Vrije Universiteit Amsterdam

Faculty of Sciences

Parallel and Distributed Computer Systems

Master Thesis

An Adaptive Cloud Computing Architecture for Streamlining Globally Distributed Applications

Author:
Ismail El-Helw

Student ID:
1824384

Supervisor:
Guillaume Pierre

Second Reader:
Thilo Kielmann

August 26, 2010

Contents

Chapter 1

Introduction

Cloud computing has gained massive success in the last decade. It provides unforeseen benefits to businesses and application developers [12, 27]. Using the cloud, small and medium sized organizations can scale their services without the need for building or maintaining large data centers. The success of the cloud computing model attracted many technology companies into providing their own infrastructure and platforms as services. Some data centers became publicly accessible to host applications and systems of customers while others remain private. This new technology ecosystem poses new challenges on data center management and resource provisioning. Namely, with the rising popularity of clouds, data centers will experience more dynamically changing demands for resources. Situations will arise in which more compute resources are required while the data center is already overloaded. Moreover, a data center may need to reclaim some of the resources it provided publicly in order to support rising demands of internal applications. In such settings, collaboration and resource sharing between different data centers will be needed.

Data centers that host cloud computing applications usually employ virtualization technologies to cater for their needs as it allows for efficient resource utilization and consolidation. For example, current data centers relocate virtual machines from one physical machine to another within a single data center's network to improve resource utilization. However, to our best knowledge, there were no attempts to relocate virtual machines across wide area networks. There is great potential for streamlining application performance if data centers apply consolidation techniques on a wider scale. For example, consider the varying HTTP request frequencies observed by a web server. Typically, high loads of client requests are processed during day hours compared to the minimal number of requests observed at night. A global web sever may decide to "follow the sun" and relocate from one data center to another such that it serves every geographical region in daytime. At any given time, the server will be in close proximity to daytime clients and serve them quickly.

The current state of cloud computing infrastructure that is offered by major service providers however does not encourage collaboration and resource sharing between different data centers. The services that they provide remain disjoint to a large extent. For example, consider Amazon's Elastic Compute Cloud (EC2) [2]. When spawning a new EC2 instance, a cloud user would need to specify the region that is going to host it. In this case, the region specifies the geographical location of the hosting data center. This characteristic remains constant for the lifetime of the newly created instance. To move the services provided by this instance to another EC2 region, a user needs to spawn a new instance in the target region and manually start the services in it before freeing the resources that were used by the first instance. The current cloud computing infrastructure does not provide an application transparent mechanism to perform such a task. Therefore, migration may create disruptions in network connections and affect applications that are not capable of handling it gracefully.

One important reason to relocate applications running over the cloud across different data centers is to optimize their internal communications. For example, if a group of processes participate in a leader election algorithm, it may be beneficial to place the leader process at a privileged location which has superior bandwidth available to attain higher performance. However, it is a very challenging problem to predict task delegations within a distributed system. Moreover, automated relocation in the cloud becomes more complicated if we assume that the platform does not have specific knowledge of the behavior and type of applications it is hosting. If the application in question is interactive and provides services over the Web, it may experience varying request intensities and flash crowds. The cloud infrastructure should therefore provide scalability features to cope with the fluctuating demands for resources. Web applications may also witness increasing popularity in certain areas of the world. In that case, migrating more server processes to a location in close proximity to the consumers is advantageous. Content Delivery Networks (CDN) are prime examples of how beneficial it is to place servers in strategic positions in the network. Web based systems that serve content to millions of clients usually employ CDNs to replicate their pages and serve them from locations in close proximity to the clients in a scalable fashion.

This thesis aims to streamline the utilization of data center resources by distributed applications. Paying attention to the networking patterns of a distributed application can increase its performance significantly without allocating to it more compute resources. In this work, we discuss the design and implementation of a cloud computing architecture that spans multiple data centers. The architecture is designed to support efficient wide-area migrations of virtual machines. To our best knowledge, this has never been done before as virtual machine mobility is traditionally confined within a single LAN. The system depends highly on IPv6 together with mobile IPv6 features to provide support for the mobility of virtual machines across data centers. Additionally, the architecture employs the Xen Hypervisor as a stable virtualization platform for compute servers. This combination of networking and virtualization technologies allow us to transparently combine the resources of multiple data centers. The following are the contributions of this work.

- We designed a cloud architecture that is capable of relocating virtual machines across multiple data centers over wide area networks.
- The system monitors the communication patterns of virtual machines and analyzes their messaging relationships.
- We developed a greedy algorithm that tackles the challenging problem of relocating the virtual machines to streamline the communication of the deployed distributed applications based on the gathered messaging information.
- We designed a unique file system structure for virtual machines to reduce the networking overhead incurred by accessing their disk images.
- We modified the Xen user space daemon to support virtual machine migration over IPv6 networks.

This thesis is structured as follows. In Chapter 2, we provide more context by discussing available technologies and related work. An overview of the architecture and our design decisions is discussed in Chapter 3. Chapter 4 discusses methods for gathering network statistics and provisioning network resources. Chapter 5 presents our findings from practical experiments and simulations. Finally, concluding remarks are given in Chapter 6.

Chapter 2

Background and Related Work

The design of a globally distributed cloud computing architecture depends on the integration of several types of technologies from different domains, namely, virtualization, resource provisioning and networking mobility. Building a cloud requires defining the relationships between these technologies and figuring out how they can be fused in a meaningful manner. This chapter discusses the technologies that distinguish cloud computing data centers from others. These are the tools we employ to design our system and combine the collective power of a multitude of data centers. Additionally, we provide an overview of related work.

2.1 Virtualization

Virtualization is an essential technology in current data centers. It allows for a flexible and configurable operating system environment setup. There are many types of virtualization that mimic the behavior of different layers in a computer system. For example, operating system virtualization emulate the behavior of the operating systems as observed by user level applications. In such cases, one can run an application built for operating system X over operating system Y. This is especially useful for running legacy software on modern operating systems. Hardware virtualization provides mediation between an operating system and the available hardware resources underneath. One of the prominent advantage of hardware virtualization is the ability to consolidate operating system instances onto a fewer number of physical machines. These are generally referred to as virtual machines or guest operating systems. Modern computer hardware have an abundance of resources that can be divided for use by more than one virtual machine. Consolidation is used on a wide scale in many data centers to reduce power consumption and increase resource utilization. However, it presents new challenges such as performance isolation and resource sharing. The virtual layer should provide performance guarantees and quotas to each virtual machine instance.

A multitude of virtualization platforms exist that can be used in data centers. For example, VMware provides solutions for full hardware virtualization [26]. Their proprietary system performs binary translation on guest operating system code to execute it safely on the hardware. This approach poses overhead that slows down the performance of the virtual machine instance. To remedy this problem, translated binary instructions can be cached for later use. However, caching space is limited and cannot cope with all of the needed translations. Another popular open source virtualization solution is KVM [5]. It offers full virtualization support but is limited to using hardware with virtualization extensions. OpenVZ is a successful operating system virtualization platform as it reports minimal performance overhead of approximately 3% [6]. However, offering virtualization containers at the operating system level limits its users to a single operating system. Furthermore, it uses a save-restore mechanism when migrating its containers which is unacceptable for critical applications.

In our design, we chose to use a flavor of hardware virtualization, namely, paravirtualization using the Xen hypervisor [11]. Paravirtualization requires some changes to be made to the guest operating system in order to attain higher virtualization performance on hardware without hardware virtualization extensions. Additionally, it is capable of hosting unmodified operating systems if the underlying hardware supports virtualization. Xen is a complex virtual machine monitor that provides a scalable solution for hosting virtual machines. It supports live migration in which virtual machines can be relocated to different physical machines without interruption. It is a popular virtualization platform being used in the industry. For example, Amazon's Elastic Compute Cloud (EC2) instances[2] are powered by the Xen hypervisor.

