtime configuration of the monitoring agents. These features are obtained by means of an election-based hierarchy of brokers that collect, process and transmit monitoring information, where the communication topology and the kind of computations are dynamically modified according to the status of the monitored resources.

Adaptability

A monitoring system is adaptable if it can adapt to varying computational and network loads in order not to be invasive (i.e. impeding for other activities). Due to the complexity and the dynamism of the Cloud scenarios, adaptability is fundamental for a monitoring system in order to avoid as much as possible a negative impact of monitoring activities on normal Cloud operations, especially when active measurements are involved. The workload generated by active measurements, together with the collection, processing, transmission and storage of monitoring data and the management of the monitoring subsystem, require computing and communication resources and constitute a cost for the Cloud infrastructure.

Thus, the ability to tune the monitoring activities according to suitable policies is of significant importance to meet Cloud management goals. Providing adaptability is not trivial, because it requires to quickly react to load changes, maintaining the right trade-off between precision (e.g. predictable latencies) and invasivity. In literature, such issue has been faced by several studies by tuning the amount of monitored resources and the monitoring frequency. For instance, Park et al. presented an approach based on Markov Chains, to analyze and predict resource states, in order to adaptively set a suitable time interval to push monitoring information. Another example is the Monalytics system, which configures its agents according to the monitoring topologies (i.e. those used to collect, process and transmit monitoring data), modeled as Dynamic Computational Communication Graphs (DCG). Depending on the workload, it allows to configure the existing agents in real time – providing them with new monitoring and analysis codes or changing the methods being used – and to dynamically discover and attach to new data sources. The latter approach is further analyzed by Wang et al., who assessed and compared the performance of such dynamic topologies against traditional ones, in terms of time-to-in-sight (see the following section on Timeliness) and management cost (modeled as capital cost for hardware and associated software management), showing that this approach is both flexible and performance/cost effective.

Timeliness

A monitoring system is timely if detected events are available on time for their intended use. Monitoring is instrumental to activities related with core goals of a Consumer or a Provider, hence failing to get the necessary information on time for the appropriate response (e.g. to raise an alarm, to provision more resources, to migrate services, to enforce a different policy) would void the usefulness of monitoring itself. Timeliness is interdependent with other properties of the monitoring system, such as Elasticity, Autonomicity and Adaptability. Thus, granting it implies the same challenges or trade-offs between opposing requirements. More in detail, the time between the occurrence of an event and its notification can be broken down in different contributions: sampling, analysis and communication delay. Each of them poses some issues. The shorter the sampling interval, the smaller is the delay between the time a monitored condition happens and is captured. Thus, to obtain up-to-date information, a trade-off between Accuracy and sampling frequency is necessary, considering also the resource constraints (e.g. CPU, network bandwidth or memory). The analysis delay poses a similar issue regarding complex events (i.e. the result of a computation over multiple parameters), which requires to consider also the time to get all the necessary information, besides the computing time itself. Finally, being the Cloud a distributed system, the communication delay can be significant because the information may have to travel across multiple links to reach processing nodes and this delay is even more important when considering complex events involving information coming from remote sources.

In literature, the choice of the sampling interval has been considered by Park et al. in order to cope with highly volatile resources (of mobile devices) a behavioral model of the resource is used to predict the suitable interval duration. The problem of keeping the communication and analysis delays as low as possible has been considered by Wang et al., who proposed a close-to-the-data analysis approach realized by performing computations on information gathered by nearby nodes and by adapting the communication and analysis topology to meet low delay goals. In order to evaluate the Timeliness, they defined the Time to Insight metric as ‘‘the latency between when one monitoring sample (indicating event of interests) is collected on each node and when the analysis on all of those monitoring samples has completed’’. Such metric is then used to evaluate different communication and analysis topologies and the trade-offs with the related infrastructure costs.

Autonomicity

An autonomic monitoring system is able to self-manage its distributed resources by automatically reacting to unpredictable changes, while hiding intrinsic complexity to Providers and Consumers. As Cloud infrastructures are meant to provide on-demand self-service and rapid elasticity while operating continuously with minimal service interruptions, it is extremely important for the monitoring system to be able to react to detected changes, faults and performance degradation, without manual intervention. Supporting autonomicity in such a monitoring system is not trivial, since it requires to implement a control loop that receives inputs from a huge number of sensors (i.e. the monitoring data) and propagates control actions to a large number of distributed actuators. This in turn implies elasticity and Timeliness. Moreover the analysis capabilities for situation awareness must be implemented (the complexity and layering of Cloud infrastructure pose obstacles to this) and the definition of suitable policies to drive the behavior of the monitoring system in response to the detected events is necessary.

For example, focusing on bottlenecks and over-provisioning for a multi-tier Web application hosted on a Cloud, Iqbal et al. proposed two methodologies for the same. Such methodologies are driven by maximum response time requirements and are shown to be useful to provide SLAs. About system failures, Ayad and Dippel propose an agent-based monitoring system that continuously checks for the availability of VMs and automatically recovers them in case of failures. In order to automatically cope with SLA violations, the DeSVi architecture allocates computing resources for a requested service based on user requests, and arranges its deployment on virtualized infrastructures. Resources are monitored using a framework capable of mapping low-level resource metrics (e.g. host up- and down-times) to user-defined SLAs (e.g. service availability). The detection of possible SLA violations is performed using predefined service level objectives and knowledge databases for the analysis of the monitored information, thus allowing to automatically propose reactive actions.

Comprehensiveness, Extensibility and Intrusiveness

A monitoring system is comprehensive if it supports different types of resources (both physical and virtualized), several kinds of monitoring data, and multiple tenants it is extensible if such support can easily be extended (e.g., through plug-ins or functional modules); it intrusive if its adoption requires significant modification to the Cloud. The first two properties are strictly related: the latter represents the possibility to enhance the former without modifying the monitoring framework. Having a comprehensive monitoring system is useful for both developers (IaaS and PaaS Consumers) and their respective Providers. The advantage for the former is related to the possibility to adopt a single monitoring API, independently of what kind of monitoring information is actually used. For the latter, the advantage consists in deploying and maintaining only one single monitoring infrastructure. By also providing extensibility, such advantages can easily persist to changes or additions of underlying components, and maintaining low intrusiveness allows to minimize the instrumentation costs.

Cloud Computing is a relatively new paradigm and no common standards have been widely adopted by deployed systems. Most non-Cloud-specific monitoring systems were already designed to provide extensibility and low intrusiveness, and their extension to Cloud scenarios retained such features. Some issues arise when considering comprehensiveness. A first issue is related to the fact that a holistic monitoring system has to support different underlying architectures, technologies, and resources, while preserving isolation among different tenants.

On the other side, a comprehensive monitoring system allows to better perform troubleshooting activities, which raises another issue due to the intrinsic dynamicity of Cloud environments and to the large number and heterogeneity of resources and parameters considered at different layers. The preservation of isolation has been explicitly addressed for the first time in literature by Hasselmeyer and d'Heureuse in their agent-based architecture. It achieves isolation in terms of tenant visibility by directing monitoring information flows through the same stream management system, which exposes the information only to the intended recipients. Moreover, in order to allow for interoperability of the functional blocks, these are connected with adapters that abstract the data from the specific technologies. Regarding the support for heterogeneous virtualized environments, the VM Driver monitoring subsystem has been proposed for the interception of events occurring at the VM level (set at the OS layer). This allows to monitor the state of virtual machines hiding the differences of guest OSes. About the issues implied in troubleshooting large numbers of dynamic and heterogeneous components, several studies have been car ried out to understand the cause of the performance observed in Cloud environments. Most of them selected Amazon EC2 as a case study. Hilland Humphrey were unable to identify the cause of the performance observed for scientific applications.

For instance, in their experiments, the maximum rate at which two processes were exchanging information could be attributed to different causes (network, L2 caches, etc.), depending on where the processes were running (indifferent hosts in the same or different data centers, in the same host, etc.), on the number of concurrent processes (and therefore VMs) on the same host, etc. Wang and Ng found that, even when the data center network is lightly utilized, virtualization can still cause significant throughput instability and abnormal jitter, and identified the processor sharing mechanism as the main responsible. Schad et al. found the performance observed at different layers (application and OS) significantly variable with time and VM instances, thus impacting data-intensive applications predictability and repeatability of wall-clock timed experiments.

Working on a small testbed, Mei et al. focused on the impact of co-locating applications in a virtualized Cloud in terms of throughput and resource sharing effectiveness. They found that in presence of idle instances, other VMs result to be scheduled less frequently for less time, which is primarily due to two factors: (i) the execution of timer tick for the idle guest domain and the context switch overhead, and (ii) the processing of network packets, such as address resolution protocol (ARP) packets, which causes I/O processing in guest domain. They observed also that the duration of performance degradation experienced due to the creation of new VMs on demand is typically bounded within 100s, and it is related with the machine capacity, the workload level in the running domain, and the number of new VM instances to start up. Finally, they found that co-locating two applications on VMs hosted on the same physical machine produces performance degradation when involving CPU intensive tasks and, when multiple guest domains are running, the context switches among them lead to more frequent cache and translation lookaside buffer (TLB) miss, which result in more CPU time consumption in serving the same data.

Resilience, Reliability, and Availability

A monitoring system is resilient when the persistence of service delivery can justifiably be trusted when facing changes, that basically means to withstand a number of component failures while continuing to operate normally; it is reliable when it can perform a required function under stated conditions for a specified period of time; it is available if it provides services according to the system design whenever users request them. As monitoring is functional to critical activities of the Cloud, such as billing, SLA compliance verification and resource management, the monitoring system has to be resilient, reliable and available in order not to compromise such activities.

With the heavy usage of virtualization technologies by Cloud platforms, monitored hosts and services can migrate from a physical computer to another, invalidating the classical monitoring logics and mining the reliability of the monitoring system.

Hence, the necessity to provide such properties for Cloud monitoring poses several issues, such as tracking and managing heterogeneous monitored and monitoring resources, characterizing possible faults of the monitoring system itself and protecting against them. Several research works considered different aspects regarding resilience. Some works acknowledged the resilience to faults in mobile Cloud Computing scenarios, where mobile devices which do not have significant computational power and storage space – are considered as a high volatile resource, and such volatility influences the choice of the monitoring frequency.

Ayad and Dippel considered resilience as affected by the adopted virtualization technologies. As for reliability, some researchers aimed at determining the performance of particular Clouds, for example when analyzing the performance of Amazon Web Services, experiencing difficulties related to the fact that the probes used for monitoring were not available in some periods.

Romano et al., proposed a Cloud monitoring facility suitable for QoS, called QoS-MONaaS, which stands for ‘‘Quality of Service MONitoring as a Service’’, that is specifically designed to be reliable and offers monitoring facilities ‘‘as a Service’’, allowing its user (Provider or Consumer) to describe in a formal SLA the Key Performance Indicators (KPIs) of interest and the alerts to be raised when an SLA breach is detected. The basis for its reliability is drawn from high-level facilities provided by an underlying platform (SRT-15), where anonymization is applied before processing monitoring data.

Finally, focusing on availability, Padhy et al. considered byzantine faults affecting a Cloud monitoring system and proposed a publish-subscribe paradigm for communication and event handling with redundant brokers, and leveraged Byzantine Fault Tolerance algorithms to ensure tolerance to attacks and failures.

Accuracy

We consider a monitoring system to be accurate when the measures it provides are accurate, i.e. they are as close as possible to the real value to be measured. The accuracy is important for any distributed monitoring system because it can heavily impact the activities that make use of the monitoring information. For instance, when the monitoring system is used for troubleshooting, inaccuracy in measure may lead to incorrect identification of the cause of the problem. In the context of Cloud Computing, accuracy becomes even more important. Firstly, since Cloud services are subject to well-defined SLAs, and Providers have to pay penalties to their customers in case of SLAs violations, inaccurate monitoring can lead to money loss. Secondly, being the monitoring system used for important activities of the Cloud, accurate monitoring is necessary to effectively and efficiently perform them.

The analysis of the literature reveals two main issues related to the accuracy of monitoring systems in Cloud Computing scenarios.

- The first one is related to the workload used to perform the measurements: in order to monitor the Cloud, especially when using active monitoring approaches, it is necessary to apply a suitable stress (e.g. the HTTP GET to a WEB server in the Cloud must arrive with a certain statistical distribution in order to accurately compute the average response time of the WEB server).
- The second issue is related to the virtualization techniques used in the Cloud: performed measurements may be affected by errors imputable to the virtualization systems that add additional layers between applications and physical resources (e.g. time-related measurements are impaired by the sharing of physical resources such as CPUs, interface cards, and buffers).

