Aggregated Service Consolidation by Multi-Server Virtualization

Phillip Uwizeye Nkubito

This thesis is submitted in partial fulfillment of the academic requirements for the degree of

Masters of Science in Electrical Engineering

in the Faculty of Engineering and The Built Environment

University of Cape Town

November 2012

As the candidate's supervisor, I have approved this dissertation for submission.

Name: Dr. Alexandru Murgu

Signed: _____

Table of Contents

Declaration

I hereby declare that this thesis is my own unaided work, both in conception and execution, and that apart from the normal guidance of my supervisor. I have received no assistance except as stated in the text document. Neither the substance nor any part of the thesis has been submitted in the past, or is being, or is to be submitted for a degree in this University or any other University.

I am now presenting the thesis for examination for the degree of MSc in Electrical Engineering. I also grant the University free license to reproduce the above thesis in whole or in part, for the purpose of research.

Phillip Uwizeye NKUBITO

_____ Date _____

Acknowledgments

I would like to express my most heartfelt gratitude by acknowledging the following people who have made the completion of this thesis a possibility. I would like to thank Dr. Alexandru Murgu for allowing me the opportunity to work under his supervision. His enthusiasm, guidance and perspective allowed me to really apply myself and think critically throughout my research.

My thanks also go to all my friends and colleagues in my research group for their invaluable support and advice when it mattered most.

To my parents and siblings, I thank you for everything you have done to get me to this stage of my life. Your prayers and continuous encouragement throughout have made this all worthwhile.

Abstract

Virtualization is seen as the underlying technology that will make ubiquitous internet a reality by giving service providers a means to manage and consolidate their services on fewer infrastructures, while also being able to still deliver agreed service levels to their customers. Data centres have taken on server virtualization due to its benefits such as reduced operational cost, centralized administration, enhanced security, load balancing and improved hardware utilization. The resource management capabilities offered by server virtualization implies that these providers can now make use of previously redundant computing resources by aggregating them into a pool of resources.

Virtual machine (VM) live migration is a core feature in resource management as it allows running applications in the VMs to be relocated to another host so as to maintain Service Level Agreements (SLAs) or for host maintenance purposes, with near zero degradation to the performance of the hosted service. However there is a need to manage optimally the live migration process especially when dealing with heterogeneous workloads such as CPU, memory and network intensive hosted applications which put a strain on different computing resources. By doing so, a service provider reaps maximum benefits from the live migration process.

This study aims to optimize the live migration process during the migration of a VM running a Voice over IP (VoIP) service from a source to target host residing on the same Local Area Network.

The proposed design framework focused on modifying the pre-copy algorithm which is in charge of performing the live migration. The modifications introduced a statistical dirty page tracking scheme into the algorithm and also incorporated the use of CPU scheduling to improve the performance of the live migration. A proof of concept was setup in the form of a testbed using Xen as the virtualization platform. Experiments carried out, showed that the downtime and migration time of the VoIP service were significantly reduced with minimal degradation on the call quality as perceived by the user during the live migration.

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface
CAPEX	Capital Expenditure
CentOS	Community Enterprise Operating System
CPU	Central Processing Unit
HPC	High Performance Computing
IaaS	Infrastructure as a Service
iSCSI	Internet Small Computer System Interface
ITU-T	International Telecommunications Union – Telecommunications Standardization Sector
KVM	Kernel-based Virtual Machine
MOS	Mean Opinion Score
NAT	Network Address Translation
NGN	Next Generation Network
NIC	Network Interface Card
OS	Operating System
OPEX	Operating Expenditure
PaaS	Platform as a Service
PCPU	Physical CPU

PDV	Packet Delay Variation
RAM	Random Access Memory
RTP	Real Time Protocol
SaaS	Software as a Service
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SSA	Slowdown Scheduling Algorithm
VCPU	Virtual CPU
VDI	Virtual Desktop Infrastructure
VM	Virtual Machine
VoIP	Voice over IP
WAN	Wide Area Network

1. Introduction

The prospect of the next generation internet is that it will be ubiquitous; offer seamless links between its users and their everyday essential internet services irrespective of what device is being used or their geographical location. This implies connectivity anytime, anywhere with any device. The development and efficient delivery of these internet services are evolving from their current state to be able to achieve the above milestones.

Cloud computing is currently a hot topic both within industry and academia alike. According to the National Institute of Standards and Technology (NIST), cloud computing is "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1].

With this paradigm shift, service provisioning is now instantaneous, scalable with reduced upfront investment and has better performance as well as fault-tolerant capabilities [2]. Amazon, Google as well as Microsoft are some of the big industry players offering services in the cloud.

The International Telecommunication Union (ITU) technology watch report of 2009 [3] highlighted cloud computing as a candidate technology for standardization work within ITU. At the time of writing this paper, cloud computing and virtualization are being included as part of an ITU-T study group in future networks [4].

The development of Service Oriented Architecture (SOA) is coming to the forefront as a means of improving service interaction on the web. SOA is a method of logically designing means to offer service provisioning to end users or other distributed services in a network via a published and discoverable interface [5]. This development is briefly mentioned so as to put in context the efficient delivery aspect of these services as a result of virtualization technology; a key concept in this research.

In the context of a service provider, a data centre is a secure facility which centrally stores and manages all the servers running business critical applications. Traditionally, data

centres had a silo-oriented architecture as shown in Fig.1-1 below. An application had a dedicated server along with all its resources; RAM, CPU, storage and networking infrastructure controlled by an OS, serving the hosted application's needs. In trying to fulfill SLAs, these data centres over-provisioned these resources so as to meet peak resource demand of their hosted applications. This resulted in a high peak-to-average ratio on the server's resource consumption. Also, 'server sprawl' [6] came about as a result due to the large number of 'one server - one application model' that were deployed.

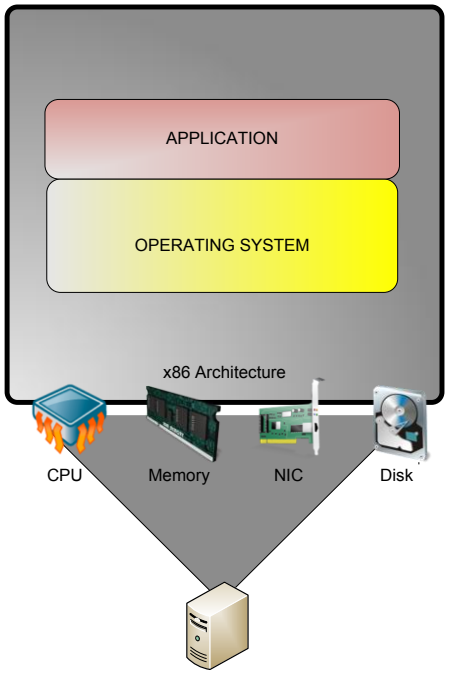


Figure 1-1: Silo model.

To address the challenges associated with the silo-model as shown in Fig.1-1 above, server virtualization has been widely adopted by service providers in their data centres. Data centres have taken on server virtualization due to its benefits such as reduced operational cost, centralized administration, enhanced security, load balancing and improved hardware utilization [7]. With ever rising costs and a need to maintain stringent Service Level Agreements (SLAs), service providers are turning to virtualization as a solution to their aforementioned problems [8].

The resource management capabilities offered by server virtualization implies that these providers can now make use of previously redundant computing resources, and still meet the performance criterion required by the user. By consolidating servers and their hosted

services through virtualization in a data centre, a pool of computing resources is also created. This allows service providers to create host server clusters and manage the resource needs of their services on a per-demand basis.

A multi-tenancy model is adopted as shown in Fig.1-2. Multiple virtualized servers, and implicitly the services they host are now located on the same physical host. These virtual instances are referred to as Virtual Machines (VMs).

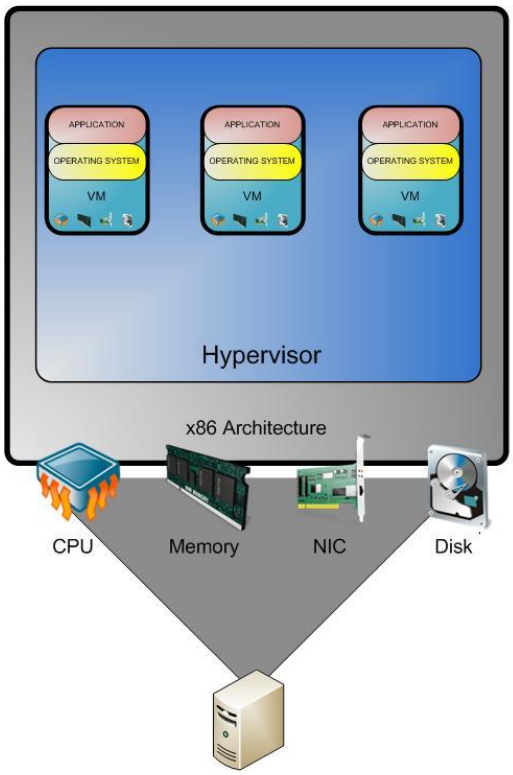


Figure 1-2: Multi-tenancy model.