2.1.1 Xen Hypervisor

A running Xen system is structured into 3 main components shown in Figure 2.1; the hypervisor, the management interface and guest operating systems [11]. The hypervisor is the core of the Xen system that guarantees its stabil-

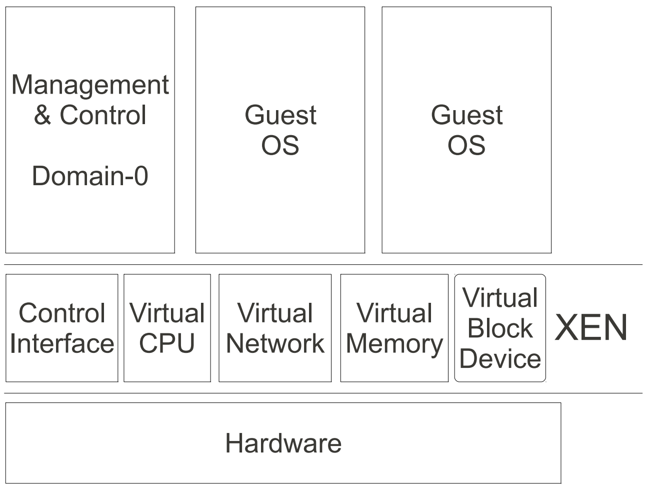

Figure 2.1: The architecture of a Xen system

ity and security. It is part of the system kernel that mediates the interaction between the hardware resources and the guest operating systems. The control and management component is an application level software suite that takes advantage of user level software libraries. It is used to administer the system and perform common tasks as creating a new operating system instance. Access to the control interface is limited to a single guest system that is referred to as Domain-0. Xen relies on different privilege levels supplied by processor architectures to isolate different guests from each other. For example, the x86 processors provide 4 privilege levels that are referred to as rings 0-3. Operating system code normally executes in the most privileged ring 0 that is capable of executing privileged instructions. On the other hand, application code executes in ring 3, the most limited level. In Xen, guest operating systems are forced to run in ring 1 which prevents them from executing privileged instructions directly. Instead, all privileged instructions are paravirtualized and must pass through Xen for validation and execution. Xen offers a set of hypercalls that are similar to system calls available in regular operating systems to perform such tasks. Moreover, guest operating systems are isolated from applications running in ring 3. Device I/O operations are abstracted by Xen from the guest operating system. All such operations are carried out through Xen using shared-memory to transfer buffers. Device interrupts are replaced by an asynchronous event delivery system that allows Xen to notify guest operating systems.

2.1.2 Virtual Machine Migration

Virtual machine migration is an essential component of scalable data centers. It assists administrators in consolidation management and resource utilization as they can relocate virtual machines from one physical machine to another. Xen is capable of running a multitude of guests on a single physical machine given that it has enough hardware resources. Additionally, it provides its administrators with the ability to migrate guest operating systems from one physical machine to another with negligible down time [14]. This is referred to as live migration of virtual machines. During this process, all primary memory of the virtual machine must be relocated and sent to the destination server. Disk storage is not relocated as it would take much more time to transfer. Instead, Xen relies on using a network file system to access a virtual machine's disk storage. When the primary memory is copied from a Xen system to another, all of the internal state is kept consistent. This means that process control structures and open files will not be affected.

Live migration in Xen is performed in an iterative process. It begins by initializing the virtual machine container at the target Xen system. In every iteration, memory pages of the migrating guest system are read by the management software in Domain-0 at the origin server and sent to the target Xen system. In every consecutive iteration, only the memory pages that have been modified since the last iteration are sent. This operation is carried out until the total amount of modified pages are small enough to be sent out together. In such a state, the guest system is stopped at the origin Xen system, the target system receives the last memory pages and starts executing at its new physical location.

However, migration in Xen depends on sending out an unsolicited ARP reply advertising the new physical location of the IP address bound to the virtual machine. Relying on ARP replies limit the mobility of virtual machines within a single LAN. This is one of the issues that we tackled in our architecture design and will present its solution in the following chapter.

2.2 Resource Provisioning in the Cloud

The development of resource provisioning mechanisms is important for the success of any cloud platform. Given the vast amounts of resources and accompanied operating costs, a successful data center must utilize its resources as efficiently as possible. Current data centers concentrate on provisioning compute resources to their applications and take advantage of high speed local networks. However, in a globally distributed cloud, the importance of network resources increases.

Provisioning and allocating resources to distributed applications in the cloud is a well studied problem because its essentials fall back on data center management literature. A variety of different approaches have been developed to utilize the resources efficiently. For example, resource provisioning for iterative jobs [18] analyzes multiple scheduling priority disciplines. VM Multiplexing [22] looks into joint virtual machine provisioning. Monitoring individual virtual machine resource consumption is a key aspect of understanding their needs. However, understanding the behavior of the applications running inside the virtual machine is not needed. This approach exploits the dynamic resource utilization patterns of virtual machines as they change over time. Consolidating underutilized virtual machines with those of high utilization can help increase total utilization of resources. For example, if a virtual machine was promised 2GB of memory but it was only using 1GB, the excess memory can be provided to other virtual machines that need it.

Meeting Service Level Agreements (SLA) is a corner stone of successful data center management. Much work has been done to maintain SLAs such as [28] which uses a control mechanism to efficiently adapt and provision resources as needed. This entails studying specific application behavior and monitoring its performance. For example, monitoring HTTP traffic and accounting for the response time of a web server. The SLA will include ranges of acceptable response times that need to be enforced by the system. If the web server failed to meet the SLA requirements, the system would provide more resources to the application so that it could keep up with the request load and frequency. One way of achieving this could be increasing the CPU share of the virtual machine hosting the web server. When the request frequency drops down, the system could reclaim some of the provisioned resources.

Most resource provisioning work focuses on providing compute resources to virtual machines. The traditional design of data centers relies on having all resources within one physical location. This simplifies the management and control of the whole infrastructure. Additionally, it takes advantage of having high bandwidth interconnects between all compute nodes. The notion of aggregating the resources of multiple data centers is uncommon although it offers great potential for scalability, crisis management and cost efficiency. However, it introduces new challenges such as consolidation on a global scale and the use of Internet bandwidth. In our work, we view network availability and properties as valuable resources and provision them as demanded by virtual machines.

2.3 Google App Engine

A good example of a platform capable of automatic resource provisioning is Google's App Engine [4]. App Engine is a cloud platform targeted for hosting web applications. It runs on the same infrastructure that hosts Google's own web applications such as Google Docs and Gmail. App Engine users do not need to worry about the provisioning of resources to the servers that host their content. This platform is capable of scaling dynamically to cater for millions of users. The App Engine environment is purely web based. No other forms of applications can be deployed on the App Engine cloud. Furthermore, applications using this cloud must utilize Google's own software development kit (SDK). They offer multiple SDKs for different programming languages, all of which are interpreted. They depend on interpreted languages because they harden their security by removing standard libraries that pose security risks. These include module loaders, process control and thread management functions. There are two unique forms of executable code in App Engine. The first is responsive code that gets invoked when a web request is received. The other form is batch processing which is not executed in response to requests. Instead, developers can issue requests to schedule long running tasks that typically aggregate data store entries. The execution time of the script cannot be specified by the developers. The platform receives the batch request and executed it when resources become available.

App Engine relates to our work as it automatically provisions resources to maintain web applications within the demanded service level agreements (SLA). However, it is targeted to host web applications only while this thesis aims to provision resources for arbitrary applications.