Several contributions have been provided in literature with regard to these two issues. As for the workload, research efforts in this area comprise the characterization of real workloads, the reproduction of such workload in the Cloud, which tests to perform and how, which parameters to measure, etc. A number of research groups carried out experimental campaigns on different Clouds to understand their performance, both in general and for specific applications. Several studies investigated the performance of specific Clouds in order to understand if and how they can support scientific and high performance applications. Most of these works can be located at the application layer because they used custom applications running in the Cloud.

Ostermann et al. performed also an analysis at user layer, thanks to emulated web browsers issuing requests to servers running in the Cloud. The range of metrics considered is very wide and includes the money cost for using the Cloud services, execution time of specific jobs, VM acquisition/release times, disk throughput, access time and throughput of memory, CPU speed, and throughput and/or latency of messages exchanged by the applications.

Moreover, most of these research works have been conducted on Amazon EC2 while a few others used also other kinds of testbeds, typically located at the researchers' premises. On the other hand, different works in literature studied the possibility to use Cloud for supporting database and service-oriented applications. Typically, these studies are conducted at application or user layer. The metrics considered by these works include CPU speed, disk throughput, VM startup time, throughput, jitter and loss of the network, memory throughput, server throughput and money cost. These research works have been conducted on several different commercial Clouds, including Amazon EC2, Google App Engine, Microsoft Azure, as well as on local testbeds.

Finally, Binnig et al. evidenced a number of limitations of the benchmarks used by many of the previously cited works. In particular, they suggest that aspects such as scalability, peak loads, and fault tolerance are not considered by current state-of-the-art benchmarks such as TPC-H for OLAP, TPC-C for OLTP, or TPC-W for e-commerce applications. The authors also propose a number of other tests and parameters to evaluate these important aspects of modern Clouds. As for the impact of virtualization on measurement accuracy, the works in literature analysed the accuracy of RTT, jitter, capacity and available bandwidth, topology, and also the performance of auto-tuning applications. Regarding the delay, jitter, capacity and available bandwidth, the problem consists in having accurate time stamping at the measuring nodes. Implementing VMs at the end nodes requires a timely scheduling and switching mechanism between the different VMs.

As a consequence, packets belonging to a specific VM may be queued until the physical system switches back to that VM, which leads to inaccurate time stamping. Some works reported that accurate RTT measurements are possible only under low network and computing loads, and that most delay is introduced while sending packets (as opposed to receiving packets). They conclude that kernel-space timestamps are not enough accurate under heavy network load, and access to time-stamps as seen by physical network interfaces would be necessary to overcome this issue.

Regarding topology measurements, Abujoda proved that network virtualization generates several virtual topologies on top of a single physical topology, and common active measurement tools like traceroute are unable to discover the real physical topology. Moreover, their accuracy is affected by the migration of nodes, which dynamically modifies the placement of virtual nodes and the distance among them.

Finally, regarding the performance of auto-tuning applications, Youseff et al. [90] showed that the combination of ATLAS auto-tuning and Xen paravirtualization delivers native execution performance and nearly identical memory hierarchy performance profiles. Moreover, they showed that paravirtualization has no significant impact on the performance of memory-intensive applications, even when memory becomes a scarce resource.

2.3 Libvirt

Libvirt is an open source API, daemon and management tool for managing platform virtualization. It can be used to manage KVM, Xen, VMware ESX, QEMU and other virtualization technologies. These APIs are widely used in the orchestration layer of hypervisors in the development of a cloud-based solution.

Libvirt itself is a C library, but it has bindings in other languages, notably in Python, Perl, OCaml, Ruby, Java, and PHP. Libvirt for these programming languages is composed of wrappers around another class/package called libvirtmod. *libvirtmod* implementation is closely associated with its counterpart in C/C++ in syntax and functionality.

Libvirt is used by various virtualization programs and platforms. Graphical Interfaces are provided by Virtual Machine Manager and others. The most popular command line interface is 'virsh', and higher level tools like oVirt.

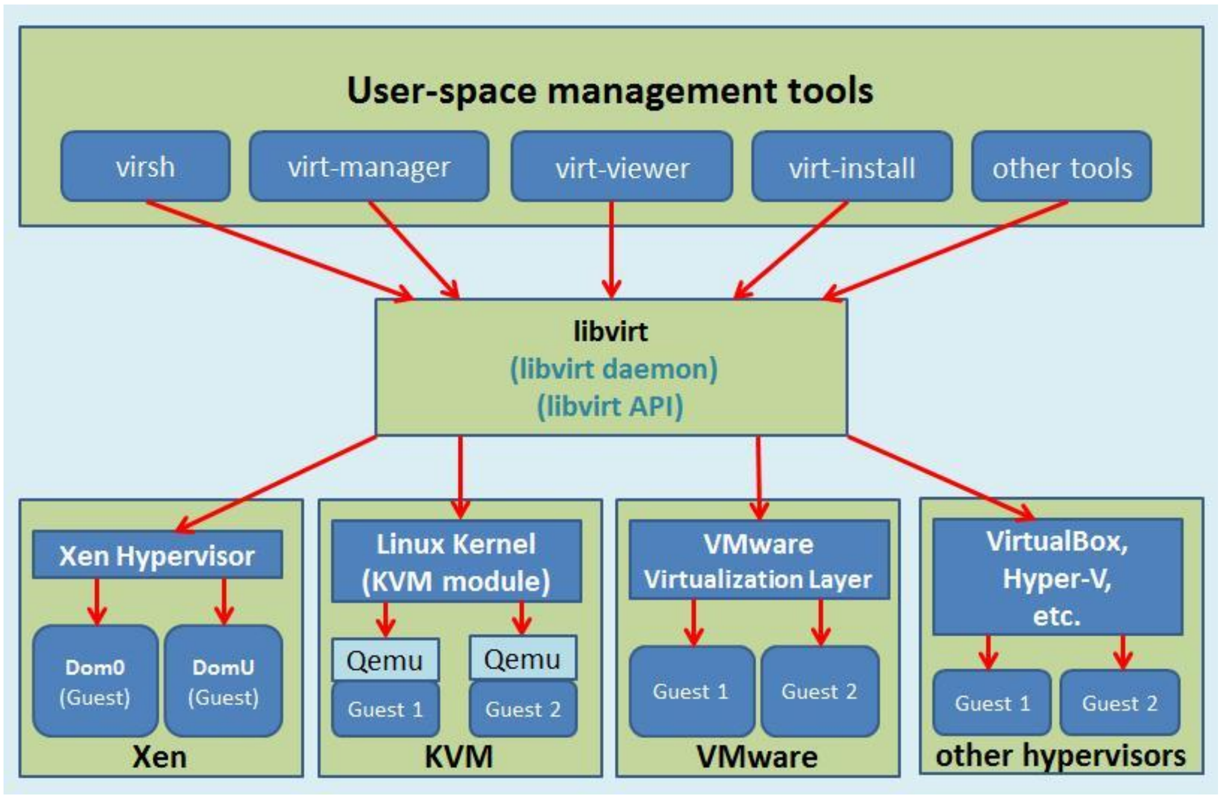


Fig (2-5) Libvirt architecture

Development of libvirt is backed by Red Hat, with significant contributions by other organisations and individuals. Libvirt is available on most Linux distributions; remote servers are also accessible from Apple Mac OS X and Microsoft Windows clients.

Virtualization is defined as the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, operating systems, storage devices, and computer network resources.

Libvirt is an open-source collection of software that provides a convenient way to manage virtual machines and other virtualization functionality, such as storage and network interface management.

Libvirt can be used to manage different hypervisors using same programming interface. It includes an API library, a libvirt daemon (libvirtd), and virsh, a command line utility for management. It is primarily written in C language and is a C toolkit. Bindings are also available in other languages such as Python, Java, Ruby and C #.

When it comes to scale-out computing (such as cloud computing), libvirt may be one of the most important libraries you've never heard of. Libvirt provides a hypervisor-agnostic API to securely manage guest operating systems running on a host. Libvirt isn't a tool *per se* but an API to build tools to manage guest operating systems. Libvirt itself is built on the idea of abstraction. It provides a common API for common functionality that the supported hypervisors implement. Libvirt was originally designed as a management API for Xen, but it has since been extended to support a number of hypervisors.

2.4 PyQt5

PyQt is a Python binding of the cross-platform GUI toolkit Qt. It is one of Python's options for GUI programming. Popular alternatives are PySide (the Qt binding with official support and a more liberal licence), PyGTK, wxPython, and Tkinter (which is bundled with Python). Like Qt, PyQt is free software. PyQt is implemented as a Python plug-in.

PyQt is developed by the British firm Riverbank Computing. It is available under similar terms to Qt versions older than 4.5; this means a variety of licenses including GNU General Public License (GPL) and commercial license, but not the GNU Lesser General Public License (LGPL). PyQt supports Microsoft Windows as well as various flavours of UNIX, including Linux and OS X.

PyQt implements around 440 classes and over 6,000 functions and methods including:

- a substantial set of GUI widgets
- classes for accessing SQL databases (ODBC, MySQL, PostgreSQL, Oracle, SQLite)
- QScintilla, Scintilla-based rich text editor widget
- data aware widgets that are automatically populated from a database
- an XML parser
- SVG support
- classes for embedding ActiveX controls on Windows (only in commercial version)

To automatically generate these bindings, Phil Thompson developed the tool SIP, which is also used in other projects.

In August 2009, Nokia, the then owners of the Qt toolkit, released PySide, providing similar functionality, but under theLGPL after failing to reach an agreement with Riverbank Computing to change its licensing terms to include LGPL as an alternative license.

Some components of PyQt are:

The *QtCore* module contains the core non-GUI classes, including the event loop and Qt's signal and slot mechanism. It also includes platform independent abstractions for Unicode, threads, mapped files, shared memory, regular expressions, and user and application settings.

The *QtGui* module contains the majority of the GUI classes. These include a number of table, tree and list classes based on the model–view–controller design pattern. Also provided is a sophisticated 2D canvas widget capable of storing thousands of items including ordinary widgets.

The *QtNetwork* module contains classes for writing UDP and TCP clients and servers. It includes classes that implement FTP and HTTP clients and support DNS lookups. Network events are integrated with the event loop making it very easy to develop networked applications.

The *QtOpenGL* module contains classes that enable the use of OpenGL in rendering 3D graphics in PyQt applications.

The *QtSql* module contains classes that integrate with open-source and proprietary SQL databases. It includes editable data models for database tables that can be used with GUI classes. It also includes an implementation of SQLite.

The *QtSvg* module contains classes for displaying the contents of SVG files. It supports the static features of SVG 1.2 Tiny.

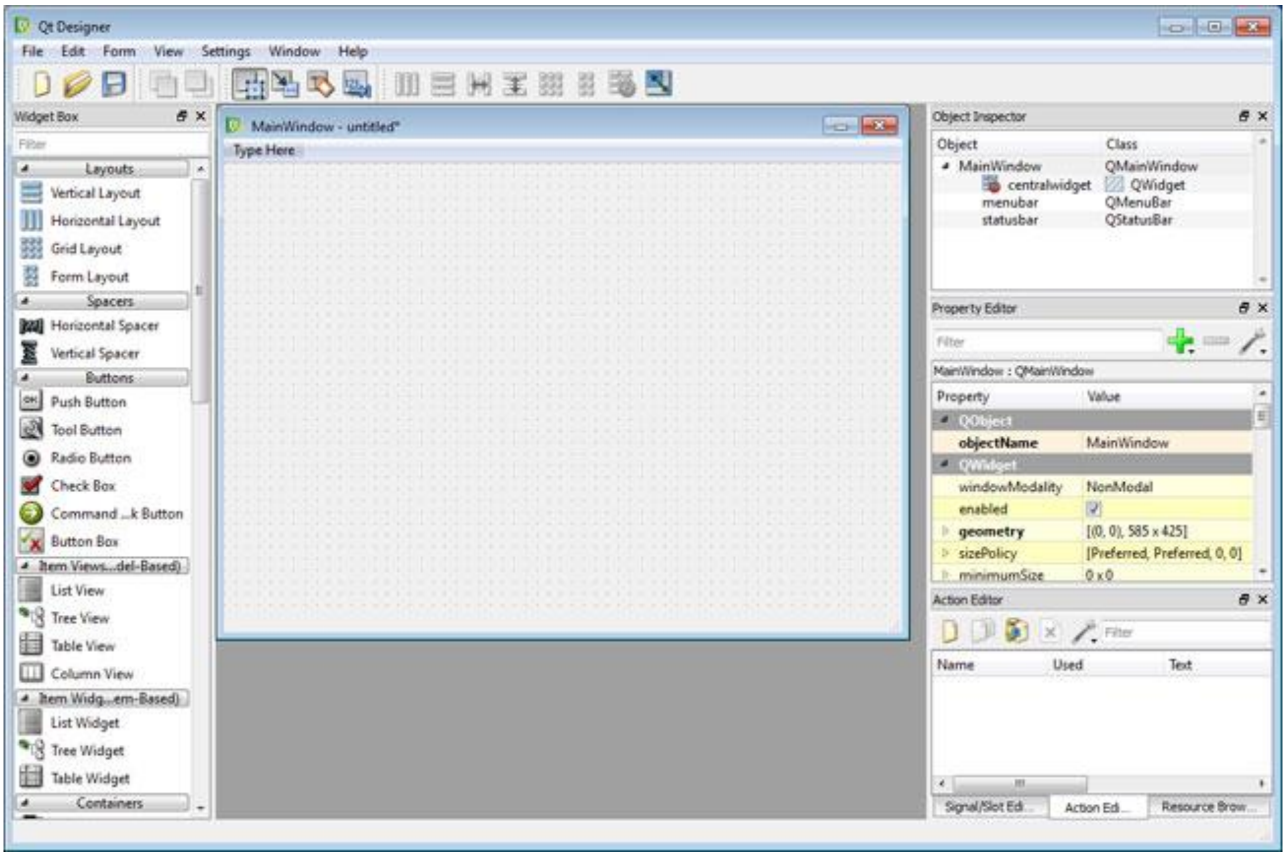
The *QtXml* module implements SAX and DOM interfaces to Qt's XML parser.