A core resource management feature in server virtualization is live migration. VM live migration is the transfer of a running VM's active memory and execution state seamlessly over a shared network infrastructure from a source to a destination physical host, with no noticeable disruption to the hosted application from a user's point of view. The benefits of this to service providers is improved manageability, performance as well as fault tolerance [9].

With this capability, a service provider can balance the system load across the available physical hosts as well as automate VM migration from one host to another in the event of a catastrophic fault on one of the hosts. This plays a crucial role in service availability as described in a SLA. In data centres where workload dynamics are ever changing, live migration plays a role in ensuring efficient usage of all available pooled resources. Through live migration, considerable energy savings are made [10], [11]. This is achieved by monitoring VMs' workloads and migrating them into fewer physical hosts during off-peak periods.

The consolidation of servers through multi-server virtualization, offers business enterprises and service providers a resource pool on which to run their business critical services. VM migration is central in resource management and maintaining of SLAs. It is therefore imperative that the live migration process be managed efficiently to suit the needs of the service provider in offering and guaranteeing a specified quality of service to the end user.

1.1. Problem Definition

VM live migration, as earlier mentioned, plays a pivotal role in resource management within a virtualized environment. There is a need to optimally manage the live migration process especially when dealing with heterogeneous workloads such as CPU, memory and network intensive hosted applications which put a strain on different computing resources.

For service providers, the need for a near zero performance degradation during live VM migration is paramount in trying to meet the stringent SLA while avoiding the heavy financial penalties which are associated with violating these agreements.

1.2. Research Objectives

The primary objectives of this research are to:

i. Investigate the live migration process and its role within a virtualized environment as a resource management mechanism.

ii. Propose a pre-copy scheme that will alleviate the performance degradation experienced by a real-time service during the live migration process.

iii. Implement the proposed scheme above in a virtualized environment with consolidated services running in VMs on an open source hypervisor.

Goals specific to the pre-copy scheme of the live migration process are:

- To minimize the downtime and migration time of the VM, and inherently, the service being accessed by the user.
- To analyze and assess the service performance with metrics which are service-specific after carrying out the live migration.

1.3. Scope and Limitations

The main scope of this thesis will be to present an implementation that minimizes the degradation of service performance during a live migration within a LAN. Live migration across a WAN is beyond the scope of this research but study areas have been highlighted for future work as will be discussed in the last chapter.

This work will be limited particularly to server virtualization and will not delve into the details of cloud computing. Cloud computing is only mentioned to give the reader a case study of where virtualization is extensively deployed. Virtualization is intrinsic to cloud computing and is the enabling technology that makes it a viable solution for service provisioning in the cloud.

This thesis introduces server virtualization and its benefits to service providers as well as business enterprises. In this work, the benefits of live migration as a resource management tool will be illustrated to show the maintenance of SLA in terms of service availability. In trying to demonstrate real-world applications, the availability of a real time service will be considered in this work.

1.4. Research Contributions

It is envisioned that this research will offer more insight into the performance of consolidated services running in a virtualized environment. The performance of these consolidated services during live migration will be improved upon by implementing and assessing the proposed design framework.

Through a practical implementation using real-world applications, the work will illustrate the significance of resource managing the live migration process and how this translates as a benefit to the service provider when considering SLAs.

The use of VoIP service as a case study in this research is in line with the adoption of VoIP as a value-added service for Telco-service providers and this study may be further applied to a real network.

1.5. Thesis Outline

The remainder of the thesis follows the below structure:

Chapter 2 gives a broad review of the literature related to server virtualization and the benefits it offers to service providers in facilitating service consolidation. Related works that have guided this research are reviewed as well in this chapter. VM migration which is the cornerstone of this research is also explained in detail with particular attention being given to the pre-copy algorithm that is used in the live migration of VMs.

Chapter 3 introduces the design framework to optimize the live migration process using a statistical approach. This begins firstly by giving a detailed model of the pre-copy algorithm and highlighting key parameters that are vital to the performance of the live migration. The uses of both a standard deviation tracking scheme as well as CPU scheduling are introduced as a means of achieving the research objectives stated.

Chapter 4 discusses the evaluation framework in which the proposed concept in the preceding chapter will be implemented. A breakdown of the hardware and software used in the evaluation of the design framework is stated in this chapter. The software used in the evaluation process is for traffic generation and performance monitoring during the live migration. Clear procedures of the experiments that need to be carried out to for impact assessment are also explained in this chapter.

Chapter 5 is an analysis of the results obtained from the experiments carried out on the testbed setup. Different scenarios are setup to assess different aspects such as the testbed's performance capabilities and most importantly the design framework. Parameters relevant to the live migration performance as well as the effect of the design implementation on the service quality are also recorded and discussed.

Chapter 6 then proceeds to give conclusions as observed from the work undertaken in this research. Potential areas for future work are also recommended as a point of continuation of this thesis.

2. Background and Literature Review

In this chapter, relevant background is discussed so as to give the reader an understanding of the subject matter within this work. Related works that have guided this research are reviewed as well in this chapter.

The chapter begins by expounding on the key concept of what server virtualization is and its merits in aggregating services within a data centre environment. It later introduces the VM live migration feature and the role it plays maintaining SLAs between service providers and their clients. An overview of related works pertaining to the pre-copy algorithm that is used in the live migration of VMs is discussed.

2.1. Server Virtualization

Virtualization was first developed by IBM Corporation in the 1960s [12] to partition mainframe computers into logical instances and run these instances on the same physical mainframe. This was done so as to increase the efficiency of the mainframes as well as reduced maintenance overheads. From the late 1990s new virtualization platforms have come into existence firstly by the entrance of VMware and subsequently other major players such as Microsoft and Xen.

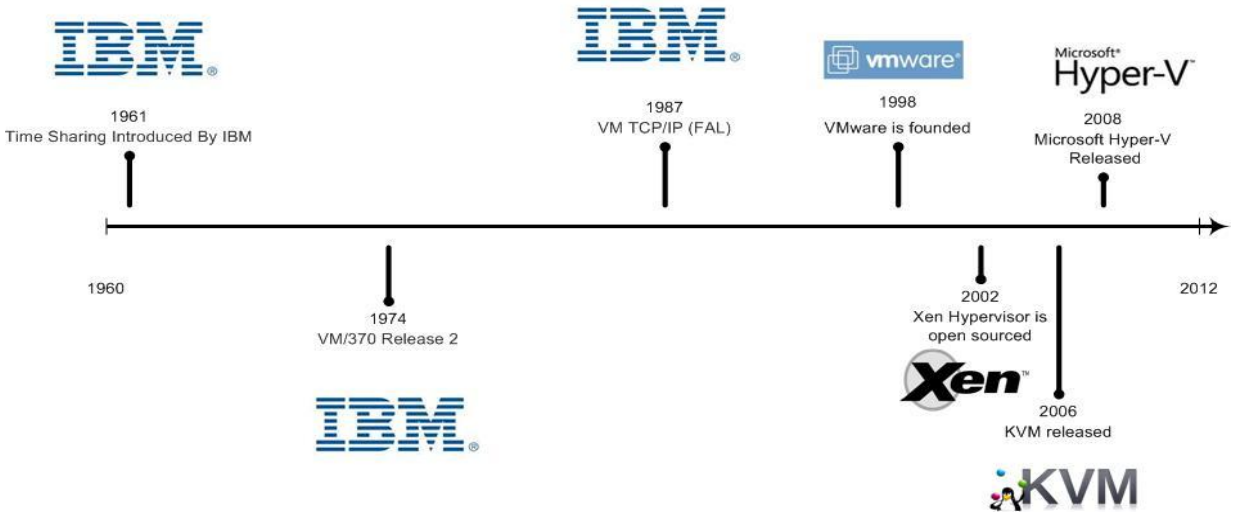


Figure 2-1: Timeline of virtualization.

Server virtualization is the technology that introduces a layer of software abstraction between a computer's hardware and the operating system (OS) on which an application is

running. This abstraction layer is referred to as a virtual machine monitor (VMM) or the hypervisor as illustrated in Fig.2-2.