2.4 Content Delivery Networks

Content Delivery Networks (CDN) are good examples portraying the significant impact of relocating computing elements on the overall system performance. CDNs exercise efficient network utilization techniques to serve their clients. They place web servers in strategic positions in the network to reach their clients with minimal response delays. CDNs offer insight on the importance of network resources to distributed application performance. For example, web based systems that serve content to millions of clients usually employ CDNs to replicate their pages and serve them from locations in close proximity to the clients in a scalable fashion. CDNs are particularly efficient and scale well for read-only pages that are modified infrequently. There are many success stories of CDNs in the industry including Akamai [16] and Amazon's CloudFront [1]. The latter specializes in static or read-only documents [1]. Akamai has developed a massive CDN that is capable of serving static and dynamic content to clients [16]. Traditional structures of delivery networks have an origin server that contains all of the documents [16]. New or updated documents are stored directly on the origin server. A series of edge servers are deployed at strategic locations on the Internet to serve the clients quickly and balance the load. If an edge servers receives a request for a document that it does not have, it can quickly request it from the origin server, cache it and respond to the client request. The notion of proximity or distance of the edge servers is defined in terms of round trip latency, bandwidth and packet loss rates [16]. In Akamai's system, edge servers have fixed locations that have been picked to meet their client's requirements.

We view the network as a valuable resource to distributed applications. Our architecture's management of network resources were inspired by CDN's efficiency and scalability. The same principles of network distance apply to distributed applications. Opposite to the nature of CDNs, we do not assume detailed knowledge regarding the applications hosted in the cloud platform. The effects of bandwidth, latency and packet loss may vary from on application to another. The cloud platform analyzes the dynamic communication patterns of virtual machines. It uses this information to place virtual machines at locations which increase the utilization of the network resources. Additionally, it adapts to changing virtual machine behavior by continuously monitoring the communication patterns and migrating the virtual machines as needed. For example, if a web application was deployed over the cloud platform, the client requests would amount for the majority of the traffic. The system would adapt by placing the virtual machines hosting the web server processes at data centers in close proximity to the majority of client traffic.

2.5 Networking and Mobility

Current data centers restrict virtual machine mobility within a LAN as this can be solved by sending ARP unsolicited messages. However, our proposed architecture migrates virtual machines in wide area networks across

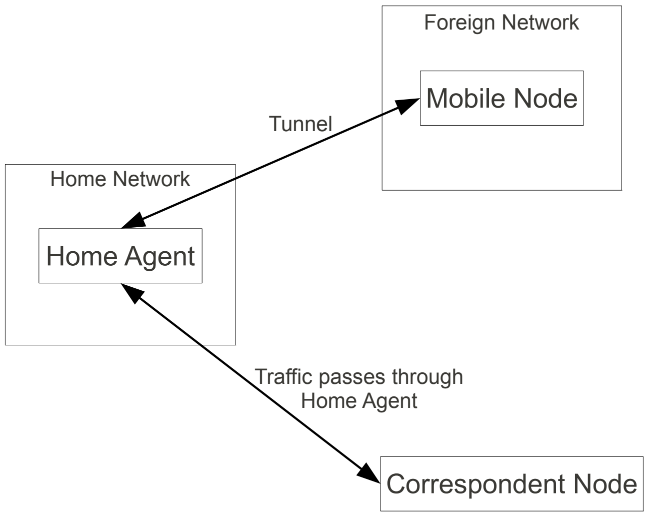

Figure 2.2: A mobile node in a foreign network using mobility for IPv4.

geographically distributed data centers. Therefore, we need a different networking mechanism to support virtual machine mobility. In the following sections, we discuss candidate network mobility technologies and how they relate to cloud computing environments.

2.5.1 IPv4 Mobility

Initially, IPv4 was not designed with mobility in mind. With the rising popularity of mobile agents such as notebook computers, it became apparent that mobility support is needed. This lead to the proposal of IP Mobility Support for IPv4 [23]. In IPv4 mobility, each mobile node is assigned a permanent or long term Home Address. This address will be used by correspondent nodes communicating with the mobile node regardless of its current location. If the mobile node was not attached to its Home Network, a router in the Home Network will act as its Home Agent. When a mobile node moves to a Foreign Network, it will detect an Agent Advertisement from the new network and get assigned a new Care-of Address. The Care-of Address resembles the new point of attachment of the mobile node to the network. The mobile node should then register its Care-of Address with its Home Agent. Network packets destined to a mobile node will always arrive at its Home Agent which forwards the traffic to the mobile node at its new

Care-of Address using a tunnel. Traffic sent from the mobile node could be tunneled back to the Home Agent and then forwarded to its destination. Alternatively, it could be sent directly using the Home Address as the source of the packets.

This form of mobility support triggers multiple concerns. Normally, network providers filter outgoing traffic that is not originating from their network for security reasons. As Figure 2.2 shows, packets being sent from the mobile node have to be tunneled to the Home Agent and forwarded from there. Packets destined to the mobile node will be received by the Home Agent and then tunneled to the Care-of Address. This increases the latency observed by the communicating parties due to the increased network hop length. Additional side effects include limiting the bandwidth of the mobile node to that of its Home Agent even if its current network provides superior connectivity. Furthermore, there is a constant overhead induced by the tunnel headers in every exchange between a mobile node and its Home Agent.

2.5.2 IPv6 Mobility

IPv6 and its extensions solve the mobility problems faced by IPv4 [15]. Mobility support in IPv6 relies on similar concepts such as the Home Agent and Care-of Address. However, using IPv6 mobility [21] and route optimization [10], correspondent nodes can identify the Care-of Address of a mobile node and cache it. The Care-of address can be used directly for subsequent communication. This scheme relieves the Home Agent from forwarding traffic back and forth and become a highly congested point in the flow. Additionally, there is no need to tunnel traffic, which avoids networking overhead.

There are many more advantages to using IPv6 other than its sophisticated mobility support. IPv6 provides a massive range of unique addresses which will play an important role in the growth of the Internet and data centers. Its introduction and deployment can only progress in phases as the legacy IPv4 only networks shrink in size. Therefore, efforts focusing on integration of both IP systems have been developed. The most notable work, presents a standard addressing mechanism that requires no tunneling to transfer IPv6 traffic through IPv4 systems [13].

In our proposed architecture, each virtual machine is assigned a globally unique IPv6 address. We rely on IPv6 mobility to maintain the reachability of virtual machines when they are relocated across different data centers. Therefore, established network connections will not break after the migration process. This superior technology provides us with the flexibility of choosing which data center should host a given virtual machine as its behavior change over time.

2.5.3 Amazon Elastic IP Addresses

Amazon EC2 offers a highly scalable web service that controls the creation and destruction of virtual machines referred to as Amazon Machine Instances (AMI) [2]. On the creation of an AMI, one must select the data center that will host it. Amazon provides Elastic IP Addresses that can be programmatically configured and attached to AMI instances. This service aids the masking of failures by allowing AMIs to takeover the roles of others. However, the state of the executing AMI is not migrated along with the new IP address assignment. Therefore, established network connections will break and the management of application state has to be performed manually.

2.5.4 Network Virtualization

Recent research on network virtualization[17] proposes solutions to the problems of mobility within IPv4 networks. The essence of network virtualization is to combine multiple physical routers to create a virtualized router. This creates the illusion that nodes are connected to the same network or subnet despite being in different physical networks. The system architecture is composed of multiple routers referred to as Forwarding Elements (FE) and a Centralized Control (CC) [17]. The CC instructs the FEs to create tunnels or VPN connections so that traffic can be forwarded between them transparently through the Internet. All FEs advertise themselves as a single network with BGP to the Internet. Each FE serves a set of nodes that are attached to it with IPv4 addresses. The CC has a global view of all nodes and their current attachments to FEs. Packets destined to a node in the network would be delivered to any of the FEs by the Internet. Once it enters the virtual network, the routing rules configured by the CC will usher the packets to the FE where the node is attached. From there, the packets will be forwarded to the destination node.