The *QtMultimedia* module implements low-level multimedia functionality. Application developers would normally use the phonon module.

The *QtDesigner* module contains classes that allow Qt Designer to be extended using PyQt.

The *Qt* module consolidates the classes contained in all of the modules described above into a single module. This has the advantage that you don't have to worry about which underlying module contains a particular class. It has the disadvantage that it loads the whole of the Qt framework, thereby increasing the memory footprint of an application. Whether you use this consolidated module, or the individual component modules is down to personal taste.

The *uic* module implements support for handling the XML files created by Qt Designer that describe the whole or part of a graphical user interface. It includes classes that load an XML file and render it directly, and classes that generate Python code from an XML file for later execution.



Object	Class
MainWindow	QMainWindow
centralwidget	QWidget
menubar	QMenuBar
statusbar	QStatusBar

MainWindow : QMainWindow

Property	Value
QObject	
objectName	MainWindow
QWidget	
windowModality	NonModal
enabled	✓
geometry	[(0, 0), 585 x 425]
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	0 x 0

Fig (2-6) Qt Designer

2.5 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB. SciPy makes use of matplotlib.

Matplotlib was originally written by John D. Hunter, has an active development community,[1] and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in 2012.

As of 30 October 2015, matplotlib 1.5.x supports Python versions 2.7 through 3.5. Matplotlib 1.2 is the first version of matplotlib to support Python 3.x. Matplotlib 1.4 is the last version of matplotlib to support Python 2.6.

The pylab interface makes matplotlib easy to learn for experienced MATLAB users, making it a viable alternative to MATLAB as a teaching tool for numerical mathematics and signal processing.

Some of the advantages of the combination of Python, NumPy, and matplotlib over MATLAB include:

Based on Python, a full-featured modern object-oriented programming language suitable for large-scale software development

Free, open source, no license servers

Native SVG support

Typically pylab is imported to bring NumPy and matplotlib into a single global namespace for the most MATLAB like syntax, however a more explicit import style, which names both matplotlib and NumPy, is the preferred coding style.

Advantages of using matplotlib

Default plot styles with built-in code

Deep integration with Python

Matlab style programming interface

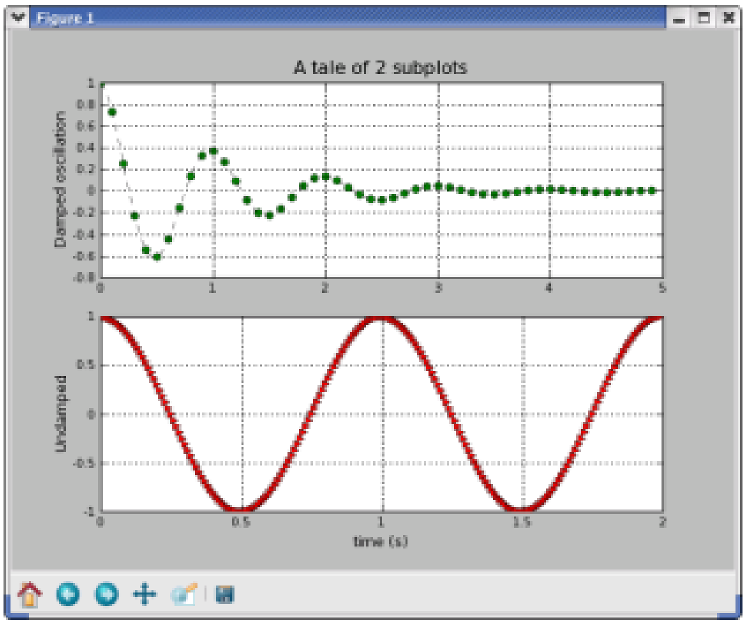


Fig (2-7) A matplotlib graph

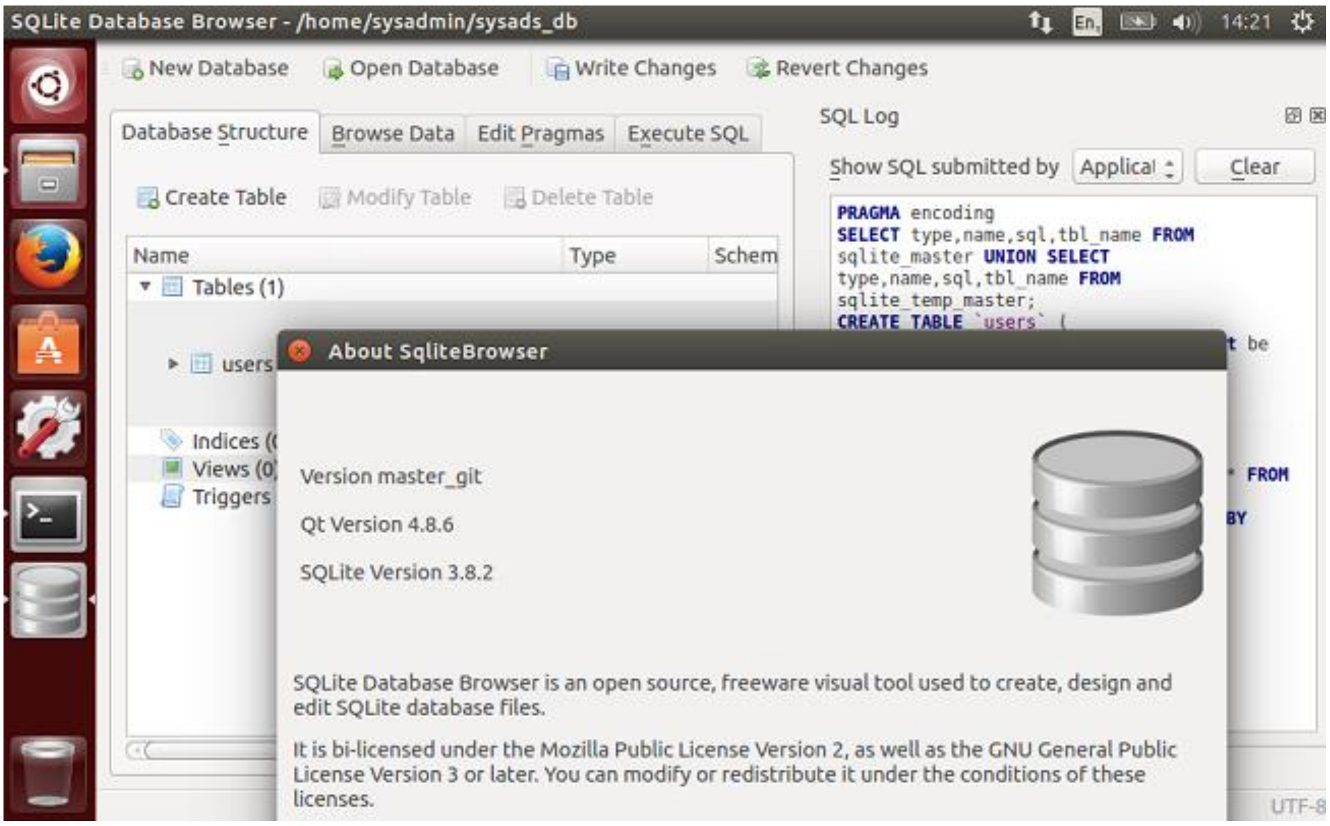
2.6 SQLite3

SQLite is a software library (cross platform C library) that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain.

In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

SQLite is ACID-compliant and implements most of the SQL standard, using a dynamically and weakly typed SQL syntax that does not guarantee the domain integrity.

The SQLite3 can be integrated with Python using sqlite3 module which was written by Gerhard Haring. It provides an SQL interface compliant with the DB-API 2.0 specification described by PEP 249.



```
PRAGMA encoding
SELECT type,name,sql,tbl_name FROM
sqlite_master UNION SELECT
type,name,sql,tbl_name FROM
sqlite_temp_master;
CREATE TABLE `users` (
```

Version master_git

Qt Version 4.8.6

SQLite Version 3.8.2

SQLite Database Browser is an open source, freeware visual tool used to create, design and edit SQLite database files.

It is bi-licensed under the Mozilla Public License Version 2, as well as the GNU General Public License Version 3 or later. You can modify or redistribute it under the conditions of these licenses.

Fig (2-8) SQLite Database browser

Features of SQLite3

Transactions are atomic, consistent, isolated, and durable (ACID) even after system crashes and power failures.

Zero-configuration - no setup or administration needed.

Full SQL implementation with advanced features like partial indexes and common table expressions.

A complete database is stored in a single cross-platform disk file. Great for use as an application file format.

Supports terabyte-sized databases and gigabyte-sized strings and blobs.

Small code footprint: less than 500KiB fully configured or much less with optional features omitted.

Simple, easy to use API and written in ANSI-C. TCL bindings included. Bindings for dozens of other languages available separately.

Well-commented source code with 100% branch test coverage.

Available as a single ANSI-C source-code file that is easy to compile and hence is easy to add into a larger project.

Self-contained: no external dependencies. Cross-platform: Android, *BSD, iOS, Linux, Mac, Solaris, VxWorks, and Windows (Win32, WinCE, WinRT) are supported out of the box. Easy to port to other systems.

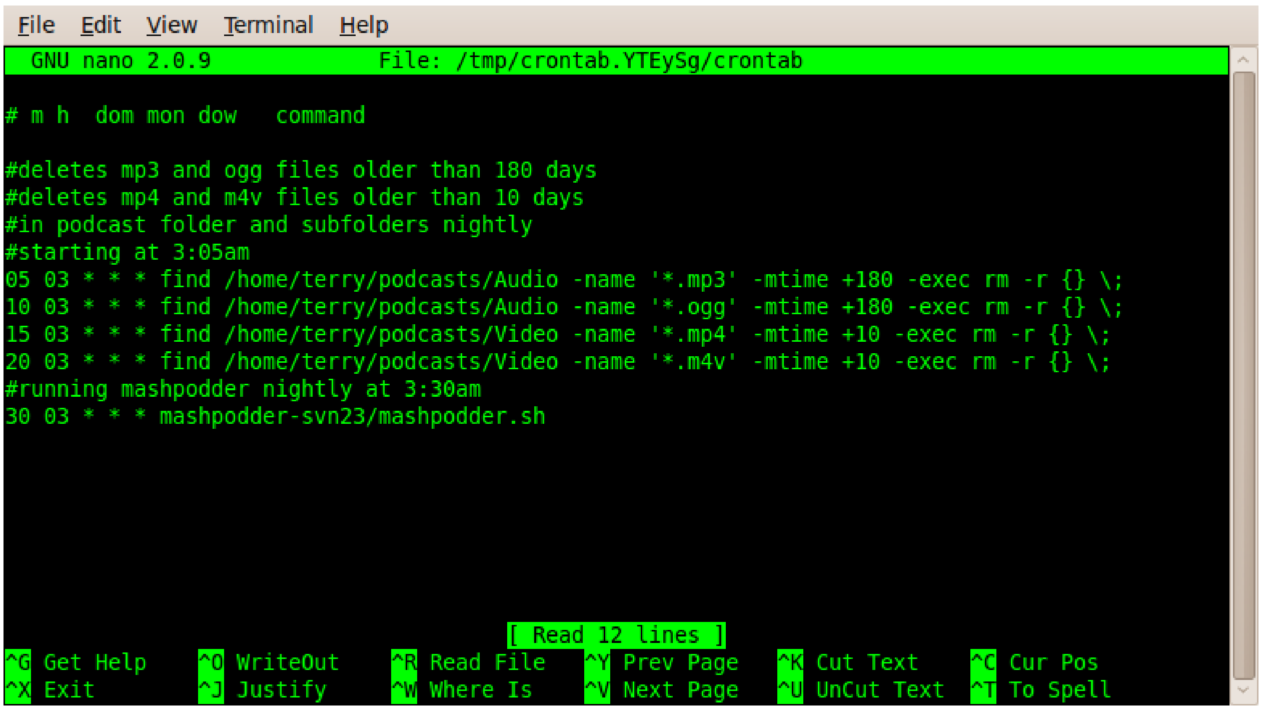
Sources are in the public domain. Use for any purpose. Comes with a standalone command-line interface (CLI) client that can be used to administer SQLite databases.