The hypervisor is a low level virtual machine monitor that runs on the physical hardware and is loaded during the boot process. The primary role of the hypervisor is to handle requests for access to physical hardware resources, trap and handle protected or privileged instructions [13]. The hypervisor hence acts as a controller of the hardware resources and with this, multiple guest OS can be run simultaneously on the same physical computing system. These independent runtime environments are referred to as virtual machines (VMs).

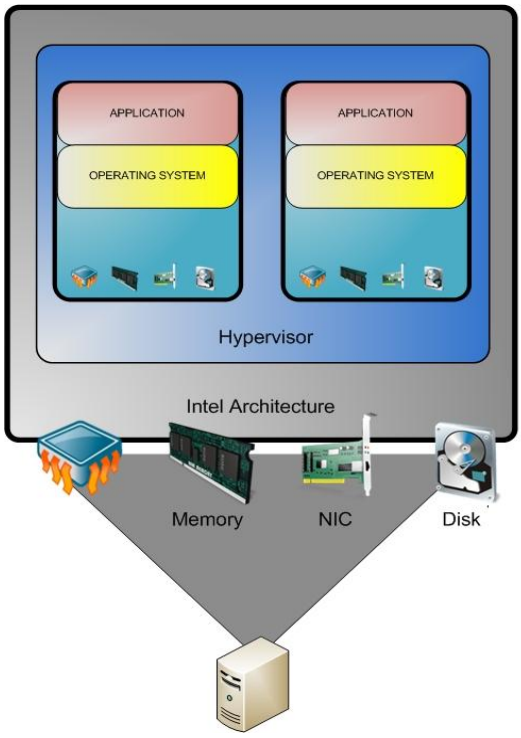


Figure 2-2: Server virtualization.

Hypervisors can also be classified as hosted or bare-metal [12].

A hosted hypervisor exists as an application running on the operating system installed in a physical computer. The benefit of this is that the hypervisor makes use of the host OS's resource management capabilities as well as hardware compatibility. This means that the hosted hypervisor is therefore easier to write and support. Inherently, the shortcomings in terms of inefficiency of the host OS also is adopted by the hypervisor thereby affecting its overall performance. Another drawback of the hosted scenario is the resource overhead

requirements are much higher than in the bare-metal scenario. Examples on the market using this type are VMware Workstation and Oracle VM VirtualBox.

Bare-metal hypervisors are directly installed onto the physical computer's hardware. This implies that the hypervisor has direct control of the computing resources and therefore has less resource overheads as compared to its hosted counterpart. The drawback is that it is more difficult to write and support though it offers a much better native performance for the VMs that run in this environment. Examples include Xen, VMware ESXi as well Microsoft Hyper-V.

The two most common types of virtualization techniques are full virtualization and paravirtualization.

Full virtualization offers complete abstraction of the computing resources. It incorporates code into the hypervisor that emulates the underlying computing hardware when necessary thereby enabling unmodified guest OSs to run on top of the hypervisor. The drawback of this type of virtualization is that emulation overheads are incurred [13]. Hypervisors that support this type of full virtualization are VMware ESXi and KVM.

Paravirtualization requires modification to the guest OS so that it can communicate with the hypervisor. Paravirtualization's improved performance over full virtualization is because the OS modifications enable the operating system to communicate directly with the hypervisor, and hence has near physical hardware performance. The Xen hypervisor supports the paravirtualization technique.

Virtualization is now being widely adopted due to its benefits it offers in data centres by consolidating services into the same physical infrastructure thereby reducing OPEX and CAPEX to service providers.

2.2.1. Service Consolidation

Traditionally, data centres had silo models; where a service had its own dedicated server resulting in a one-server one-application scenario. This meant that the profile of the workload running in these servers needed to be known so as to provision sufficient computing resources. This would ensure that even at peak demand, the QoS requirements were met thereby avoiding the violation of SLAs.

The drawback is that there was under-utilization of these computing resources when the services running on these servers were not running at their peak and thus having low efficiency. In enterprises environments it has been observed that 5-40% of the server's resources are actually used on average [14]. To illustrate this, take for example an internet service provider hosting an online shopping website that gets high traffic requests only during festive seasons. This implies that during the rest of the year, resource utilization is very low yet the service provider still has to incur operational costs to maintain the servers hosting this service.

The consolidation of IP telephony applications over a virtualized platform is also being considered by telecommunications solution providers [15] due to the previously mentioned benefits. Fig. 2-3 below gives an illustration of the main components that constitute an enterprise IP telephony system.

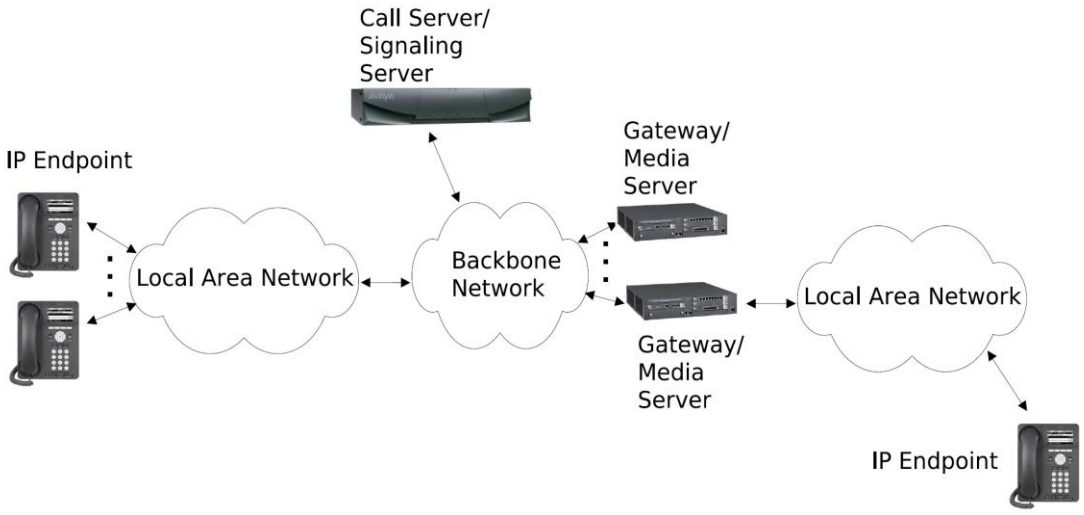


Figure 2-3: An enterprise IP telephony system [15]**.**

Components that reside within the service provider's data centre are the media server and call server.

The media server primary role is to handle incoming media streams, process and forward them to the appropriate client within the network. In the figure above, the IP endpoints represent IP phones at the end user site. Functions such as echo cancellation, tone detection, transcoding and jitter buffer manipulation are carried out in this server.

The call/signaling server initiates and terminates calls between the IP endpoints within the network. It is also in charge of features such as transfer, hold as well as multi-party calling [15].

Through the consolidation of these servers by virtualization, a service provider therefore is able to improve on resource utilization and reduce cost of operating individual silos as demonstrated in Fig.2-4. The media and call/signaling servers are now virtualized entities and are even aggregated with another VM running a different service modeled as a CPU intensive VM.

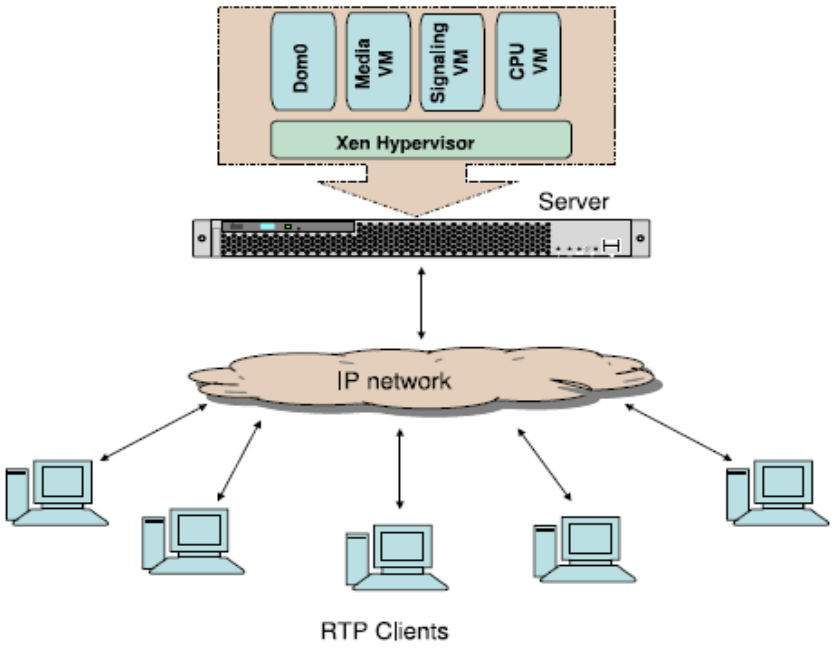


Figure 2-4: Modeling enterprise telephony workloads [15] **.**

2.2.2. Virtualization and Cloud Computing

Virtualization is seen as the technology that makes cloud computing viable both from a technical and business point view by offering scalable, on-demand service provisioning from a pool of computing resources.