In a cloud computing setting, physical networks in different data centers can act logically as a single network. Each would have there own FE and be controlled by a CC. When a virtual machine migrates to a data center, it attaches itself to the FE that serves this new physical location. The CC can reconfigure the FEs so that packets destined to the virtual machine can reach it at its new location.

The drawbacks of this approach to network virtualization are subtle.

Since all FEs create the illusion of a single network, IP address assignments of all nodes must be unique across the whole system. For moderately sized networks, this solution can function efficiently. However, it is clearly troublesome for scaling with the limitation of IPv4 address space availability. The use of private IPv4 address ranges may remedy the scalability of this approach. However, having a unique presence over the Internet is a significant trait. By using IPv6 in our architecture, we guarantee that every virtual machine has a unique IPv6 address and maintain its identity. Furthermore, there is no need to develop custom routers and network devices as IPv6 mobility is an accepted network standard.

Chapter 3

System Architecture

In order to provide a solution to combining the resources of a multitude of data centers transparently, our proposed architecture relies heavily on virtualization using the Xen hypervisor and IPv6. Designing such an architecture entails studying and working out the relationships between several technologies to integrate them in a meaningful manner.

An important requirement of the architecture is that guest operating systems do not rely on IPv4 at all. Otherwise, after the relocation of a virtual machine over a wide area network, traffic generated by it will not be handled. Furthermore, it will be become unreachable as traffic destined to it will not be routed correctly. Therefore, all virtual machines must rely completely on IPv6 to deliver all types of higher level protocols.

Figure 3.1 highlights the main components of the system architecture. The system is designed as a set of globally distributed data centers. Each data center hosts a number of compute servers that are considered its core building block. Compute servers can be commodity computers or enterprise grade hardware. They run a stripped version of Debian Linux [3] powered by the Xen hypervisor. Moreover, all compute servers have unique IPv6 addresses. Figure 3.2 presents the internal structure of a data center. Virtual machines (VM) hosting user applications are deployed over compute servers and are also assigned unique IPv6 addresses different from those of the compute servers. A distributed monitoring system logs statistics of network traffic being exchanged between corresponding virtual machines. A control unit is fed by aggregating all the observed traffic statistics and decides how the system as a whole can benefit through relocating virtual machines to target locations. Finally, a distributed file system stores user application data and files. In the remainder of this chapter, we expose the internal

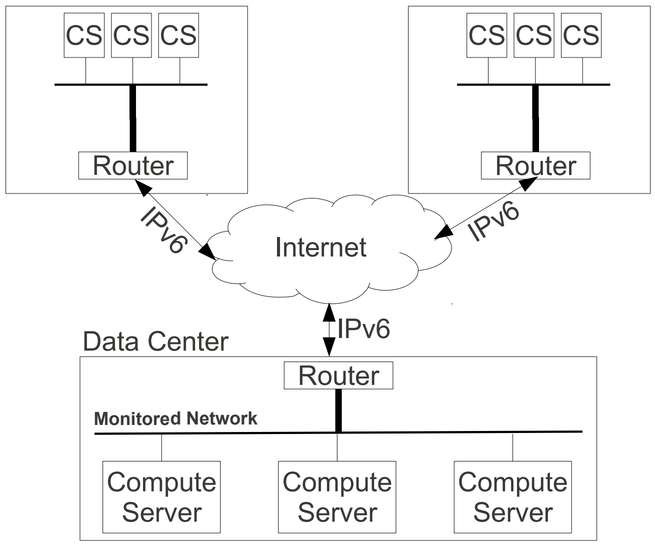

Figure 3.1: High level cloud architecture.

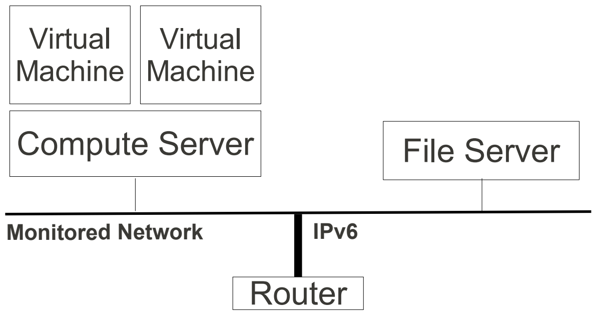

Figure 3.2: Data center design.

building blocks of our cloud architecture.

3.1 Virtual Machines

The purpose of the cloud computing platform is to provide users with computing resources and facilitate their control. The compute resources are abstracted by virtual machines. Virtual machines act as containers for processes, provide performance isolation and security. Therefore, it is crucial that the virtual machine design conforms to the goals of the system as a whole. In the following sections, we discuss the challenges faced in creating a VM template that meets the architecture's requirements.

3.1.1 Virtual Machine File system

To support virtual machine mobility, we need to carefully design their file system structure and the locations of servers hosting them. A scalable file system architecture that is capable of coping with the cloud is required. Since this cloud architecture is globally distributed, having a virtual machine in one data center and its files in another is simply unacceptable. For example, we cannot take advantage of local file system storage available to compute servers to store virtual machine disk images. This would limit the existence and mobility of a virtual machine to a single compute server. A popular solution to the storage problem that is being implemented in data centers is accessing the disk images over the Network File System (NFS) developed by Sun Microsystems. However, NFS poses several problems. It does not support IPv6 which is a core component of the architecture. Additionally, NFS serves file contents using Remote Procedure Calls (RPC), a client-server model. Therefore, files will be located at a central file server which is vulnerable to failures. Having backup servers ready to takeover in case of failures may remedy this problem. However, the data will be stored at a single physical location. When a virtual machine is migrated to a remote data center, its disk image will be left behind at the origin. File access time will definitely increase and may counter the benefits of migrating the virtual machine in the first place.

We intend disk access to be as fast as possible and available to all data centers alike. Instead of relying on NFS, we designed a mixed disk image structure which separates the user files from the operating system files. User files will be served using XtreemFS, a distributed file system [19]. On the other hand, operating system files are replicated across the compute servers where they will be accessed by virtual machines directly. This separation of files caters for high scalability.

Root File System

In our architecture, we employ Debian Linux as the operating system of choice for the virtual machines. This choice of operating system poses some file system limitations and challenges. For example, the Linux root file system can only be loaded from a local disk as viewed by the virtual machine. Therefore, the virtual disk contents needs to be provided by the host platform. This limitation forces compute servers to provide the virtual disk images of the virtual machines by storing them locally or accessing them over the network. However, storing the disk locally will hinder the mobility of the virtual machine. On the other hand, accessing the disk over the network might introduce inconsistencies after the migration of a virtual machine as the state of the open virtual disk image is kept at the origin compute server. If the origin compute server did not flush its local cache of modifications to the network, the virtual machine will read different file system blocks when it arrives at the new compute server. Therefore, we designed a structure in which the root file system is mounted read-only. We modified the base Linux file system structure so that it can be mounted in read-only mode while abiding by the Linux Standard Base (LSB)[8]. This entails some file system redirection hacks and using a RAM disk for a number of writable files. Any temporary directories and files created by the operating system will be stored in the RAM disk. For example, the tmp directory will be redirected and stored in the RAM disk. Moreover, the contents of the RAM disk are volatile as they will vanish with the destruction of the virtual machine.