2.7 CRON Job

A CRON Job is a script that web servers execute at any given time. It is a time-based scheduler for Linux-based operating systems. If your web server uses CRON jobs then you are in luck. It’s because CRON Jobs can prevent unnecessary file changes in your system, prevents breakage to your website should you happen to install a faulty plugin, or even back up your entire WordPress blog even when you are sleeping.

CRON is a system daemon used to execute desired tasks (in the background) at designated times. A CRON tab is a simple text file with a list of commands meant to be run at specified time. These commands and their run times are then controlled by CRON daemon, which executes them in the system background.

Each user has a CRON tab file which specifies actions and times at which they should be executed, these jobs will run regardless of whether user is actually logged into the system or not. There is also a root CRON tab for tasks requiring administrative privileges. This system CRON tab allows scheduling of system-wide tasks such as log rotations and system database updates.



```
  GNU nano 2.0.9            File: /tmp/crontab.YTEySg/crontab

# m h  dom mon dow   command

#deletes mp3 and ogg files older than 180 days
#deletes mp4 and m4v files older than 10 days
#in podcast folder and subfolders nightly
#starting at 3:05am
05 03 * * * find /home/terry/podcasts/Audio -name '*.mp3' -mtime +180 -exec rm -r {} \;
10 03 * * * find /home/terry/podcasts/Audio -name '*.ogg' -mtime +180 -exec rm -r {} \;
15 03 * * * find /home/terry/podcasts/Video -name '*.mp4' -mtime +10 -exec rm -r {} \;
20 03 * * * find /home/terry/podcasts/Video -name '*.m4v' -mtime +10 -exec rm -r {} \;
#running mashpodder nightly at 3:30am
30 03 * * * mashpodder-svn23/mashpodder.sh







                                       [ Read 12 lines ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Fig (2-9) CRON file

Usually we intend to handle CRON daemon in a controlled way. One use case is when we just want to supply a command and set a CRON Job without editing file manually. A python library python-crontab provides a simple and effective way to access a CRON tab from python utils, allowing programmer to load CRON jobs as objects, search for them and save manipulations.

2.8 PyFPDF

PyFPDF is a library for PDF document generation under Python, ported from PHP. Compared with other PDF libraries, PyFPDF is simple, small and versatile, with advanced capabilities and easy to learn, extend and maintain.

Features of PyFPDF

Python 2.5 to 2.7 support (with experimental Python3 support)

Unicode (UTF-8) TrueType font subset embedding

Barcode I2of5 and code39, QR code support

PNG, GIF and JPG support (including transparency and alpha channel)

Templates with a visual designer & basic html2pdf

Exceptions support, other minor fixes, improvements and PEP8 code cleanups

Chapter-3

SYSTEM DEVELOPMENT

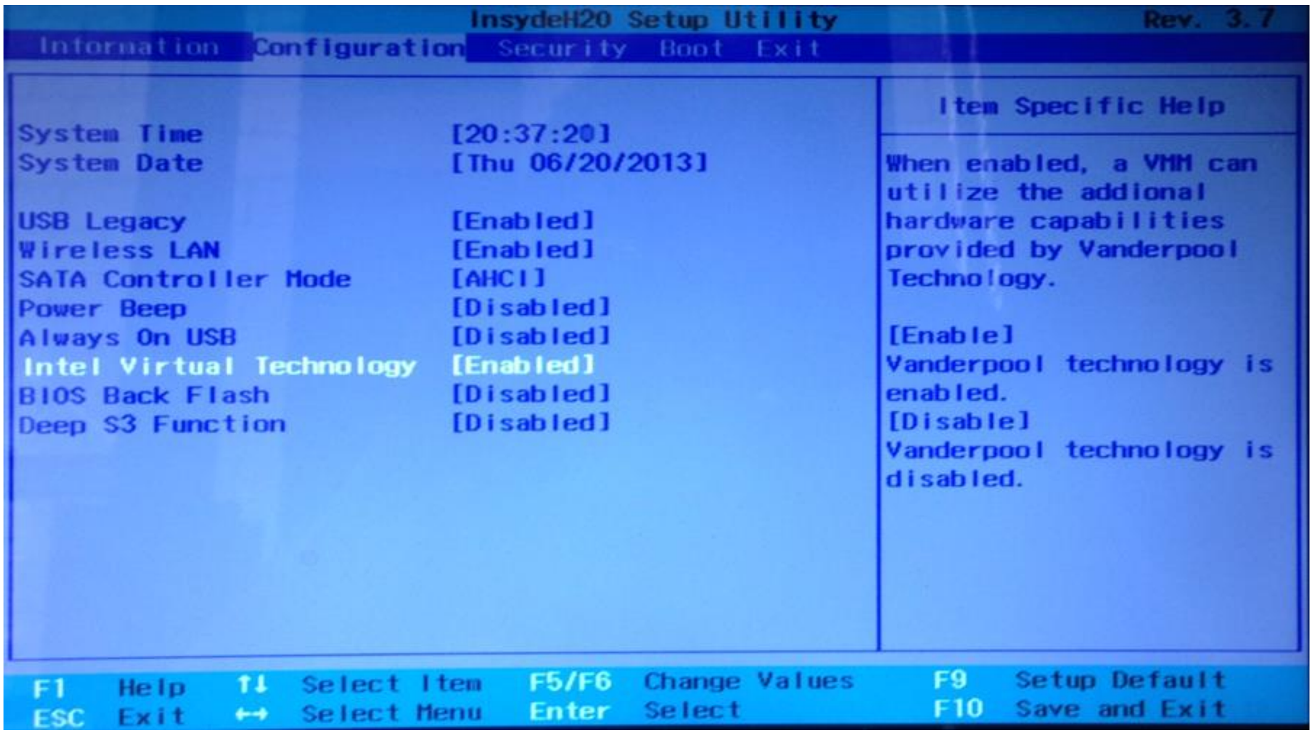
3.1 Putting together the development hardware

To emulate the cloud infrastructure as closely as possible, we will need all the hardware technologies that are commonly found in server-grade machines. E.g. Intel VT-x/AMD-V, Intel VT-d etc. Most of the high-end laptops and desktop machines include fairly the same set of hardware features.

The host system has the following specs:

- Intel Core i5 4th Generation 4210U 1.7 GHz Dual-Core Processor (includes SSE4.1/4.2, AVX 2.0, VT-x with EPT (Extended Page Tables), VT-d, vPro)
- 8GB DDR3 Dual-Channel RAM
- 500 GB Disk Space
- Nvidia GT840M GPU

The virtualization technologies are usually disabled by default in consumer-grade personal computers, therefore the switch needs to be toggled in the BIOS setup.



```
System Time                    [20:37:20]
System Date                    [Thu 06/20/2013]

USB Legacy                     [Enabled]
Wireless LAN                   [Enabled]
SATA Controller Mode           [AHCI]
Power Beep                     [Disabled]
Always On USB                  [Disabled]
Intel Virtual Technology       [Enabled]
BIOS Back Flash                [Disabled]
Deep S3 Function               [Disabled]
```

```
Item Specific Help

When enabled, a VMM can
utilize the addional
hardware capabilities
provided by Vanderpool
Technology.