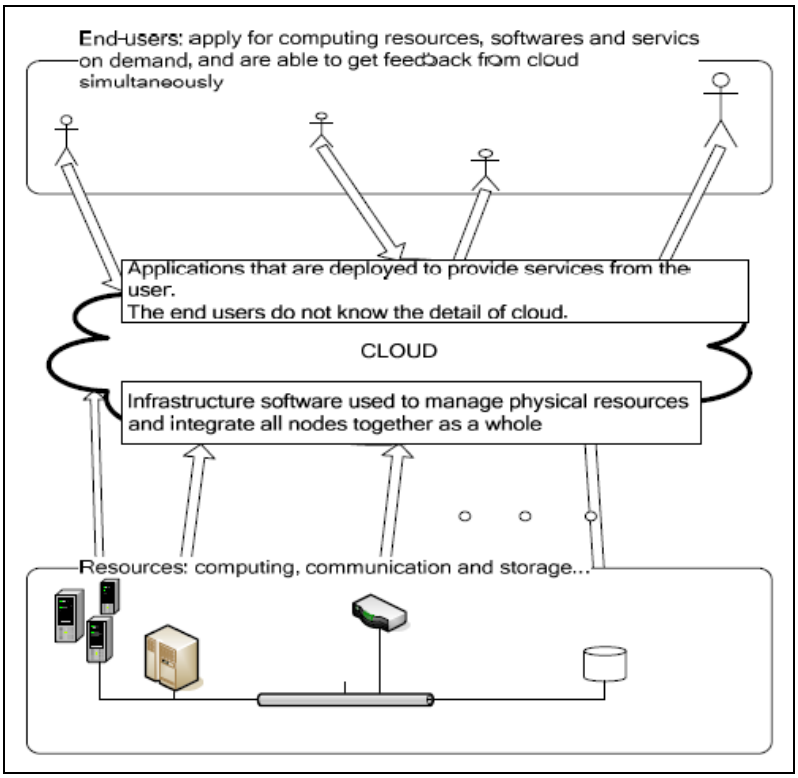


Figure 2-5: Model of pattern for cloud computing [16]**.**

In the above Fig. 2-5, virtualization is what abstracts the resources and offers them to the cloud.

In cloud computing, there are 3 service models [1] namely:

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

The end user has a choice of requesting for the above three types of services.

IaaS providers offer virtualization platforms to the consumers. These consumers purchase the computing resources (CPU, RAM and storage) from these providers and are billed according to the duration and the amount of resources consumed [17]. The consumer can deploy and manage their software on this virtual instance. An example of an IaaS provider is Amazon Elastic Compute Cloud (EC2) [18] which uses Xen as the virtualization platform on which to offer these services.

PaaS allows consumers to create and deploy their own applications created using programming languages offered to them by the service provider. The consumer only has access to the platform application environment and not the underlying infrastructure. Google App Engine is an example where users can create web applications in a Python or Java environment [17].

SaaS offers consumers ready-made applications created by the provider and hosted in the cloud. They may be accessed by the user through thin clients such as web browsers. Consumers are limited to only having control of user-specific configurations. An example of this type of service model is Google Drive package as well as web-based email [17].

To reiterate as was mentioned in the scope of this work, the research will not delve any further into cloud computing, but rather merely acknowledge the relationship between virtualization and cloud computing.

2.2.3. Hypervisors

Currently the four common virtualization platforms that are widely deployed on the market are:

- VMware
- Xen
- Microsoft Hyper-V
- KVM

Xen is an open source virtual machine monitor or hypervisor which allows the multi-tenancy of guest operating systems on the same physical computer. In Xen, these guest operating systems or virtual machines are referred to as domains as illustrated below in Fig.2-6. Domain 0 (dom0) acts as the sole administrative point for Xen's control functions such as create, shutdown or save of other guest virtual machines referred to as domain U (domU). Dom0 has privileged access to the underlying hardware through the hypervisor and exports abstractions of hardware devices to the domUs [19].

VMware is a proprietary hypervisor and is recognized as having the largest share in the hypervisor market [20]. In VMware the guest operating systems are referred to as Virtual

Machines (VMs). VMware has a range of virtualization platforms to cater for both end users as well as data centre and cloud infrastructure owners.

KVM [21]is a kernel-based open-source virtualization platform that only supports full virtualization. KVM kernel is included in Linux, as of version 2.6.20.

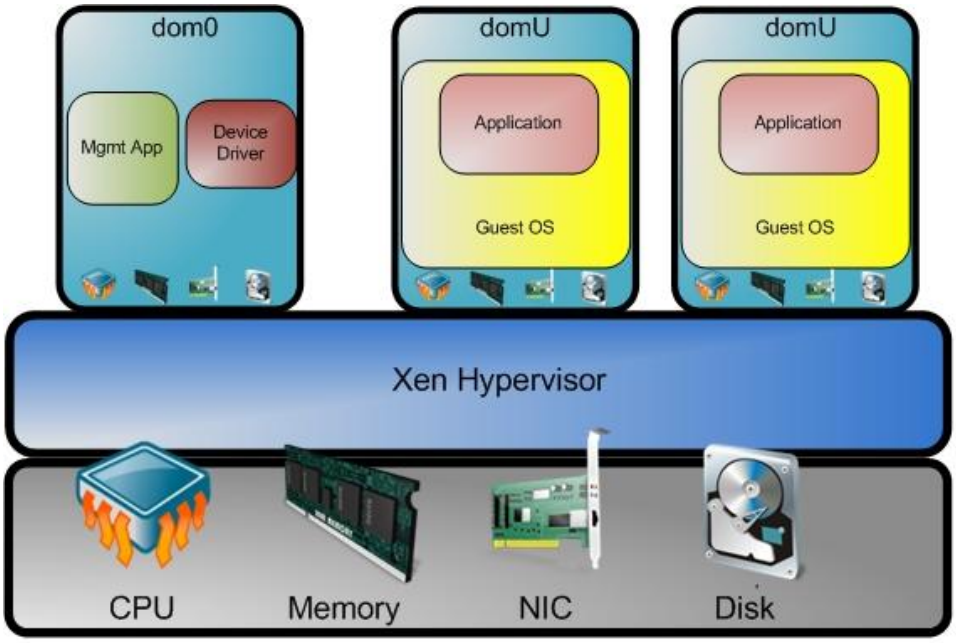


Figure 2-6: Xen hypervisor and domains.

Microsoft Hyper-V [22] is a Microsoft product that enables the consolidation of workloads onto one physical machine thereby reducing costs as well as improve efficiency. This platform is used often as a Virtual Desktop Infrastructure (VDI) environment where a client accesses their own virtual desktop through a thin client.

2.2. Xen's CPU Credit Scheduler

As has been mentioned earlier on, the Xen hypervisor's role is to act as a controller and offer the computing resources to the hosted VMs or domains. These resources include CPU, RAM, storage as well as NIC.

The credit scheduler [23], is a proportional fair share CPU scheduler deployed by Xen. The credit scheduler automatically load balances guest virtual CPUs (VCPU) across all

available physical CPUs (PCPU) on an SMP (symmetric multiprocessor) host. Each domain is assigned a weight and a cap to tune per VM scheduler parameters.

The weight parameter controls proportional share of CPU per guest VM. The default is 256. Therefore a VM with a weight of for example 128 will get relatively half the time on the PCPU as compared to a VM with weight of 256. Permissible values for the weight range from 1 to 65535.