The root file system structure is packaged in a virtual disk image and replicated on every compute server. It is intended to be mounted in read-only mode by all virtual machines and acts as their root file system. Since the file system is read-only, all virtual machines on a compute server can use the same disk image which saves disk space. The base file system is not meant to change over time, therefore, consistency is not a concern. When a virtual machine is relocated to a new compute server, the server will have the capacity of providing it with an identical block by block copy of its root file system. Therefore, any internal cache or file state that is maintained by the virtual machine will remain valid after its relocation. If a new disk image is later needed, it can be distributed to all compute server with a new identifier such that it does not collide with the previous version. Newly created virtual machines can start using the new disk image while older running instances can still use the older version. The root file system contains all required software and library dependencies needed by the virtual machine to function correctly.

User Files

User files can be dynamically modified by user applications therefore, they must be efficiently available regardless of the current placement of a virtual machine. Our file system of choice is XtreemFS [19]. XtreemFS is an object-based file system that was initially designed for Grid environments. It is composed of meta data servers and Object Storage Devices (OSD). It supports replication and stripping of data across multiple OSDs. XtreemFS offers adjustable policies that control the behavior and consistency of the file system. Additionally, it supports IPv6, provides a POSIX compliant interface and semantic behavior that make it a preferred choice. XtreemFS groups files into volumes which can be considered isolated file hierarchies. Each user is assigned a unique volume which provides the necessary isolation. The XtreemFS client is implemented using FUSE, a file system in user space module. When a virtual machine boots, it mounts the user files using the XtreemFS client and becomes ready to run user processes.

3.1.2 Virtual Machine Identity

Since all virtual machines use identical root file systems, a secure mechanism for distinguishing and identifying them is needed. For example, once a machine is started, it mounts its users' file system volume. We need to establish the identity of the machine before that in order to mount the correct volume. Additionally, no virtual machine should have the capability of assuming the identity of another and hijack its access rights. We solve this problem by supplying the virtual machines with unique kernel arguments. These arguments contain the domain name of the host and a secure key that is used for subsequent authentication. For example, a VM needs a valid key in order to mount the user's file system volume. A VM cannot forge the key and therefore cannot access other user's files. When a VM is started, it is dynamically assigned an IPv6 address through IPv6 router advertisements[24]. Other VMs cannot predict the address of the newly created VM. In order for the VM to be discoverable by other hosts, it will dynamically register its domain name entry with its newly assigned IPv6 address using its key. We use the standard dynamic Domain Name System Update protocol[25]. Once the user file system is mounted by the VM and the initialization is complete, the user can login to the VM to execute processes.

3.2 Compute Servers

The compute server is one of the core resources of the architecture. It is powered by Xen to provide a stable virtualization platform to user virtual machines. As stated earlier, Xen relies on a privileged virtual machine instance, namely Domain-0, to provide application level utilities that facilitates the management of the system. Most of the application level services provided by Domain-0 are implemented in Python. These services are bundled together in the Xen user-space daemon which supports creating a new virtual machine instance, booting it and requesting its migration from one Xen system to another. However, the Xen user-space daemon supports IPv4 only, which cannot cope with the network mobility requirements of the cloud architecture. Therefore, we modified it and added IPv6 support which was lacking. This enables the migration of virtual machines over IPv6.

We implemented a software library that is capable of querying and modifying the state of virtual machines remotely over IPv6. This library builds over XenAPI, a Xen user-space daemon module which is a growing effort to standardize the VM control interface [9]. Using this library, users can issue requests to create new virtual machines, start, stop and migrate them remotely.

Creating a new VM in our architecture is a simple task as its root file system image already exists. Therefore, no file copying or creation is needed. Instructing Xen to start a new VM with the disk image as its root file system will take effect immediately. Additionally, we pass the private key parameters to the kernel of the newly created VM to establish its identity.

Chapter 4

Streamlining Distributed Applications

In the last chapter, we covered the basis of our proposed architecture which aims to streamline globally distributed applications. In particular, it facilitates the relocation of virtual machines in a wide area network. This potentially allows the cloud platform to migrate virtual machines in order to optimize their communications. We now focus on selecting appropriate locations for network resources.

In our architecture, we assume that distributed application processes may be hosted by multiple distinct data centers. In a globally distributed application, communication efficiency plays an important role and affects its performance. In order to streamline the communication performance of distributed applications, our cloud platform first needs to monitor their network activity. In a second phase, it employs a greedy algorithm to relocate virtual machines and place them at improved locations in the network. The following sections discuss these two issues in detail.

4.1 Network Monitoring

Gathering precise network usage statistics is a challenging problem as we need to monitor the traffic produced and consumed by every hosted virtual machine. For our purposes, we gather bandwidth usage statistics only as a proof of concept. However, the system can be extended to account for other network properties such as latency. In our architecture, we do not assume any particular behavior of the hosted applications. Therefore, communication protocol information is irrelevant. We need to keep track of the traffic end points which are conveniently expressed by IP addresses.

The accuracy of the gathered network information is critical to the success of the cloud platform as it is going to be analyzed for the consolidation decision making. One might initially think that network packets traveling in between the data centers are the only ones that matter as they cause the most overhead by passing through the Internet. However, it is also necessary to track packets traveling in between VMs within a single data center as well. For example, consider a situation in which VMs A, B and C host a distributed application. Assume that VMs A and B are located within the same data center and have high bandwidth network interconnects in between them while node C is hosted at a remote location. If A and C started communicating together intensely for an extended period, the cloud platform would need to estimate the opportunity cost of having A and C reside at closer locations to improve their communication efficiency. An important component of this estimation is the communication history and intensity witnessed between A and B. It may be the case that leaving the VMs in their current locations is the preferred setting because A and B collaborate more.

It is clear that no centralized network monitoring is up to the task of gathering all of the needed data. In this section, we discuss a multitude of candidate distributed monitoring schemes. In all of the proposed schemes, network usage statistics must be collected and aggregated to be ready for analysis.

4.1.1 Routers & Switches

Network devices such as routers and switches are ideal for collecting statistics about network usage. They are located at strategic positions in the network and can distribute the logging load across multiple topological levels. Stored statistics can be periodically delivered for analysis. However, gathering per VM statistics may become a huge burden since it will increase their processing and storage demands. Additionally, having deployed devices from proprietary vendors will limit the ability to extend their features. Moreover, network packets communicated between VMs residing on the same compute server will never traverse the physical network. This means that collected network statistics will be incomplete.

4.1.2 VM logging

Every virtual machine is aware of the network traffic it produces and consumes. Aggregating and logging network usage information at the level of the VM can be another alternative. A user level application could log the raw packets as they travel over the network interface. However, the accuracy of user level applications would depend on the network traffic speed as well as the processing load forced by executing applications. Extending the kernel can be an efficient solution to receiving precise statistics about the network usage. However, a user level application would need to read this data from the kernel and transmit it over the network for analysis periodically. All of these approaches add overhead to the processing of the VM as it would need to dissect every packet it sends and receives. Additionally, the user level applications can be manipulated by the VM's owners as they have privileged access to it.

4.1.3 Compute Server Logging

Compute servers offer unique advantages to monitoring network traffic as they act as an intermediary between the VMs and the network. The privileged Domain-0, acts as a network bridge and forwards traffic generated by the hosted VMs to the physical network. Similarly, it delivers traffic destined to the VMs that was received from the physical network. If two co-located VMs interact, the generated traffic will not reach the physical network and will only travel through the internal network bridge. To have minimal performance impact on the compute server, a logging component can be added to the system kernel. The kernel of Domain-0 processes all network traffic anyway and can be extended to log aggregate statistics. This information can be exposed to application level software running in Domain-0 through an additional set of system hyper-calls. A user level application can periodically update its logs and deliver the aggregated information to a central repository where the analysis will take place. It is important to note that no VM can modify or access information residing in Domain-0.