[Enable]
Vanderpool technology is
enabled.
[Disable]
Vanderpool technology is
disabled.
```

Fig (3-1) Toggling the VT-x switch in BIOS setup of the machine

3.2 Setting up the Software Environment

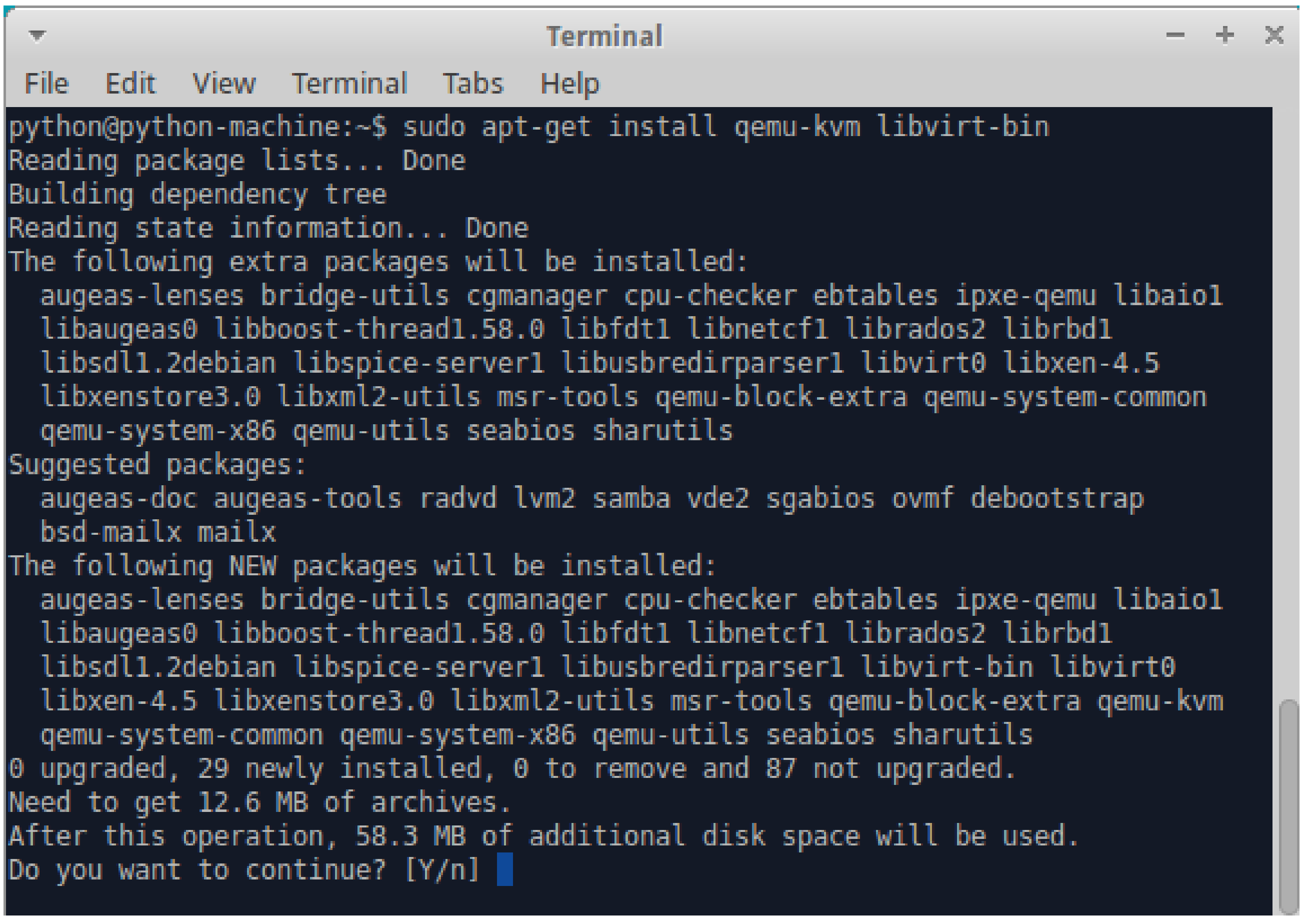
The development setup is on Linux, so a space must be allocated for linux operating system on the secondary memory. Ubuntu 15.10 is used to develop and implement this project. Every software or library required for the project is installed on this operating system.

3.3 Installing Libvirt on Ubuntu

Libvirt is a package available in the standard distribution packages repository of Ubuntu which can be trivially setup on the host system using a simple terminal command. A hypervisor must also be installed along with libvirt such as QEMU or VirtualBox.

sudo apt-get install qemu-kvm

sudo apt-get install libvirt-bin



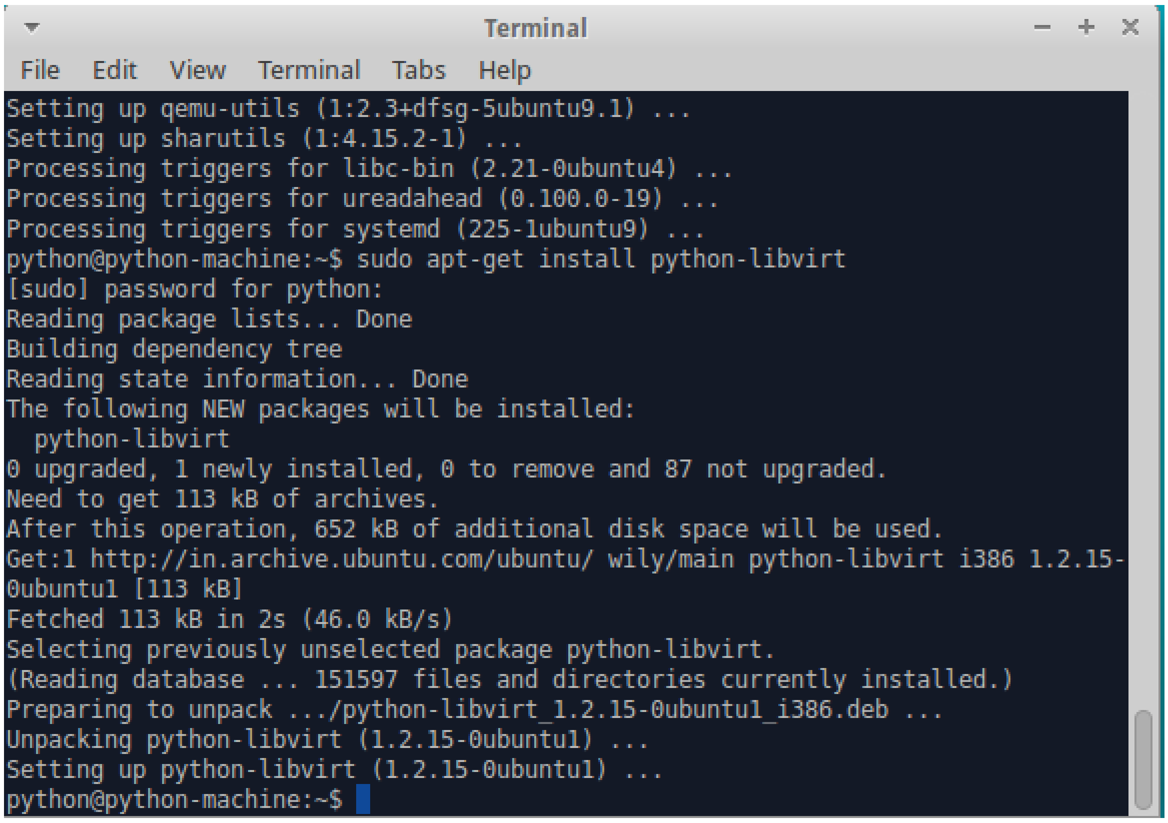
```
python@python-machine:~$ sudo apt-get install qemu-kvm libvirt-bin
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  augeas-lenses bridge-utils cgmanager cpu-checker ebtables ipxe-qemu libaio1
  libaugeas0 libboost-thread1.58.0 libfdt1 libnetcf1 librados2 librbd1
  libsdl1.2debian libspice-server1 libusbredirparser1 libvirt0 libxen-4.5
  libxenstore3.0 libxml2-utils msr-tools qemu-block-extra qemu-system-common
  qemu-system-x86 qemu-utils seabios sharutils
Suggested packages:
  augeas-doc augeas-tools radvd lvm2 samba vde2 sgabios ovmf debootstrap
  bsd-mailx mailx
The following NEW packages will be installed:
  augeas-lenses bridge-utils cgmanager cpu-checker ebtables ipxe-qemu libaio1
  libaugeas0 libboost-thread1.58.0 libfdt1 libnetcf1 librados2 librbd1
  libsdl1.2debian libspice-server1 libusbredirparser1 libvirt-bin libvirt0
  libxen-4.5 libxenstore3.0 libxml2-utils msr-tools qemu-block-extra qemu-kvm
  qemu-system-common qemu-system-x86 qemu-utils seabios sharutils
0 upgraded, 29 newly installed, 0 to remove and 87 not upgraded.
Need to get 12.6 MB of archives.
After this operation, 58.3 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Fig (3-2) Libvirt package for Ubuntu

3.4 Installing Python3 and libvirt python packages using pip

The development system is based on python3 language, therefore all the packages required must be installed for Python 3 language instead of the system-level Python 2 installation. To interface with libvirt using python language, the specific libvirt-python package must be installed in the system for Python 3.

sudo apt-get install python-libvirt



```
Setting up qemu-utils (1:2.3+dfsg-5ubuntu9.1) ...
Setting up sharutils (1:4.15.2-1) ...
Processing triggers for libc-bin (2.21-0ubuntu4) ...
Processing triggers for ureadahead (0.100.0-19) ...
Processing triggers for systemd (225-1ubuntu9) ...
python@python-machine:~$ sudo apt-get install python-libvirt
[sudo] password for python:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  python-libvirt
0 upgraded, 1 newly installed, 0 to remove and 87 not upgraded.
Need to get 113 kB of archives.
After this operation, 652 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu/ wily/main python-libvirt i386 1.2.15-
0ubuntu1 [113 kB]
Fetched 113 kB in 2s (46.0 kB/s)
Selecting previously unselected package python-libvirt.
(Reading database ... 151597 files and directories currently installed.)
Preparing to unpack .../python-libvirt_1.2.15-0ubuntu1_i386.deb ...
Unpacking python-libvirt (1.2.15-0ubuntu1) ...
Setting up python-libvirt (1.2.15-0ubuntu1) ...
python@python-machine:~$
```

Fig (3-3) Python bindings for libvirt

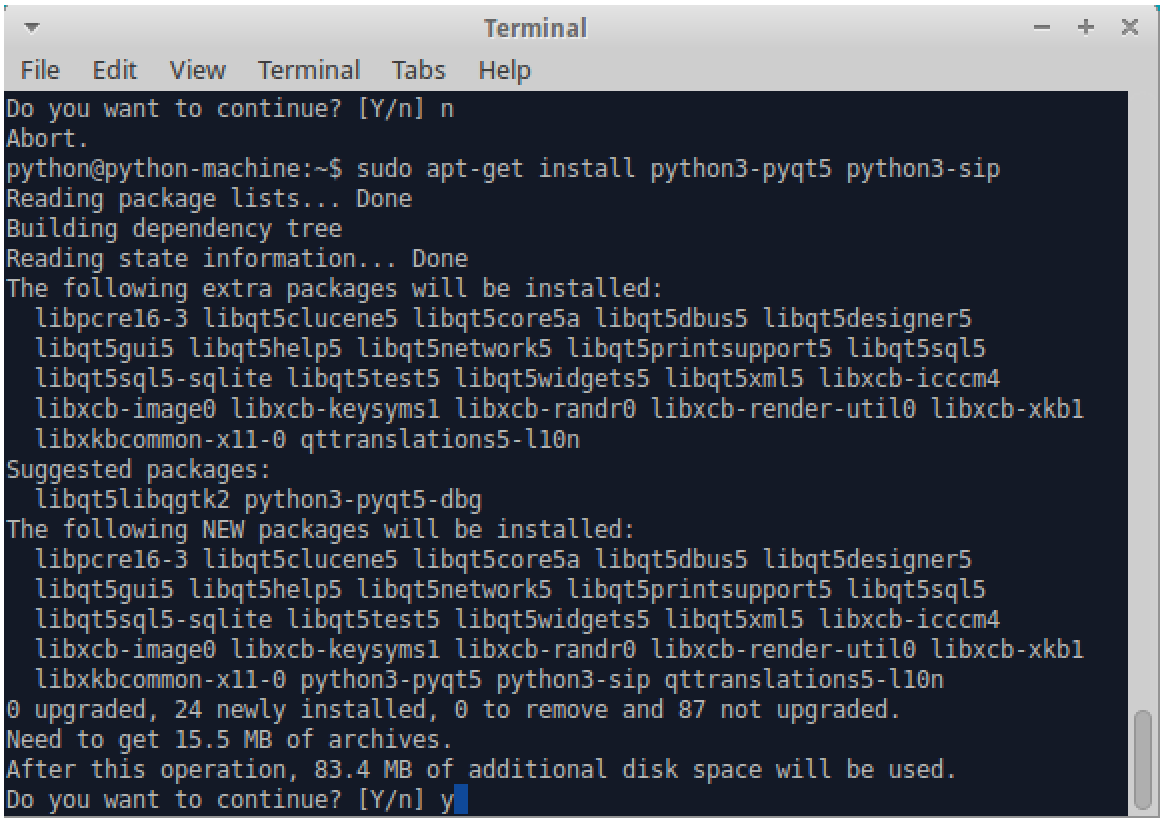
3.5 Install PyQt5

The Qt app framework's runtime is already included in the default Ubuntu installation, but the development libraries of Qt (version 5) and PyQt bindings for the framework needs to be installed separately along with the dependencies such as SIP. The installation of PyQt framework also requires presence of a compilation toolchain for the C language such as Clang or GCC.