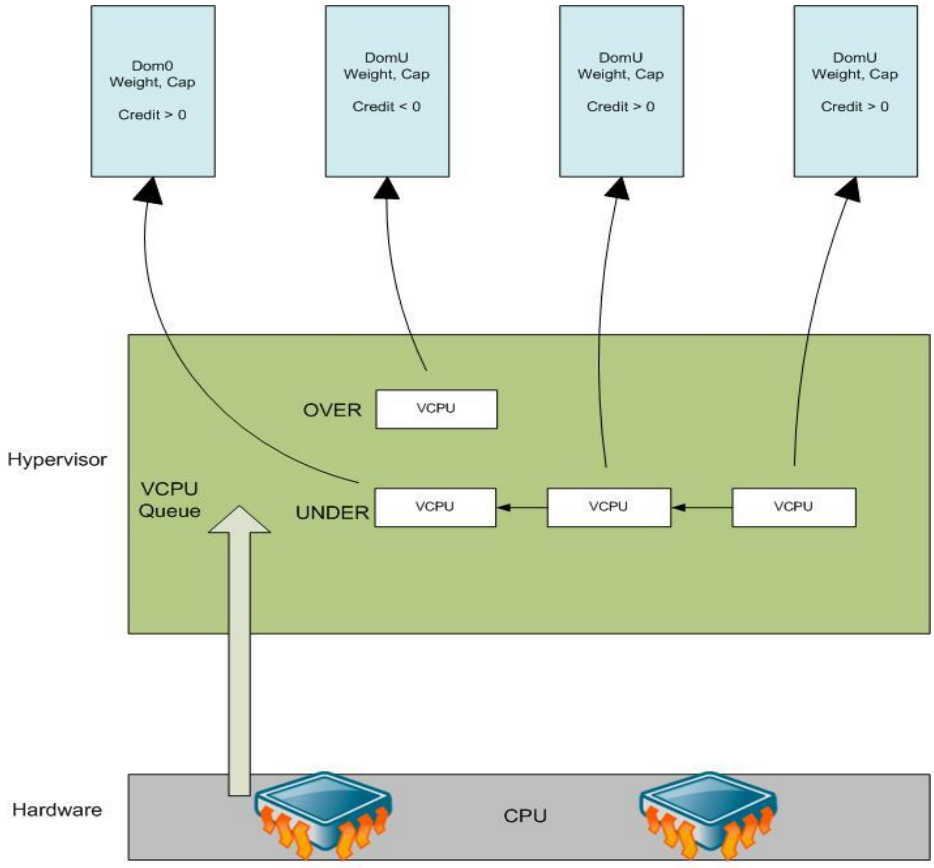


Figure 2-7: Credit scheduling in Xen.

The cap optionally fixes the maximum amount of CPU (in %) a domain will be able to consume. By default Xen sets a cap of 0, which means there is no upper cap i.e. any available CPU will be utilized by the VM.

In Xen's credit-based scheduling, [13] each CPU manages its own local queue of VCPUs each belonging to their respective VMs. The credit scheduler allocates credits to the VCPUs based on a function of the weight assignment of the VM. As the VCPUs run they

consume their allotted credits as illustrated in Fig.2-7 above. This credit allocation determines the priority of the VCPUs which may either be “over” or “under” and this represents whether the VCPU has or has not exceeded its fair share of CPU resource in the current accounting period. Fig.2-7 illustrates VCPU priorities as a result of the credit scheduling for the domains. A positive credit implies that the VCPU priority is “under” whereas a negative credit implies that the VCPU is in the “over” priority.

2.3. Virtual Machine Live Migration

VM live migration is the transfer of a running VM’s active memory (RAM) and execution state seamlessly over a shared network infrastructure from a source to a target physical host, with no noticeable disruption to the hosted application from a user’s point of view. The benefits of this to service provider is improved manageability, performance as well as fault tolerance [9]. During peak periods, resource utilization such as CPU and memory usage on a physical host can be high such that the application/service running is negatively affected. By having the ability to migrate the VM to a less congested physical host on the network, load balancing is achieved across the network infrastructure. Also with this capability, an automated VM migration from one host to another in the event of a catastrophic fault improves service availability as described in a SLA.

During a live migration, downtime refers to the brief period in which the migrating VM is not running or unavailable as it moves from the source to the target host. Total migration time refers to the period from when the migration begins until the VM is up and running on the target host.

2.3.1. Migration Related Fundamentals and Approaches

Live migration involves fundamentally the transfer of memory pages from a source to a target host. Paging is a memory-management scheme that allows the physical address of a process to be noncontiguous [24]. In other words, this is the means by which a computer accesses information from its hard disk for use in its main memory (RAM) during the execution of a process.

When a process is to be executed, data is retrieved in same-size blocks known as pages by the computer. The block sizes are computer architecture dependant and are typically powers of 2 and may range from 512 bytes to 16 Megabytes.

Memory migration techniques that exist tradeoff between downtime and total migration time [25]. In [26] a pure-stop-and-copy approach is discussed. This design trades off a shorter total migration time for a longer downtime by suspending the VM during the entire memory migration period and only restarting it on the target node upon completion.

A pure demand approach [27] contrasts the latter technique. A short stop-and copy phase for the transfer of essential kernel structures to the destination is carried out. The rest of the memory pages are transferred when accessed by the target node. The downtime is minimized at the expense of a longer total migration time.

Both approaches have negative repercussions in terms of perceived quality of service to a user accessing the service running in the migrating VM.

A pre-copy approach is hence adopted as a balance between downtime and total migration time [28]. In this technique, modified memory pages or dirty pages are iteratively copied as illustrated in Fig.2-8. Each iteration copies modified memory pages of the previous round to the destination host. In other words, this iterative mechanism is done in order to periodically check for dirtied pages that occurred during the previous iteration. The primary aim of this is such that both downtime and the total migration time of the VM are kept at a minimum.

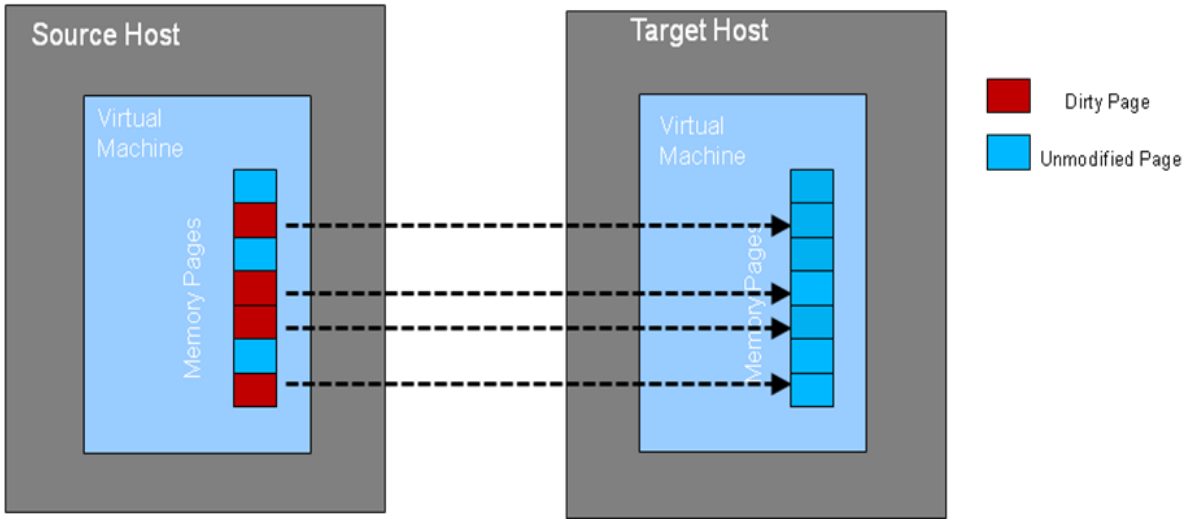


Figure 2-8: Snapshot of a pre-copy iteration during live migration.

Factors that affect the duration of the total migration and downtime period are mainly the migration link bandwidth and the dirty page rate. The latter refers to the rate at which a VM's memory pages are being modified during each pre-copy iteration [25].

2.3.2. Live Migration and High Availability

High availability [15] refers to a mechanism that ensures a seamless automated failover to the target host in the event that the source host catastrophically fails for whatever reason. What must be noted is that the source and target must belong to the same cluster for this to be possible, and that the target is always continuously replicating the VM running at the source. The robustness of live migration is further enhanced for disaster recovery by incorporating this automated process. The redundancy by continuously replicating the VM's state on the target host however, does have its drawbacks and these are addressed with a proposed solution in [29].

RAID (Redundant Array of Independent Disks) as an option is used to further enhance redundancy of shared storage which is paramount for live migration to take place. In this research, a discussion regarding shared storage and live migration is highlighted in chapter 4 and in more detail in appendix C.

2.3.3. Live Migration Performance

With the wide adoption of virtualization in data centres and also as the driving technology behind cloud computing, the performance and reliability of live migration is increasingly becoming a very important feature in resource management within these entities.