The research presented in Hop-count filtering[20] shows how the Linux kernel can be modified to track network hops between a host and its IPv4 corespondents in memory with minimum overhead. We argue that similar techniques apply to storing aggregate bandwidth values. The memory demands of storing IPv6 addresses are clearly larger. However, passing the data through system calls to a user level application periodically and clearing the used state tables in the kernel will reduce the amount of memory

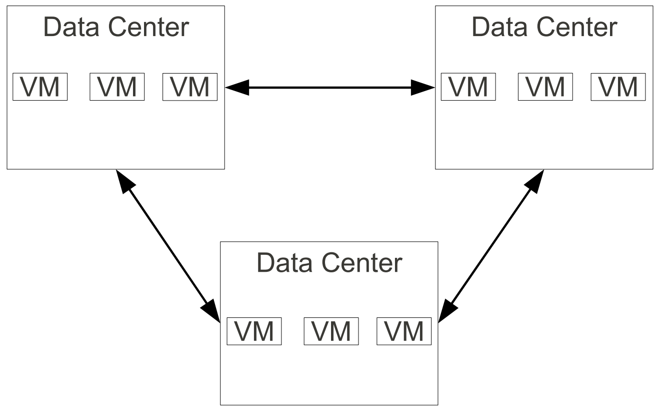

Figure 4.1: Data centers in a cloud represented as a graph.

required. Additionally, the address clustering mechanisms presented in Hop-count filtering can greatly reduce the memory footprint. This is particularly important for external IP address ranges that are not part of the platform and interact with the cloud.

4.2 Adaptive Relocation of Compute Nodes

Our proposed architecture adapts continuously to improve the network paths available in between VMs. It does so by analyzing the communication patterns of nodes and relocating them to improved locations in the network. In the following sections, we discuss our approach to solving this problem. We will present our abstract representation of the problem and discuss the algorithm used to solve it.

4.2.1 Problem Representation

In order to develop an algorithm that relocates virtual machines, we represent the cloud platform as a graph as shown in Figure 4.1. Given the network topology properties, we can draw a directed graph representing the data centers as vertices and the available network links as weighted edges. We can measure properties such as the maximum bandwidth that each network edge can endure. Furthermore, we use the VM network usage statistics to load the edges. We assume that compute servers within a single data center have minimal latency and unlimited bandwidth links available between them. Therefore, they are all expressed by the data center's single vertex and there is no need to represent them individually. VM network traffic that travels between data centers will be aggregated and added to their corresponding edges.

4.2.2 Adaptive Relocation Algorithm

The relocation algorithm relies on a greedy heuristic which aims to optimize a cost function. The cost function provides a uniform metric by which the algorithm can judge the utilization of the available resources. The algorithm iteratively considers the relocation of virtual machines without actually migrating them. When a final virtual machine placement is reached in which no more optimizations are possible, the cloud platform can order the migrations to proceed.

Cost Functions

The power of using a cost function lies in its expressiveness. A cost function can represent multiple aspects of efficiency combined together. Each can be assigned a weight that signifies its importance. For example, the following is a list of aspects that are important to data centers.

Bandwidth Bandwidth availability is a crucial aspect of connectivity. If the path between two data centers is overloaded and congested, it would be advantageous to decrease the bandwidth usage in between them. This can be done by selecting the VMs in those data centers that collaborate the most and relocate them elsewhere.

Round trip latency Minimal network delays and high responsiveness is crucial for interactive systems. One should avoid deploying real time systems over a high latency network path.

ISP data charge rates Some ISPs charge more than others. If certain VMs are communicating together intensively it would be less costly to host them within the same data center. If this is not possible, selecting data centers in ISP regions that cost less can significantly reduce costs.

Data center load and capacity Some data centers may be overloaded while other are mostly idle. Sharing hosting load between them guarantees higher quality of service. It also facilitates the monitoring of used resources.

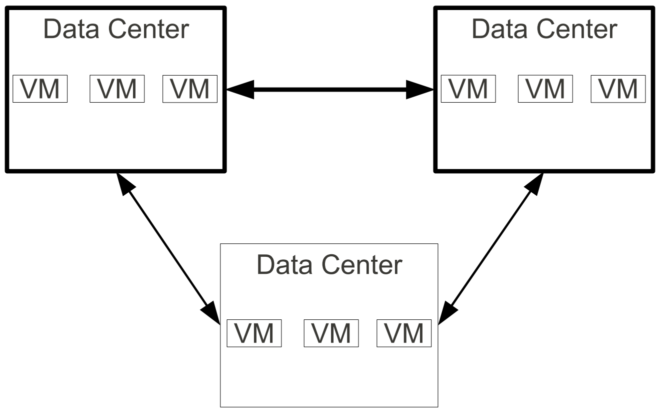

Figure 4.2: The highest loaded edge between two data centers selected by the algorithm.

Data center charges Power billing rates may differ from one region to another. Utilizing low rate regions helps cut down costs.

The cost function we use is the average bandwidth loading factor of all inter data center network links.

Algorithm

Algorithm 1 presents the outline of the relocation algorithm developed. Graph edges between data center vertices have known maximum bandwidth capacities. Each virtual machine in a data center contributes by consuming a portion of the available data center bandwidth. As the algorithm is greedy, it starts by locating the highest loaded network link between two data centers as presented in Figure 4.2. To reduce the overhead of that link, it locates the two VMs VM_0 and VM_1 that contribute the most to its load. Figure 4.3 highlights both selected VMs. It estimates the benefit of migrating VM_0 or VM_1 to different data centers and selects the highest scoring relocation operation. If the benefit of the relocation as expressed by the cost function optimizes the current cost, the migration move is simulated. The algorithm iterates continuously and attempts to find the next highly loaded network link. If the benefit of the relocation proves to reduce the efficiency, the algorithm terminates.

Algorithm 1 The relocation algorithm

```
$DataCenters$ {set of data centers}
$cost \leftarrow 1$
$newCost \leftarrow 0$
while $newCost < cost$ do
  $cost \leftarrow Cost()$ {calculate current cost}
  $newCost \leftarrow cost$
  $selectedVM \leftarrow NIL$
  $(DC_0, DC_1) \leftarrow Max\_Edge()$ {get maximum loaded edge in graph}
  $(VM_0, VM_1) \leftarrow$ {VM pair that contribute the most edge}
  for $i \in 0, 1$ do
    for $dc \in DataCenters - \{DC_i\}$ do
      if $HasCapacity(dc, VM_i)$ then
        if $COST(VM_i, dc) < newCost$ then
          $newCost \leftarrow COST(VM_i, dc)$
          $selectedVM \leftarrow i$
          $selectedDC \leftarrow dc$
        end if
      end if
    end for
  end for
  if $selectedVM \neq NIL$ then
    $Migrate(selectedVM, selectedDC)$
  end if
end while
```

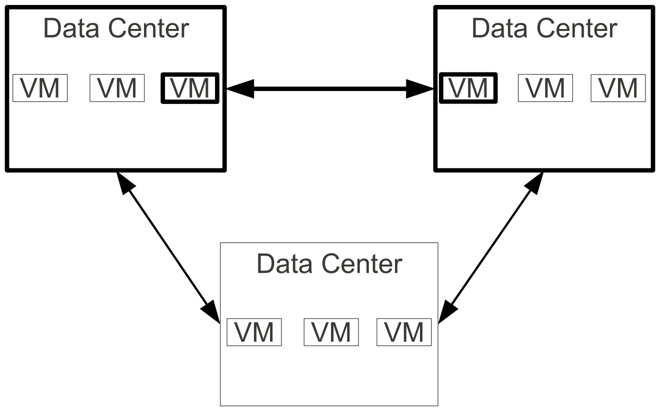

Figure 4.3: The virtual machines that consume most of the congested link's bandwidth highlighted.