sudo apt-get install build-essentials

sudo apt-get install sip

sudo apt-get install pyqt5



```
Do you want to continue? [Y/n] n
Abort.
python@python-machine:~$ sudo apt-get install python3-pyqt5 python3-sip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libpcre16-3 libqt5clucene5 libqt5core5a libqt5dbus5 libqt5designer5
  libqt5gui5 libqt5help5 libqt5network5 libqt5printsupport5 libqt5sql5
  libqt5sql5-sqlite libqt5test5 libqt5widgets5 libqt5xml5 libxcb-icccm4
  libxcb-image0 libxcb-keysyms1 libxcb-randr0 libxcb-render-util0 libxcb-xkb1
  libxkbcommon-x11-0 qttranslations5-l10n
Suggested packages:
  libqt5libqgtk2 python3-pyqt5-dbg
The following NEW packages will be installed:
  libpcre16-3 libqt5clucene5 libqt5core5a libqt5dbus5 libqt5designer5
  libqt5gui5 libqt5help5 libqt5network5 libqt5printsupport5 libqt5sql5
  libqt5sql5-sqlite libqt5test5 libqt5widgets5 libqt5xml5 libxcb-icccm4
  libxcb-image0 libxcb-keysyms1 libxcb-randr0 libxcb-render-util0 libxcb-xkb1
  libxkbcommon-x11-0 python3-pyqt5 python3-sip qttranslations5-l10n
0 upgraded, 24 newly installed, 0 to remove and 87 not upgraded.
Need to get 15.5 MB of archives.
After this operation, 83.4 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Fig (3-4) PyQt5 and SIP

3.6 Install matplotlib

To display graphs in our PyQt-based desktop application, the matplotlib Python 3 library is required to be installed.

python3-pip install matplotlib

3.7 Install PyFPDF

To create and edit pdf documents using python. This library is used to generate daily and weekly reports.

sudo pip3 install fpdf

Chapter-4

EXPERIMENTS AND RESULTS

Our hypothesis was to design and develop an Infrastructure-as-a-Service Cloud monitor to monitor all resources used by the virtual machine and hence the host machine which contains all VMs.
In the first milestone, we have monitored the CPU Utilisation, MIPS, Memory Utilisation, Disk read and write and temperature by the host machine for running the VMs in the host machine.
For the second milestone, for each VM in the host machine we designed and developed an application which monitors CPU Utilisation, Memory Utilisation, Memory Usage, Disk read and write and network read and write.
In the third milestone we implemented a background service which writes all the values to be monitored into a sqlite3 database, which is used for generating daily and weekly reports and also the power utilised (by using the cloud sim linear power model) by a host machine for running the VMs in the host machine.

The snapshots of the running program are as follows:

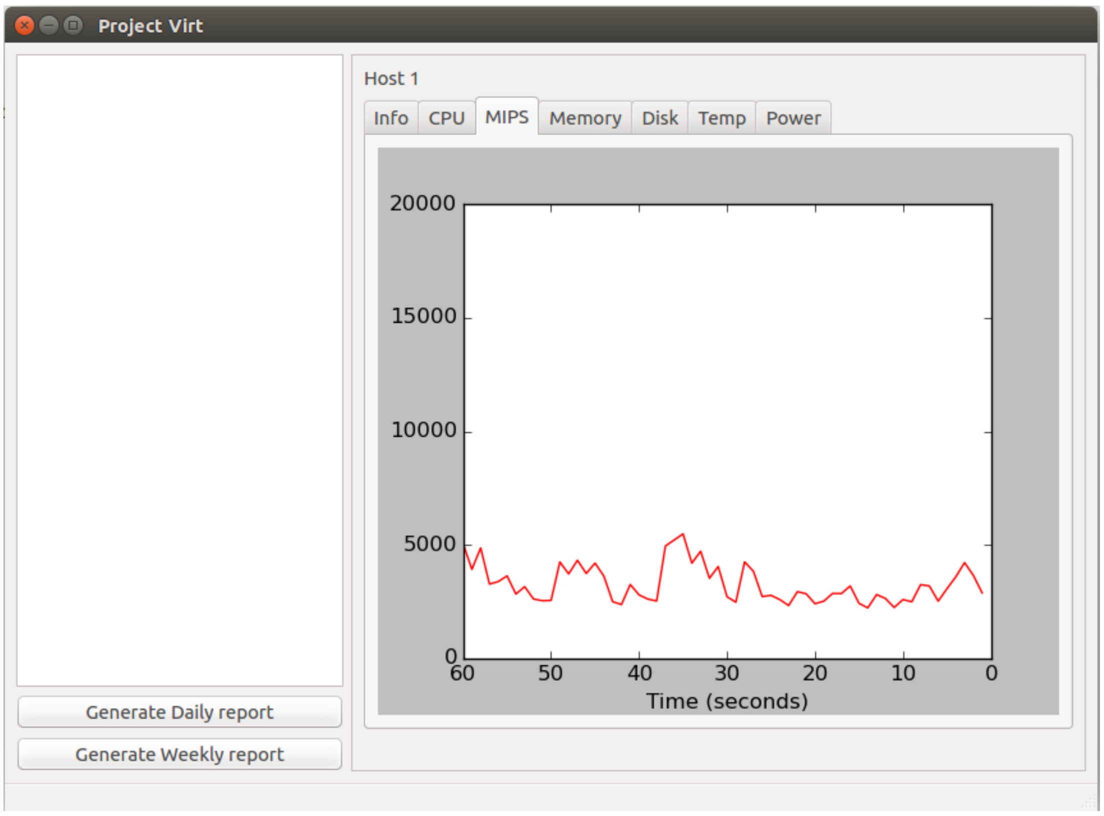
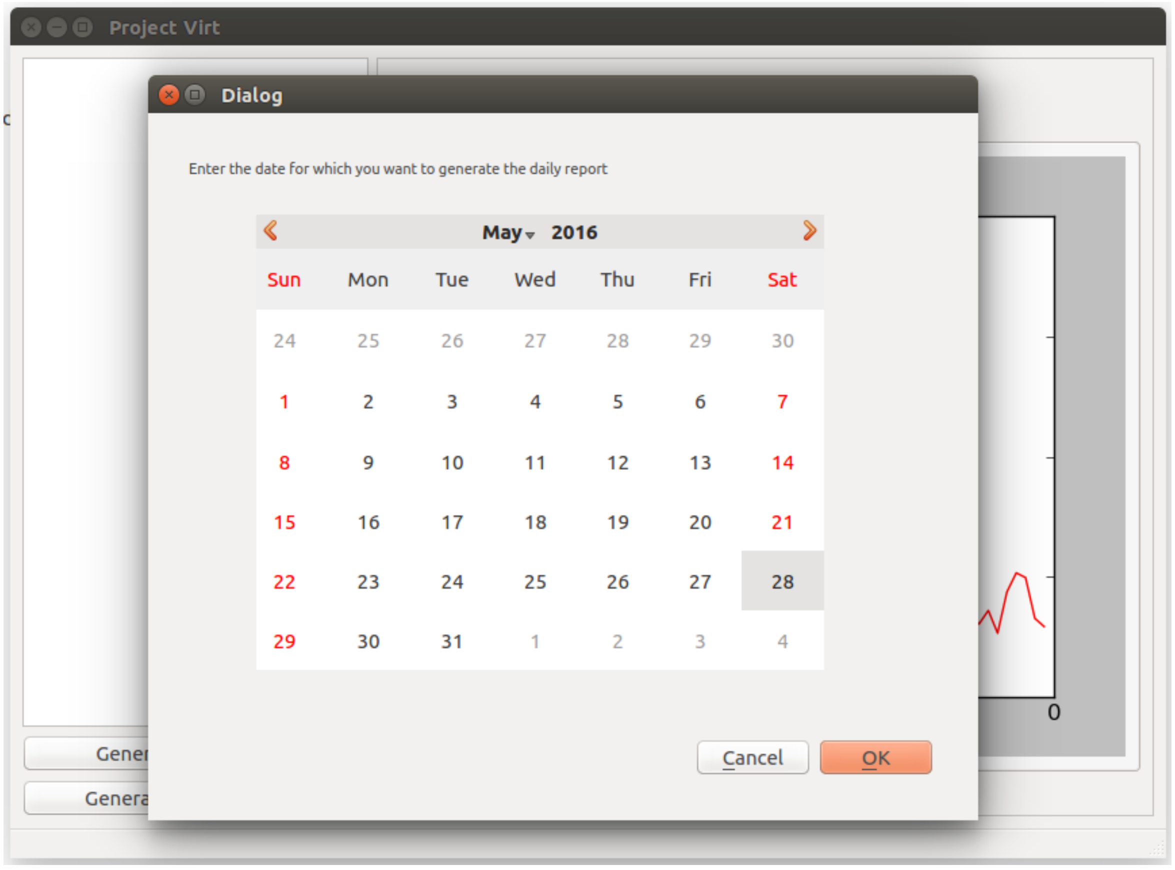


Fig (4-1) Host window application

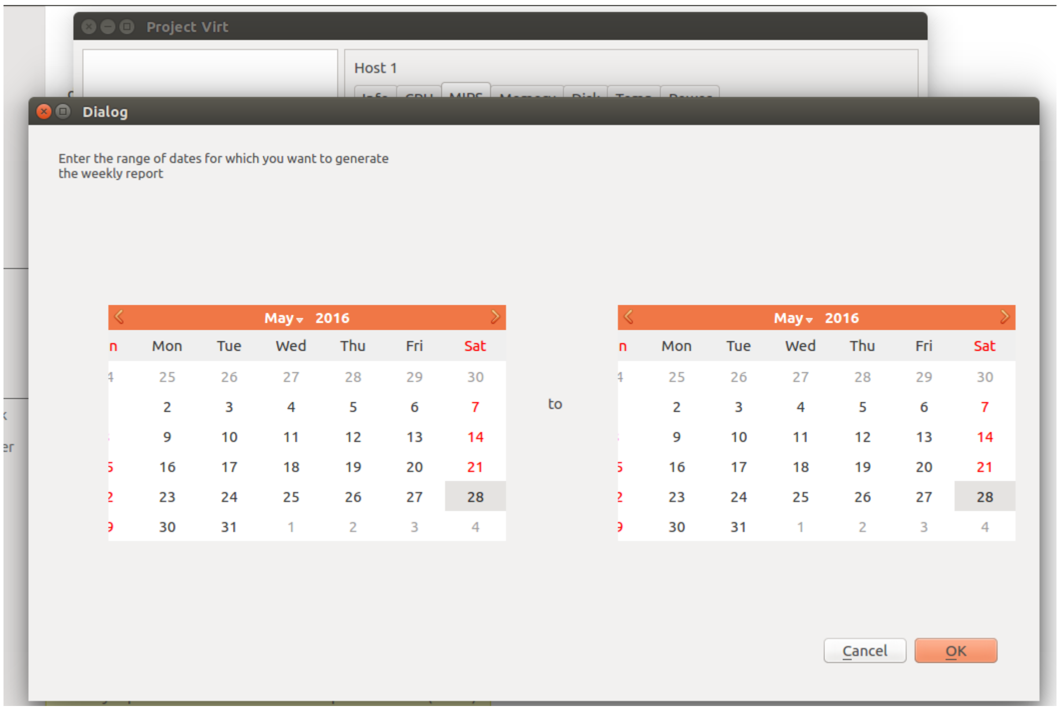


Enter the date for which you want to generate the daily report

May 2016

Sun	Mon	Tue	Wed	Thu	Fri	Sat
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Fig (4-2) Daily report generation



Enter the range of dates for which you want to generate
the weekly report

May 2016

n	Mon	Tue	Wed	Thu	Fri	Sat
4	25	26	27	28	29	30
	2	3	4	5	6	7
	9	10	11	12	13	14
5	16	17	18	19	20	21
2	23	24	25	26	27	28
9	30	31	1	2	3	4

to

May 2016

n	Mon	Tue	Wed	Thu	Fri	Sat
4	25	26	27	28	29	30
	2	3	4	5	6	7
	9	10	11	12	13	14
5	16	17	18	19	20	21
2	23	24	25	26	27	28
9	30	31	1	2	3	4

Fig (4-3) Weekly report generation

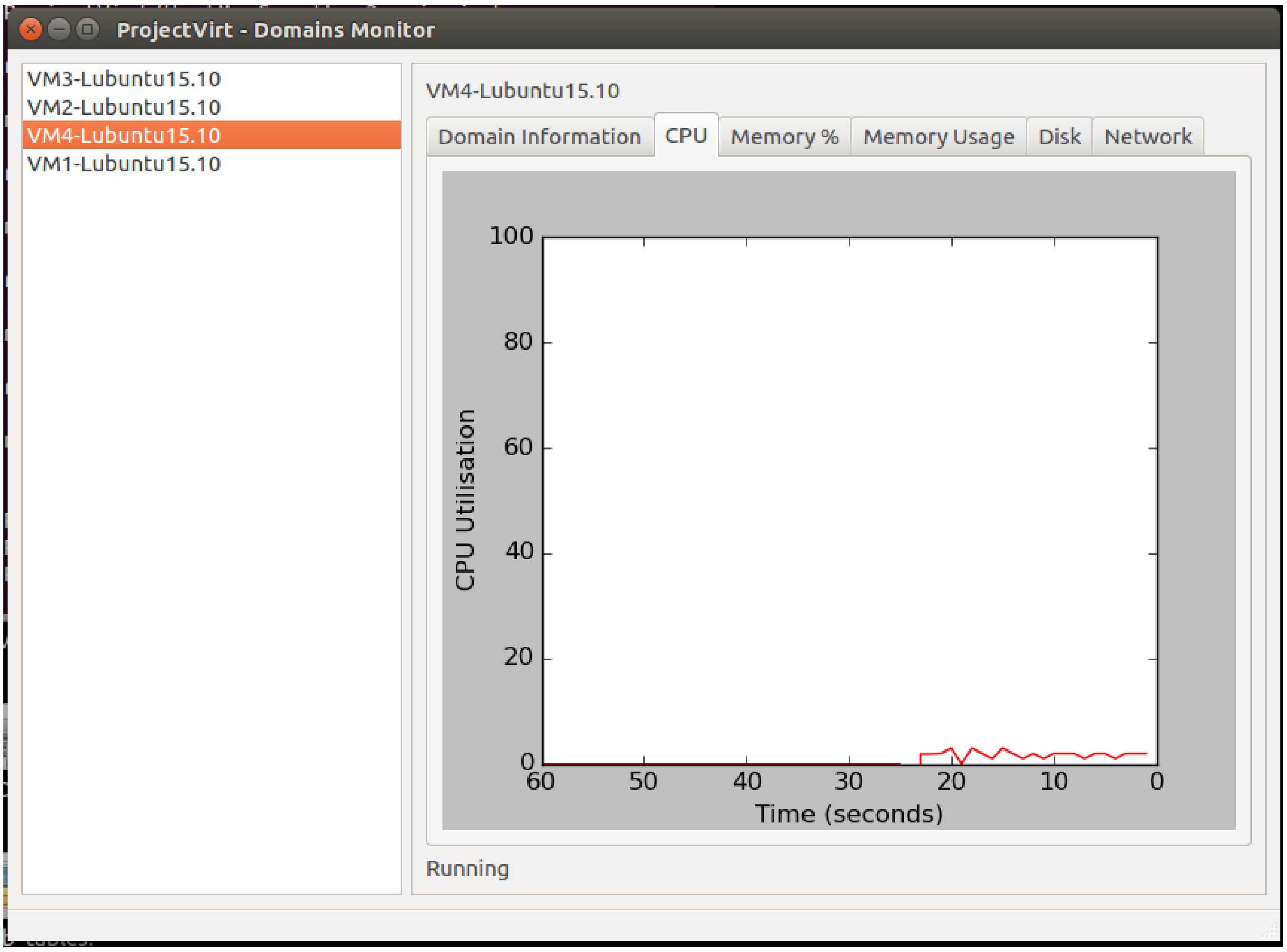


Fig (4-4) Domain monitor application

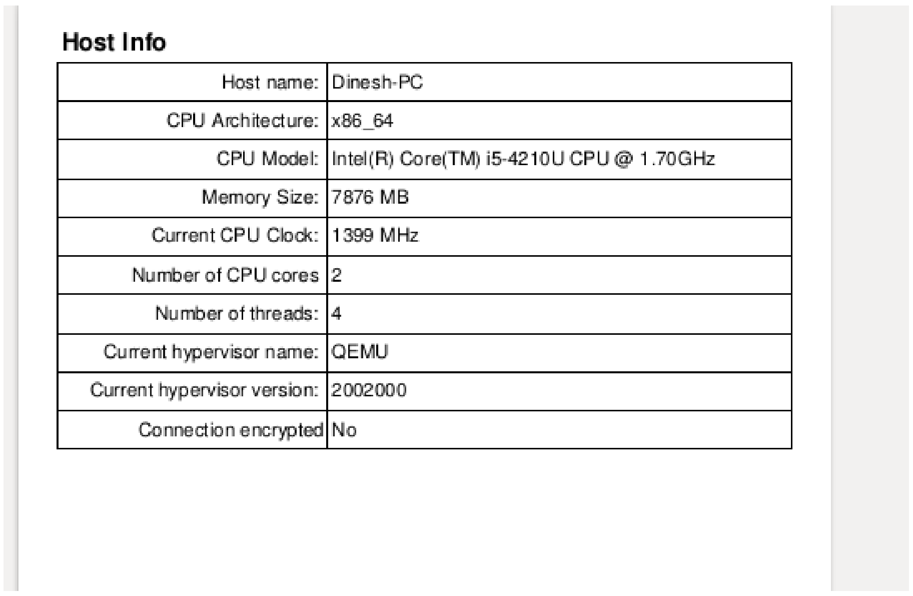
The snapshots of daily and weekly reports are as follows:

Daily report

Generated at 28-05-2016 23:15:30

Generated for 2016-05-28

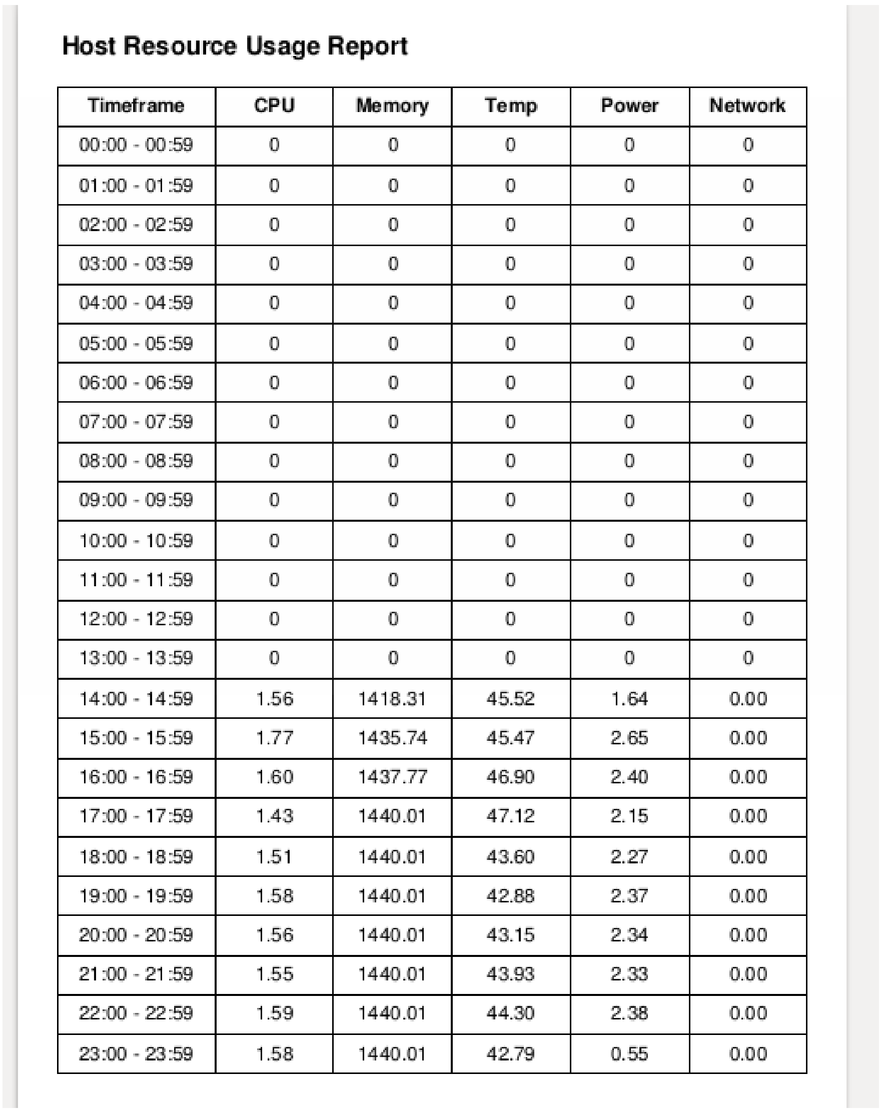
Fig (4-5) First page of daily report



Host Info

Host name:	Dinesh-PC
CPU Architecture:	x86_64
CPU Model:	Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz
Memory Size:	7876 MB
Current CPU Clock:	1399 MHz
Number of CPU cores	2
Number of threads:	4
Current hypervisor name:	QEMU
Current hypervisor version:	2002000
Connection encrypted	No

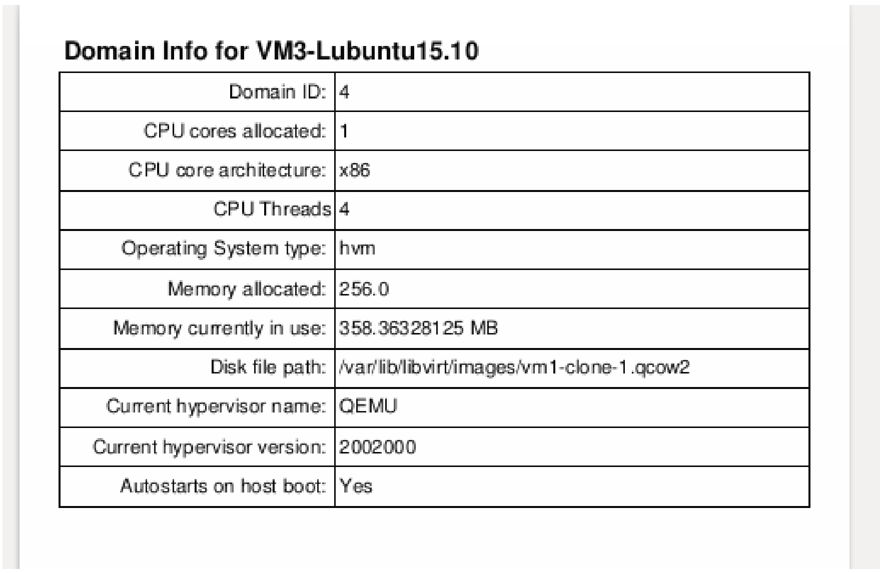
Fig (4-6) Host information in daily report



Host Resource Usage Report

Timeframe	CPU	Memory	Temp	Power	Network
00:00 - 00:59	0	0	0	0	0
01:00 - 01:59	0	0	0	0	0
02:00 - 02:59	0	0	0	0	0
03:00 - 03:59	0	0	0	0	0
04:00 - 04:59	0	0	0	0	0
05:00 - 05:59	0	0	0	0	0
06:00 - 06:59	0	0	0	0	0