The performance modeling of live migration is addressed by Wu *et al* in [30]. The literature emphasizes the importance of understanding live migration performance and proposes performance modeling based on the application behaviour and resource availability. The authors carry out live migration experiments on VMs running different resource intensive applications in Xen and create a profile for the migration time. Regression methods are then used to model the performance based on the collected profiling data. The results of their work highlight that CPU resource allocation to dom0 in Xen plays a significant role in the migration performance.

Akoush *et al.* in their work [25] discuss the importance of key parameters that affect the live migration process namely; the page dirty rate and the migration link bandwidth. Their work considers prediction of live migration performance using two models; AVG (average page dirty rate) model and HIST (History based page dirty rate) model. The former assumes a constant page dirty rate for the applications running with the VM. The latter is implemented when the VM's dirty rate cannot be approximated as an average.

2.3.4. Optimization of Live Migration

There has been work that has proposed solutions to optimize the default Xen live migration process. In so doing, it is envisaged that the service downtime and migration time will be decreased during this process.

Hu *et al.* [31] proposed a time-series based pre-copy approach for live migration. In this work, using a time-series prediction technique, high dirty pages can be identified for past and future periods with more precision and only transmit these frequently updated pages during the last iteration. This reduces on the number of retransmissions that need to be carried out copying the same dirty pages as well as downtime of the VM.

In [32] Atif *et. al.* considered live migration in a HPC environment. In this work, it is noted that in HPC environments the cost of migration is not so much the downtime but the

effect it has on the application as memory is being copied from source to the target host. Their work proposed a very simplistic approach to optimizing the live migration process for HPC applications in SMP clusters running on a Xen hypervisor. By limiting the number of pre-copying iterations to only two, there is a reduction in the total migration time and hence an improved application performance as compared to the default implementation.

Liu *et al.* [33] describes and proposes a Slowdown Scheduling Algorithm (SSA) . The primary target of this algorithm is to reduce the page dirty rate of the migrating VM by systematically decreasing the CPU time allocated to the VM by the source host. This leaves more CPU resource for other co-hosted VMs. However, this has a negative impact on the performance of the application running within the migrating VM.

2.4. Pre-Copy Algorithm

Pre-copy approach is widely implemented by hypervisors during live VM migration. Hypervisors such as VMware [34] , Xen [35] and KVM use this algorithm for migration between source and target hosts.

As discussed, this technique tries to balance the downtime and the total migration time of a migrating VM. The stages [28] involved in the algorithm are:

a) *Pre-Migration* – A target host is pre-selected for future migrations by the source host. The VM is running at the source host at this point.

b) *Reservation* – The necessary computing resources are reserved by the target host for the incoming migrating VM.

c) *Iterative pre-copy* – During the first iteration the entire RAM is copied from the source to the target host. Subsequent iterations copy only the memory pages that were modified during the previous iteration. These modified memory pages are referred to as dirty pages.

d) *Stop-and-copy* – The VM at this stage is suspended so as to copy its CPU state as well as any remaining memory pages to the target. At this point both the target and the source host have consistent copies of the VM.

e) *Commitment* – The source host releases the VM state after the target acknowledges it has received a consistent image. The target now becomes the primary controller of the VM.

f) *Activation* – The VM is then activated on the target host. Post migration code runs to reattach device drivers to the VM.

It is important to highlight that the iterative pre-copy stage is resource intensive in terms of CPU as well as networking on the source host. This has an impact on the application running in the VM in terms of performance. A downtime is observed at d) as no CPU cycles are processing the VM at neither the source nor the target host.

In order for the iterative stage described in c) to exit and go into the next stage, some conditions need to be satisfied. In the case of Xen [25], there are 3 stop conditions:

1) *Less than 50 memory pages were dirtied during the last pre-copy iteration.*
2) *29 pre-copy iterations have been carried out.*
3) *More than 3 times the total amount of allocated RAM to the VM has been copied to the target host during the iterative stage.*

These conditions are important in defining the duration of the downtime as well as the total migration. Condition 1 ensures that the downtime is kept to a minimum as few pages will be transferred in the stop-and-copy phase. Conditions 2 and 3 simply force the migration into d) irrespective of how many modified pages that still exist at the source host. This therefore will have an impact on the downtime of the VM.

2.5. Post-Copy Algorithm

A converse to the pre-copy algorithm discussed above is the post-copy algorithm. In this approach the transfer of a VM's memory pages is carried out after its executing state and essential kernel structures have been sent to the target host [36].

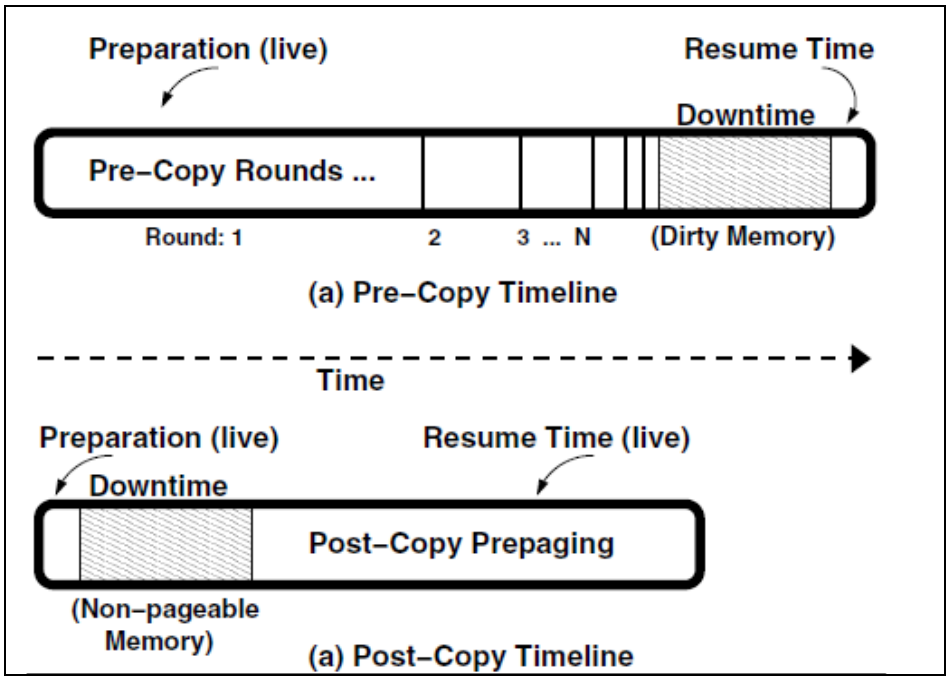


Figure 2-9: The timeline of (a) Pre-Copy and (b) Post-copy migration [36]**.**

The post-copy algorithm has the following basic stages:

a) The migrating VM is first suspended so as to copy the executing or processor state to the target host.

b) The VM is then activated on the target host.

c) Memory pages are then copied over the network from the source to the target host as required.

In Fig. 2-9 a timeline comparison is drawn to give a clearer picture of the difference between the pre-copy and post-copy algorithm. The downtime in the post-copy algorithm as shown occurs briefly as the initial processor state is migrated to the target. Thereafter the post-copying or memory fetching of the dirty pages is done over the network between the source and the target.

There are different enhanced methods that have been used to perform the fetching of dirty memory pages from the source to the target:

- Demand-paging
- Active pushing
- Pre-paging

2.5.1. Demand-Paging

The demand paging is considered as the slowest and simplest method. Once the VM is resumed at the target host after a brief copying of the essential processing state, its memory accesses result in page faults that can be serviced by requesting the referenced page over the network from the source node. The drawback of this lies in the networking overhead due to the network latency which affects the performance of the VM. Even if each dirty page is transferred only once from the source to the target node, the method incurs a high post-copying time due to the dependency on the number of remaining dirty pages at the source which can last for an indeterminate duration.

2.5.2. Active Pushing

One way[36] to reduce the duration of dependency on the source node is to proactively "push" the VM's dirty pages from the source to the target even as the VM continues executing at the target. Any major faults incurred by the VM can be serviced concurrently over the network via demand paging. Active push avoids transferring pages that have already been faulted in by the target VM.

2.5.3. Pre-Paging

The goal [36] of post-copy via pre-paging is to anticipate the occurrence of major faults in advance and adapt the page pushing sequence to better reflect the VM's memory access pattern. While it is impossible to predict the VM's exact faulting behavior, the authors' approach works by using the faulting addresses as hints to estimate the spatial locality of the VM's memory access pattern.

The differences in operation of both the pre-copy and post-copy algorithm have been elaborated in this section. In this research work however, the focus will be on the pre-copy algorithm due to its adaptation by market-leading hypervisors as mentioned earlier in the

previous section. Furthermore, this research uses the Xen hypervisor that utilizes the pre-copy approach.