Chapter 5

Evaluation

In order to assess the architecture presented in this thesis, we carried out two different types of experiments. First, we developed a testbed as a proof of concept that verifies the specified integration of technologies and studied the overhead and feasibility of migrating virtual machines over IPv6. Secondly, we simulated executions of the relocation algorithm with a variety of data center configurations. In the following sections, we present evaluations of both type of experiments.

5.1 Architecture Evaluation

In order to test the feasibility of the proposed cloud architecture, we developed a testbed that follows the guidelines presented in this thesis. We employed 4 compute servers to represent 4 geographically distributed data centers. Moreover, a single machine operated as a file server. One more machine was employed as a router which was placed at the center of a star topology. All network traffic produced by the compute servers and file server is routed through the router machine and directed to its destination. Moreover, the router simulates a wide area network by throttling bandwidth between the other machines in the network. All of the utilized machines are identical 1GHz Pentium-IIIs with 1GB of RAM and a fast Ethernet interface. To assess the overhead incurred by migrating virtual machines over IPv6, we compare the migration of an idle virtual machine versus that of one hosting a loaded web server. We deployed the PHP implementation of the Rice University Bidding System (RUBiS) running over the Apache2 web server in a virtual machine on the first compute server [7]. Another virtual machine hosted an instance of Mysql which ran on the second com-

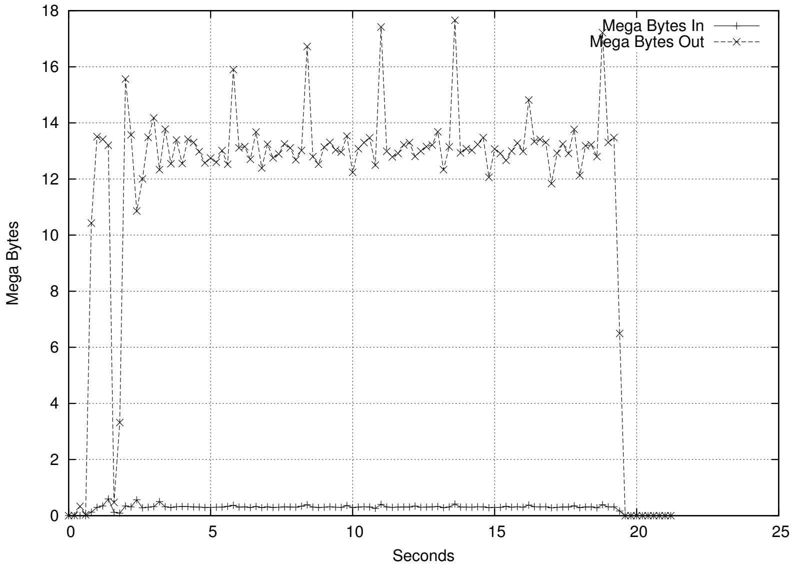

Figure 5.1: Migration of a virtual machine hosting a loaded Apache2 web server running the PHP RuBiS benchmark

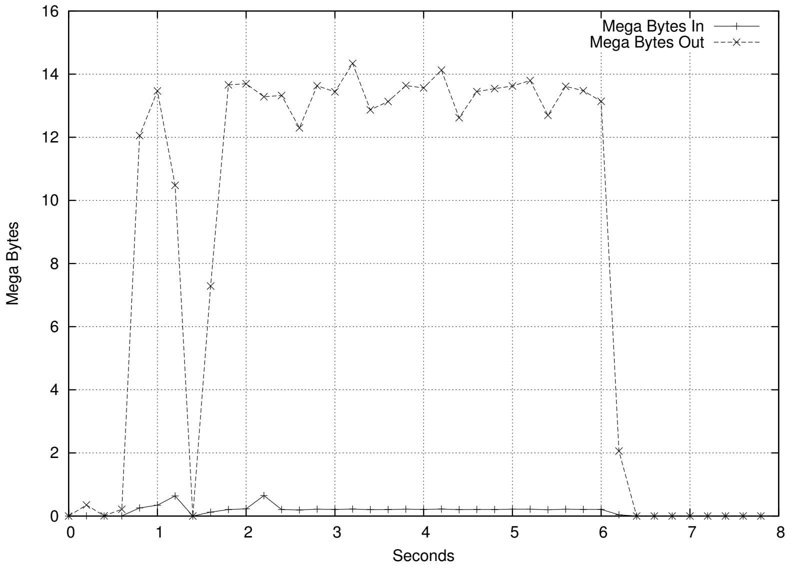

Figure 5.2: Migration of an idle virtual machine

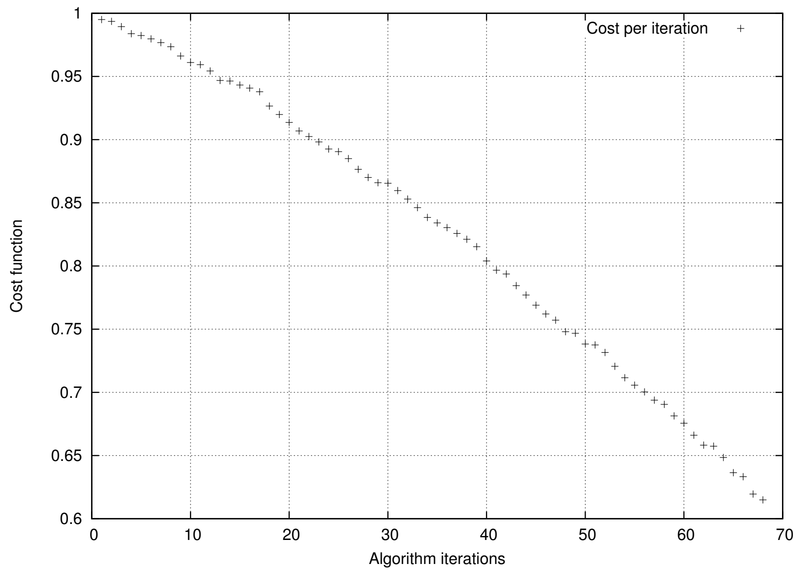

Figure 5.3: Adaptation algorithm progress with 1,000 VMs dominating the bandwidth capacity

pute server. Finally, the workload generators that emulate client requests to the web server were deployed on a virtual machine on the third compute server. While the web server was actively serving pages, we issued a migration request for its virtual machine to relocate to the fourth compute server. During the migration process, the router at the center of the star topology logged the rates of data transfer. The traffic caused by the relocation is presented in Figure 5.1. Similarly, the rates witnessed by the router while migrating an idle virtual machine is shown in Figure 5.2. The relocation of the loaded virtual machine took 19 seconds while the idle one took around 6 seconds. This is due to the rapid memory page updates caused by the executing processes which elongate the iterative migration procedure. A common feature of both figures is a valley in the outbound rate at approximately 1.5 seconds into the migration process. This is caused by the initial processing of the migration request. When a Xen system receives a relocation request, it first verifies that it has enough resources to host the relocating virtual machine and then creates a resource container for it.