07:00 - 07:59	0	0	0	0	0
08:00 - 08:59	0	0	0	0	0
09:00 - 09:59	0	0	0	0	0
10:00 - 10:59	0	0	0	0	0
11:00 - 11:59	0	0	0	0	0
12:00 - 12:59	0	0	0	0	0
13:00 - 13:59	0	0	0	0	0
14:00 - 14:59	1.56	1418.31	45.52	1.64	0.00
15:00 - 15:59	1.77	1435.74	45.47	2.65	0.00
16:00 - 16:59	1.60	1437.77	46.90	2.40	0.00
17:00 - 17:59	1.43	1440.01	47.12	2.15	0.00
18:00 - 18:59	1.51	1440.01	43.60	2.27	0.00
19:00 - 19:59	1.58	1440.01	42.88	2.37	0.00
20:00 - 20:59	1.56	1440.01	43.15	2.34	0.00
21:00 - 21:59	1.55	1440.01	43.93	2.33	0.00
22:00 - 22:59	1.59	1440.01	44.30	2.38	0.00
23:00 - 23:59	1.58	1440.01	42.79	0.55	0.00

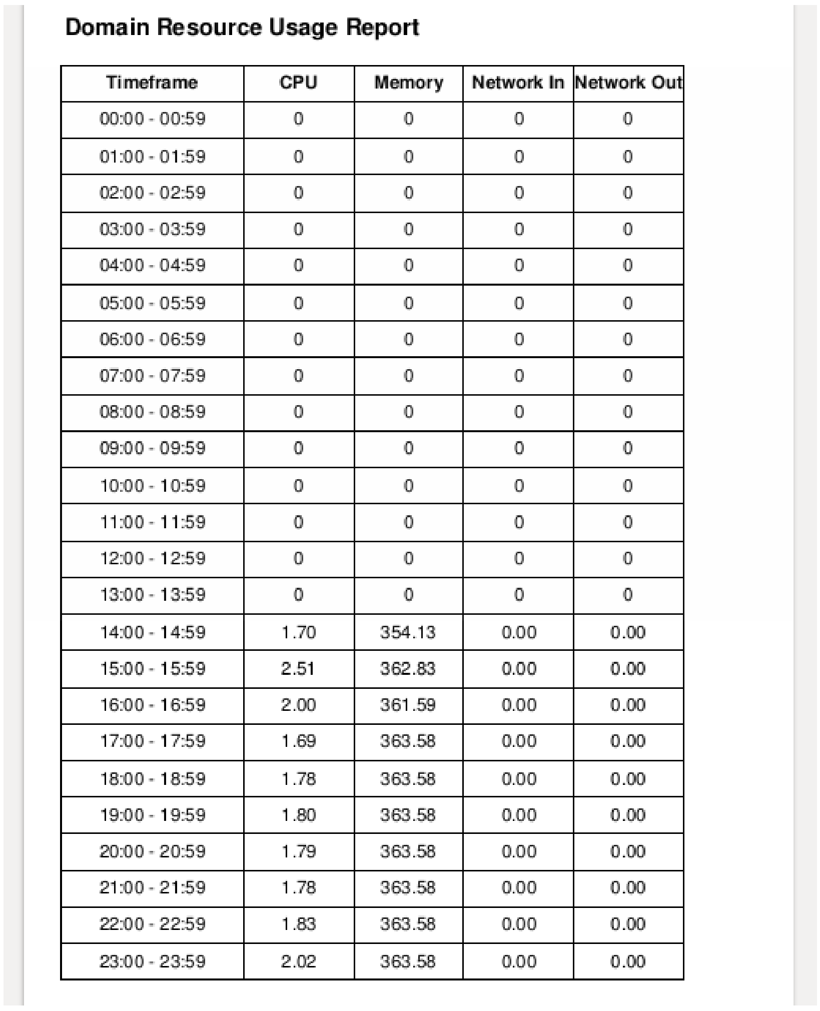
Fig (4-7) Host resource usage report



Domain Info for VM3-Lubuntu15.10

Domain ID:	4
CPU cores allocated:	1
CPU core architecture:	x86
CPU Threads	4
Operating System type:	hvm
Memory allocated:	256.0
Memory currently in use:	358.36328125 MB
Disk file path:	/var/lib/libvirt/images/vm1-clone-1.qcow2
Current hypervisor name:	QEMU
Current hypervisor version:	2002000
Autostarts on host boot:	Yes

Fig (4-8) Domain information for a VM in daily report



Domain Resource Usage Report

Timeframe	CPU	Memory	Network In	Network Out
00:00 - 00:59	0	0	0	0
01:00 - 01:59	0	0	0	0
02:00 - 02:59	0	0	0	0
03:00 - 03:59	0	0	0	0
04:00 - 04:59	0	0	0	0
05:00 - 05:59	0	0	0	0
06:00 - 06:59	0	0	0	0
07:00 - 07:59	0	0	0	0
08:00 - 08:59	0	0	0	0
09:00 - 09:59	0	0	0	0
10:00 - 10:59	0	0	0	0
11:00 - 11:59	0	0	0	0
12:00 - 12:59	0	0	0	0
13:00 - 13:59	0	0	0	0
14:00 - 14:59	1.70	354.13	0.00	0.00
15:00 - 15:59	2.51	362.83	0.00	0.00
16:00 - 16:59	2.00	361.59	0.00	0.00
17:00 - 17:59	1.69	363.58	0.00	0.00
18:00 - 18:59	1.78	363.58	0.00	0.00
19:00 - 19:59	1.80	363.58	0.00	0.00
20:00 - 20:59	1.79	363.58	0.00	0.00
21:00 - 21:59	1.78	363.58	0.00	0.00
22:00 - 22:59	1.83	363.58	0.00	0.00
23:00 - 23:59	2.02	363.58	0.00	0.00

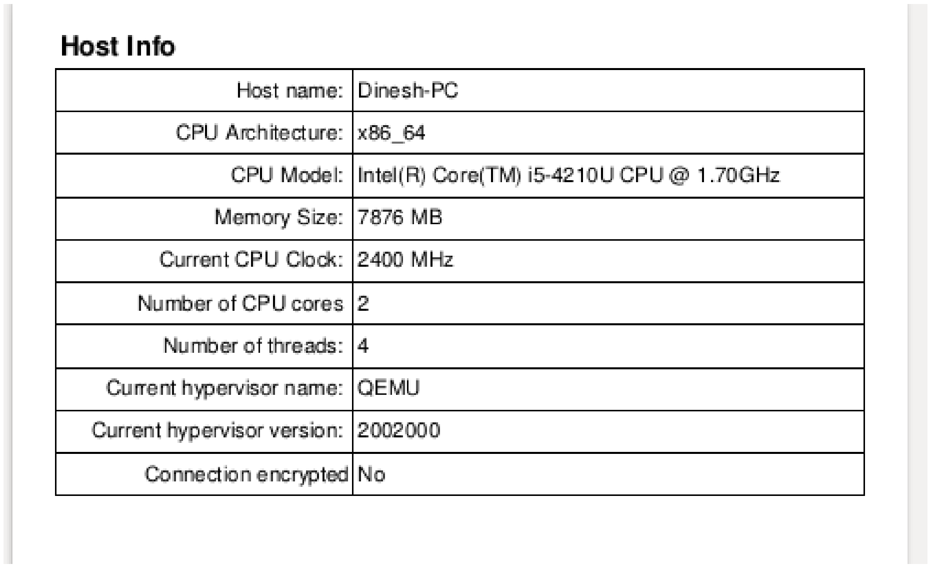
Fig (4-9) Domain resource usage report

Weekly report

Generated at 28-05-2016 23:15:42

Generated for 2016-05-27 to 2016-05-28

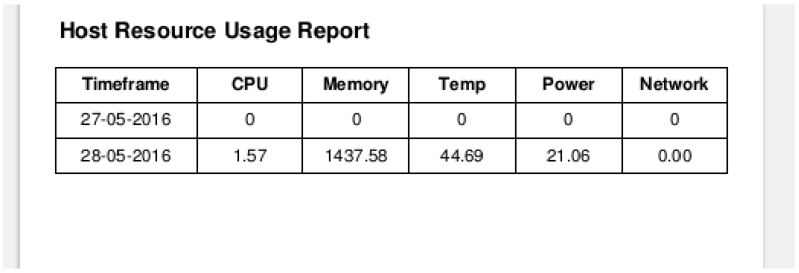
Fig (4-10) First page of weekly report



Host Info

Host name:	Dinesh-PC
CPU Architecture:	x86_64
CPU Model:	Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz
Memory Size:	7876 MB
Current CPU Clock:	2400 MHz
Number of CPU cores	2
Number of threads:	4
Current hypervisor name:	QEMU
Current hypervisor version:	2002000
Connection encrypted	No

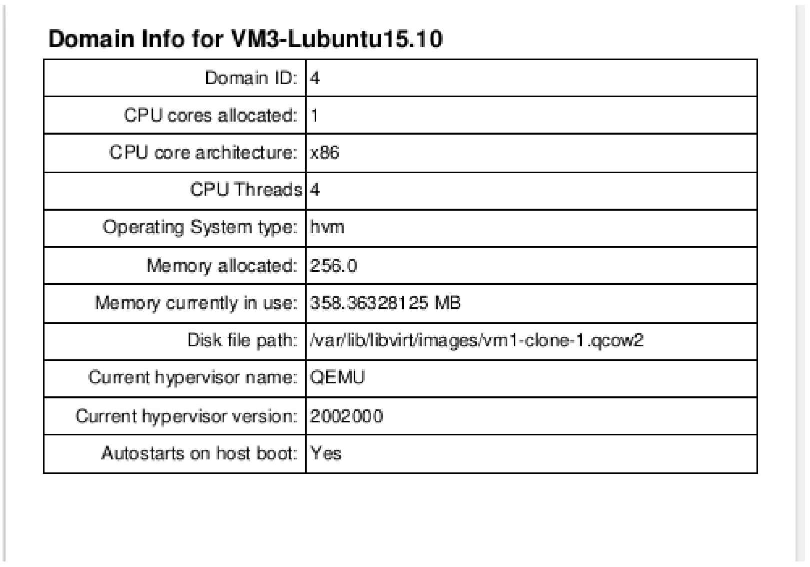
Fig (4-11) Host information in weekly report



Host Resource Usage Report

Timeframe	CPU	Memory	Temp	Power	Network
27-05-2016	0	0	0	0	0
28-05-2016	1.57	1437.58	44.69	21.06	0.00

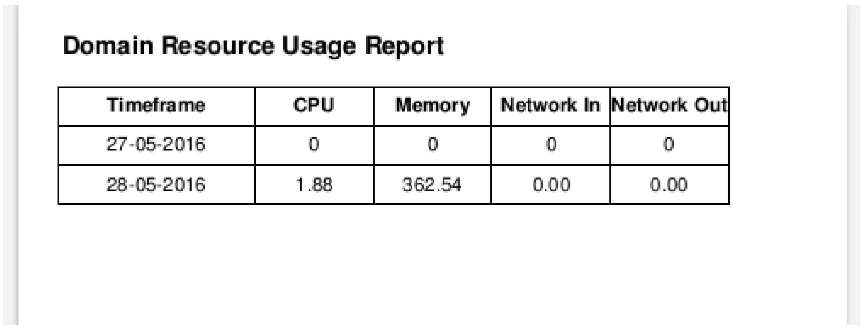
Fig (4-12) Host resource usage report



Domain Info for VM3-Lubuntu15.10

Domain ID:	4
CPU cores allocated:	1
CPU core architecture:	x86
CPU Threads	4
Operating System type:	hvm
Memory allocated:	256.0
Memory currently in use:	358.36328125 MB
Disk file path:	/var/lib/libvirt/images/vm1-clone-1.qcow2
Current hypervisor name:	QEMU
Current hypervisor version:	2002000
Autostarts on host boot:	Yes

Fig (4-13) Domain information in weekly report



Domain Resource Usage Report

Timeframe	CPU	Memory	Network In	Network Out
27-05-2016	0	0	0	0
28-05-2016	1.88	362.54	0.00	0.00

Fig (4-14) Domain resource usage report

Chapter-5

CONCLUSION

Designed and developed an IaaS Cloud Monitor Application which monitors all the resources used by the Host and VM's in the host for the data centre manager to keep track of all the resources used by the host and also the VM's. There are two parts in this project. The first one is Host monitor and the domain monitor. Second one is the power utilisation and the daily and weekly report generators. We made GUI using PyQt5, plotted graphs using matplotlib python library and implemented the inputs (CPU usage, memory usage, disk usage and temperature) for the power model. Used libvirt to find CPU utilisation, memory, disk usage and hence power utilisation using linear power model.