2.6. Summary

The introduction of what server virtualization is and its impact to service providers and end-users alike has been discussed at length in this chapter. Consolidation of servers through virtualization has led to improved hardware utilization and eventual cost saving on the part of the service provider. Through resource pooling, service providers are now able to harness previously redundant computing resources and offer high availability as well as load balancing. This is achieved through live migration.

The benefits of live migration and the merits it has in the form of maintaining SLAs have also been discussed. Live migration performance parameters such as downtime and migration time play a significant role in the quality of service a user experiences during the actual process. Therefore the management of it cannot be overlooked. This has spurred the need to better manage as well as optimize the live migration process. Related works in this regard have also been presented within the chapter.

After the extensive literature review discussed in chapter 2, the proposed design framework is presented in the next chapter as a contribution in optimizing the live migration process.

3. A Statistical Approach in the Pre-Copy Algorithm

The concept of live migration as well as its benefits has been discussed in the previous literature review chapter. As a service provider it is paramount that the live migration process is carried out as efficiently as possible in order to reap maximum benefit from a resource management and SLA point of view.

This work proposes a statistical dirty page tracking approach in the pre-copy algorithm to optimize the live migration process. In so doing it is envisaged that the hosted service performance during the live migration will be improved upon.

In this chapter, the design of a dirty page tracking scheme to be implemented in the pre-copy algorithm will be examined on a real time service being hosted in a multi-tenancy model scenario. First, a detailed formulation of the pre-copy model is made. The key parameters of the model that will facilitate in designing the improved pre-copy algorithm are identified. Following this, the assumptions about the design are stated and a case study description is given preceding the implementation chapter.

3.1. Pre-Copy Model

As has been previously discussed, the pre-copy algorithm is Xen's preferred memory migration technique. The phases elaborated in section 2.4 are illustrated diagrammatically in Fig. 3-1.

A formal definition of the pre-copy model is formulated by Jin *et al* [37].

The VM total memory is denoted by the parameter *M*. During the live migration the allotted bandwidth used is denoted by the parameter *B*. The dirty rate *p* at iteration *i*, is represented by the parameter p_i. To calculate the time t_i it takes to transfer memory in the i^{th} iteration, the dirty rate p_{i-1} of the previous iteration is needed.

This is illustrated in the following equation.

$$t_i = \frac{p_{i-1} \times t_{i-1}}{B} = \frac{M \times \prod_{k=1}^{i-1} p_k}{B^i} \tag{1}$$

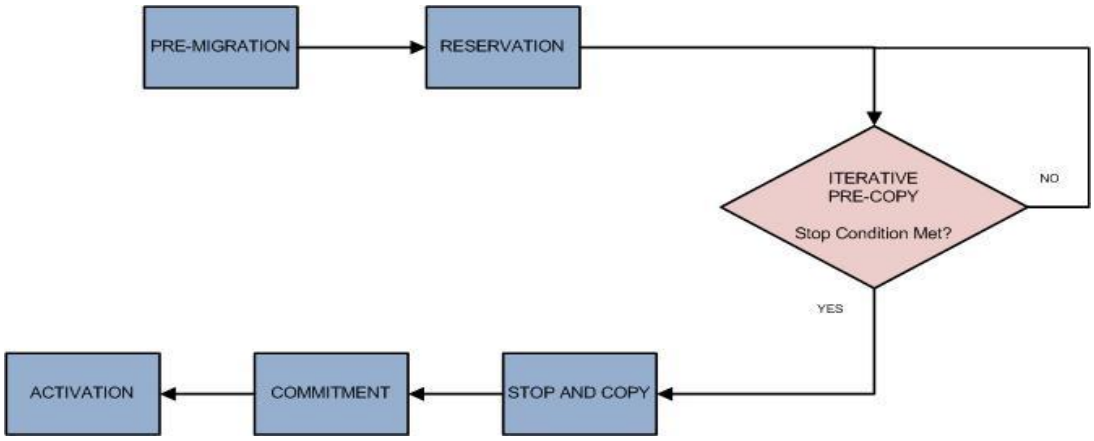


Figure 3-1: Pre-Copy algorithm flow diagram.

The maximum number of permissible iterations is denoted by *N* so as to stop the pre-copy iterative phase and proceed into the stop-and-copy phase. Another stop condition is a threshold value of dirtied memory to be transferred denoted by *h*. Then the total number of iterations, *n* is:

$$n = \min(\{i+1 \mid p_i \times t_i \leq h\}, N) \tag{2}$$

To get the total memory transfer time *T*, a sum of all the rounds is calculated as follows:

$$T = M \times \sum_{i=1}^{n} \frac{\prod_{k=1}^{i-1} p_k}{B^i} \tag{3}$$

According to Eq. (1), when $p_i < B$ the duration of each iteration will keep decreasing for every subsequent iteration. Conversely, when $p_i > B$ the time spent for the current iteration and subsequent iterations will increase.

As mentioned previously, downtime is the period from when the VM is suspended as the memory pages are being transferred from the source host to when it is restarted at the target host.

By considering both stop conditions of *N* and the threshold value *h*, the downtime t_d experienced to copy the remaining dirty memory pages can be denoted by:

$$t_d \leq \frac{h}{B} \tag{4}$$

where the number of iterations that have been carried out were within *N*-1. This target can only be achieved in a scenario where:

$$\frac{p_i}{B} \leq 1 \tag{5}$$

In the scenario where the ratio in Eq. (5) is greater than or equal to 1, any remaining dirty memory pages will be copied in the last iteration. With these dirty pages in the last iteration being more than *h*, then it implies an increase in the downtime. Worst case scenario is when the dirty rate is too fast for the assigned bandwidth to transfer and reduce the dirty pages at each iteration. This leads to the entire memory being transferred in the last round of copying. This is denoted by:

$$t_d = \frac{M}{B} \tag{6}$$

Assume that the dirty rate remains constant throughout the iterative copy. That is to say $p = p_1 = p_2 = p_3 ... p_i$. By ensuring that dirty rate p is less than assigned bandwidth B, means that the dirtied memory will keep decreasing until it meets the threshold value, h before the N^{th} iteration. Therefore the target p value that will ensure this can be obtained by the following equation.

$$p \leq \sqrt[N-1]{h/M} \times B \tag{7}$$

As the dirty rate increases beyond the target i.e. to say;

$$p > \sqrt[N-1]{h/M} \times B \tag{8}$$

The downtime begins to increase exponentially as shown in the following equation.

$$t_d = \frac{M}{B^N} \times p^{N-1} \tag{9}$$

With the Xen hypervisor, by default, the threshold h is set to 50 pages (≈200KB) and the maximum number of iterations, N is set to 30. These are the stop conditions that will trigger the end of the iterative pre-copy phase and move into the stop-and-copy phase.

As an illustration of the exponential increase in downtime t_d , we assume the following parameters:

M = 1024MB

B = 1000Mbps

h = 200KB

N = 30

To calculate the dirty rate at which the exponential increase of the downtime begins, a substitution of the above assumed parameters into Eq. (7) results in p = 744.9Mbps. Fig.3-2 illustrates this exponential phenomenon.

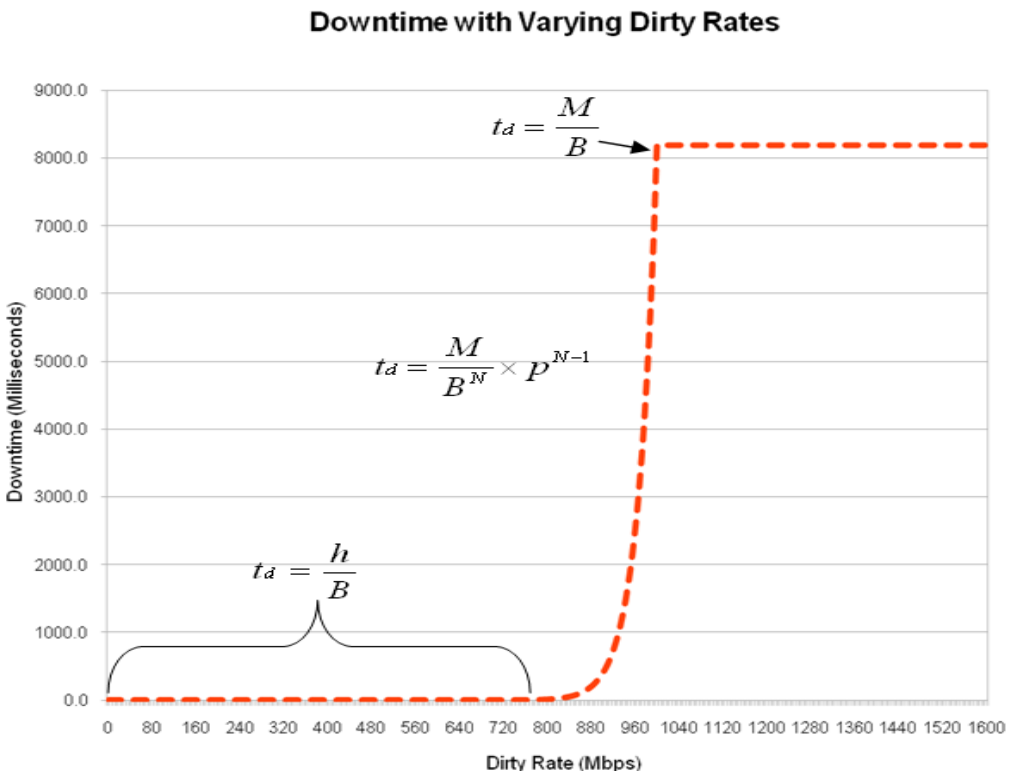


Figure 3-2: Downtime with varying dirty rate.