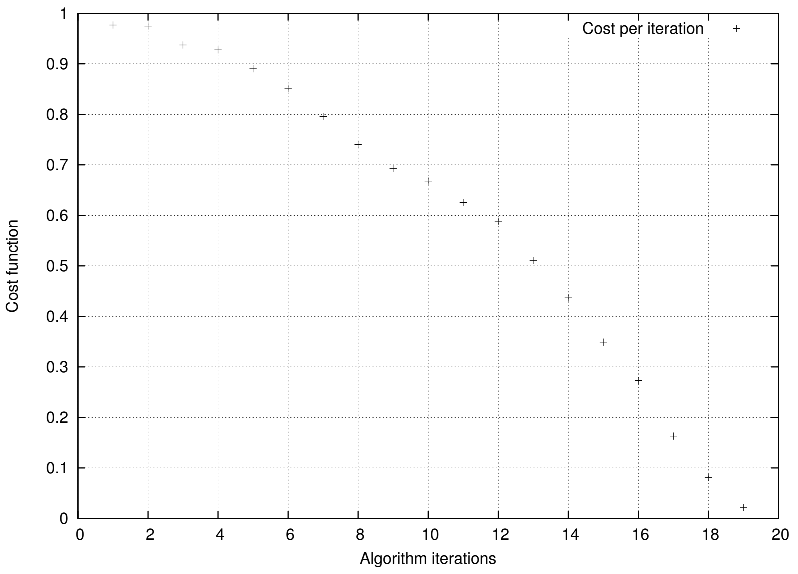

Figure 5.4: Adaptation algorithm progress with 100 VMs dominating the bandwidth capacity

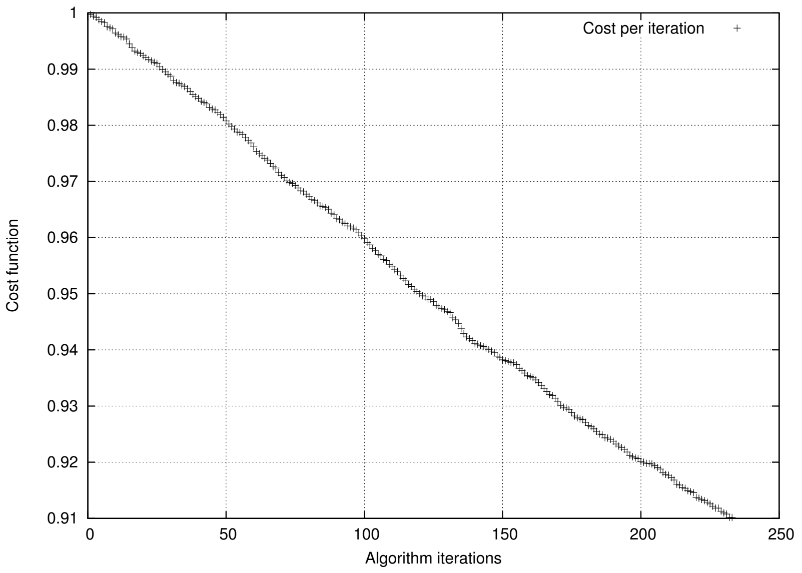

Figure 5.5: Adaptation algorithm progress with 10,000 VMs dominating the bandwidth capacity

5.1.1 Algorithm Evaluation

In order the assess the performance of the relocation algorithm, we simulated a multitude of data center environments with different capacities and communication patterns. To our best knowledge, no data centers publicized their capacities or network properties. Therefore, the values used in our simulations may not reflect actual real world values. Figure 5.3 presents the progress produced by the relocation algorithm in an environment in which 1,000 virtual machines dominate the wide area links between 4 data centers. The cost function is defined as the normalized average bandwidth across all inter data center network links. It is clear from the figure that the relocation algorithm produces incremental progress while simulating the relocation moves. After reaching a final state in which no more progress can be obtained, the algorithm terminates. Similarly, Figures 5.4 and 5.5 present the progress in environments in which 100 and 10,000 virtual machines dominate the wide are links respectively.

Chapter 6

Conclusion

The cloud computing paradigm of computation is increasingly gaining more attention. It is particularly interesting for businesses as it provides them with a cost efficient solution for hosting their applications. Therefore, data centers provide their infrastructure as a service to host a variety of systems. The rising popularity of the cloud poses new technical challenges on data centers. Namely, with the rising popularity of the cloud platform, more businesses and clients will demand resources which will lead to an overload of the available infrastructure. The current state of the data centers that provide cloud computing infrastructure remain isolated to a large extent. In this thesis, we aim to provide an architecture that solves the isolation problem of data centers. In particular, we offer a solution in which the resources of multiple data centers can be aggregated in a transparent manner. We employ hardware virtualization technologies to provide stable containers for user applications. Moreover, virtual machines guarantee performance isolation and security. We use IPv6 mobility to transparently relocate virtual machines across data centers without disrupting their live execution and the applications they host. This methodology caters for the different needs of clients as user applications need not worry about the location of their applications as long as it is performing well. Moreover, our testbed evaluations confirm the integration of the utilized technologies.

We demonstrated the benefits of combining the resource of multiple data centers. The proposed platform can be extended to implement policies and heuristics that improve it. For example, having server instances follow the sun to cater for daytime users. Moreover, we introduced a relocation algorithm that is capable of increasing the utilization of resource and reduce operating cost. Since there is much potential in this area, we believe that developing an accurate cost function that expresses the needs of utilization is a promising direction for further research.

Bibliography

[1] Amazon CloudFront. http://aws.amazon.com/cloudfront/.

[2] Amazon Elastic Compute Cloud *AmazonEC*2. http://aws.amazon.com/ec2/.

[3] Debian – The Universal Operating System. http://www.debian.org/.

[4] Google App Engine for Business. http://code.google.com/appengine/.

[5] Kernel Based Virtual Machine (KVM). http://www.linux-kvm.org/page/Main_Page.

[6] OpenVZ. http://wiki.openvz.org/Main_Page.

[7] RUBiS. http://rubis.ow2.org/.

[8] The Linux Foundation: Linux Standard Base (LSB). http://www.linuxfoundation.org/collaborate/workgroups/lsb.

[9] XenAPI. http://wiki.xensource.com/xenwiki/XenApi.

[10] J. Arkko, C. Vogt, and W. Haddad. Enhanced Route Optimization for Mobile IPv6. RFC 4866 (Proposed Standard), May 2007.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

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

[13] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), Feb. 2001.

[14] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[15] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871.

[16] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally Distributed Content Delivery. *IEEE Internet Computing*, 6(5):50–58, 2002.

[17] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song. Enhancing dynamic cloud-based services using network virtualization. In *VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 37–44, New York, NY, USA, 2009. ACM.

[18] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu. Resource provisioning for cloud computing. In *CASCON '09: Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, pages 101–111, New York, NY, USA, 2009. ACM.

[19] F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario. The XtreemFS architecture—a case for object-based file systems in Grids. *Concurr. Comput. : Pract. Exper.*, 20(17):2049–2060, 2008.

[20] C. Jin, H. Wang, and K. G. Shin. Hop-count filtering: an effective defense against spoofed ddos traffic. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 30–41, New York, NY, USA, 2003. ACM.

[21] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775 (Proposed Standard), June 2004.

[22] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient resource provisioning in compute clouds via VM multiplexing. In *ICAC '10: Proceeding of the 7th international conference on Autonomic computing*, pages 11–20, New York, NY, USA, 2010. ACM.

[23] C. Perkins. IP Mobility Support for IPv4. RFC 3344 (Proposed Standard), Aug. 2002. Updated by RFC 4721.

[24] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), Sept. 2007.

[25] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. Dynamic Updates in the Domain Name System (DNS UPDATE). RFC 2136 (Proposed Standard), Apr. 1997. Updated by RFCs 3007, 4035, 4033, 4034.

[26] VMware. Understanding Full Virtualization, Paravirtualization, and Hardware Assist, Sept 2007. White paper, http://www.vmware.com/resources/techresources/1008.

[27] A. Weiss. Computing in the clouds. *netWorker*, 11(4):16–25, 2007.

[28] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. In *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, page 25, Washington, DC, USA, 2007. IEEE Computer Society.