Our hypothesis was to design and develop a IaaS Cloud monitor to monitor all resources used by the virtual machine and hence the host machine which contains all VMs. we have monitored the CPU Utilisation, MIPS, Memory Utilisation, Disk read and write and temperature by the host machine for running the VMs in the host machine. For each VM in the host machine we designed and developed an application which monitors CPU Utilisation, Memory Utilisation, Memory Usage, Disk read and write and network read and write. we implemented a background service which writes all the values to be monitored into a sqlite3 database, which is used for generating daily and weekly reports and also the power utilised (by using the cloud sim linear power model) by a host machine for running the VMs in the host machine.

An extension to this project would be to develop an application to monitor all the host PCs in the data centre. This can be done by using secure shell to make the communication between the host PCs.

REFERENCES

1. P. Mell, T. Grance, “The NIST Definition of Cloud Computing”, NIST Special Publication 800-145, 2011.
2. Badger, Grance, Patt-Corner and Voas, “Cloud Computing Synopsis and Recommendations”, NIST Special Publication 800-146, 2012.
3. Aniruddha S. Rumale, D.N. Chaudhari, “Cloud Computing: Infrastructure as a Service”, International Journal of Inventive Engineering and Sciences 1(3), 2319–9598, 2013.
4. Giuseppe Aceto, Alessio Botta, Walter de Donato and Antonio Pescapè, “Cloud Monitoring: A survey,” Computer Networks 57(9), Elsevier, 2093–2115, 2013.
5. Giuseppe Aceto, Alessio Botta, Walter de Donato, Antonio Pescapè, “Cloud Monitoring: definitions, issues and future directions,” 2012.
6. Oracle Corporation “Making Infrastructure-as-a-Service in the Enterprise a Reality”, Copyrights Oracle Corporation and/or its affliates, 2012, http://www.oracle.com/us/products/enterprise-manager/infrastructure-as-a-service-wp-1575856.
7. Libvirt Organisation, “http://libvirt.org/docs.html”.

APPENDICES

Source Code

./BackgroundSvc/bgservice.py

This python3 script runs as background service and logs all resources usage and vm details at an interval in an optimized manner.

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/BackgroundSvc/bgservice.py

./Background/create_db_tables.py

Creates empty tables in the sqlite3 database.

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/BackgroundSvc/create_db_tables.py

./DomainMon/domainsui.ui

UI Form XML file for the Domain monitor Qt5 desktop application.

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/DomainMon/domainui.ui

./DomainMon/ui_domainmon.py

Generated python class from the UI Form file using pyuic

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/DomainMon/ui_domainmon.py

./DomainMon/domwindow.py

Source code for the Domain Monitor PyQt5 application. It monitors the various parameters of the domains using libvirt and displays them for the selected running domain.

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/DomainMon/domwindow.py

./HostMon/dailygen.ui

UI Form XML file for the date picker to generate Daily report

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/HostMon/dailygen.ui

./HostMon/daily.py

Generated python class from the UI Form file using pyuic

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/HostMon/daily.py

./HostMon/mainui.ui

UI Form XML file for the Host monitor Qt5 desktop application.

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/HostMon/mainui.ui

./HostMon/ui_mainwindow.py

Generated python class from the UI Form file using pyuic

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/HostMon/ui_mainwindow.py

./HostMon/mainwindow.py

Source code for the Host Monitor PyQt5 application. It reads the various parameters of the host using system APIs, libvirt and background service's database and displays them using informative graphs.

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/HostMon/domwindow.py

./HostMon/weeklygen.ui

UI Form XML file for the date picker to generate Weekly report

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/HostMon/weeklygen.ui

./HostMon/weekly.py

Generated python class from the UI Form file using pyuic

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/HostMon/weekly.py

./ReportGenerator/reportgen.py

This python3 script generates weekly or daily reports for the dates passed as arguments. e.g.

```
$ python3 reportgen.py –weekly 2016-05-20 2016-05-27
```

Git repo path: https://github.com/abhinavk/Virtproj/blob/master/ReportGenerator/reportgen.py