Akoush *et al.* in their work [25] describe the pre-copy model and its role in performance prediction during live migration. In their work, they describe the total migration time and downtime as:

$$\textit{Total Migration Time} = \textit{Pre-migration} + \textit{Reservation} + \textit{Iterative Pre-copy} + \textit{Stop-and-Copy} + \textit{Commitment} + \textit{Activation} \quad (10)$$

$$\textit{Downtime} = \textit{Stop-and-copy} + \textit{Commitment} + \textit{Activation} \quad (11)$$

Eq. (10) above is derived from summing up the time duration taken to complete all the phases described in the pre-copy algorithm as illustrated in section 2.5. Eq. (11) is a sum of only the last 3 phases of the pre-copy algorithm.

The authors also in the aforementioned work classify pre-migration and post-migration overheads as those incurred to initialize and mirror block devices belonging to the VM at the target host as well as reattach devices and advertise new IP addresses for the newly migrated VM. These are unavoidable overheads that are not part of the iterative memory copying during the live migration.

This research proposes that by considering the above observation then Eq. (4) is:

$$t_d \leq \frac{h}{B} + \beta \quad (12)$$

Where β are the pre-migration and post-migration overheads. This constant represents a more practical downtime which incorporates the unavoidable overheads that the live migration incurs.

3.2. Cause-Effect Relationship in Live Migration Performance

A summary of the parameters discussed above that affect the performance of live migration are summarized in the state diagram shown in Fig.3-3. These causes are further classified as external and internal causes. The former constitutes parameters which are fixed due to the nature of the VM RAM allocation, network bandwidth availability as well as the workload generation which can be interpreted as the dirty page rate. These are considered not

to be directly controlled by the hypervisor during the live migration process. Whereas the internal causes are considered to be within the control of the hypervisor.

The effects as illustrated give an indication of the live migration performance. These performance indicators are important when considering their impact on service quality and eventually SLAs.

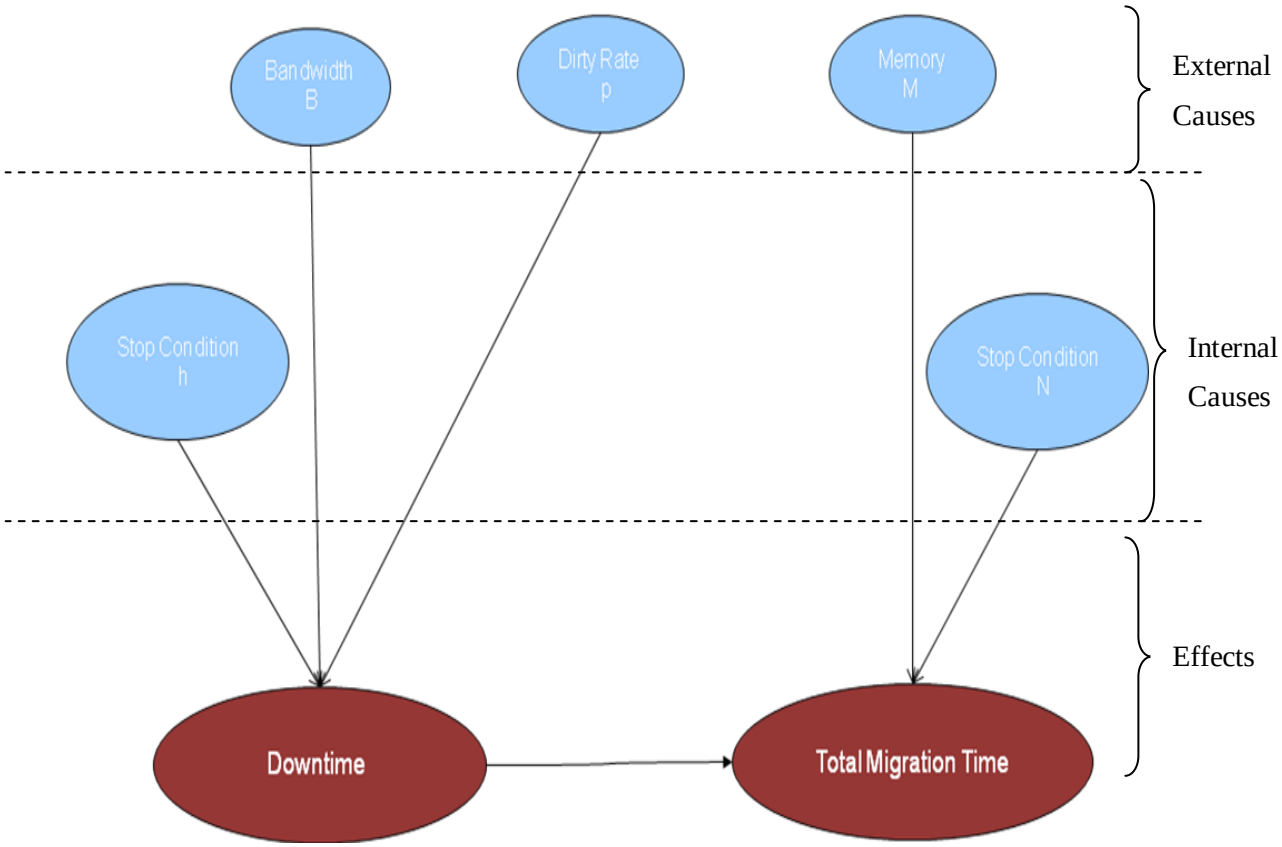


Figure 3-3: Cause-effect relationship in live migration performance.

3.3. Statistical Tracking of Dirty Pages

It is proposed in this design chapter to track the dirty pages per iteration during the live migration. This is done so as to extract statistical information that will be used to get an estimate of the dirty page standard deviation profile. With this information, it is then possible

to optimally set and stop the pre-copying activity during the live migration as compared to having fixed stop conditions.

3.3.1. Standard Deviation

In probability theory and statistics, standard deviation, (denoted by the symbol sigma, σ) is a measure of how much variation or spread there is from the mean in a given set of data. A low standard deviation indicates that the data is close to the mean. The converse would imply that the data is dispersed over a large range of value. This is illustrated in Eq. (13) below.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}} \tag{13}$$

Where:

- n denotes the number of observed values in the data set.
- x_i denotes the observed value in the data.
- $\bar{x}$ denotes the mean value of the data.

Keeping the above definition in mind, we can correlate the number of observed values to represent the dirty pages for every i^{th} iteration . The number of observed value in this case is the number of iterations carried out.

It is envisioned that a profile of the standard deviation across n iterations can be produced by calculating the standard deviation at each iteration. Fig. 3-4 is an illustrative example of the dirty pages generated per iteration and the corresponding standard deviation profile. This profile will be able to paint a picture of the dirty page variation during the iterative pre-copying phase. A relatively constant profile implies that the dirty pages generated per iteration is also relatively constant. This research then proposes that if this is the case, then the iterative process may be terminated prematurely in an effort to optimize the pre-copy algorithm.

It is envisioned therefore that an early detection of a constant standard deviation profile can be used to terminate the iterative copy phase of the pre-copy algorithm. This is primarily because further iterations will not be reducing the dirty pages any further at the source which is the aim of the original pre-copy algorithm. The benefit of this is that fewer iterations and hence less time is spent on the iterative copying phase and hence a reduction of the total migration time is observed. This is also illustrated by Eq. (3) above where the number of iterations, *n*, is reduced.

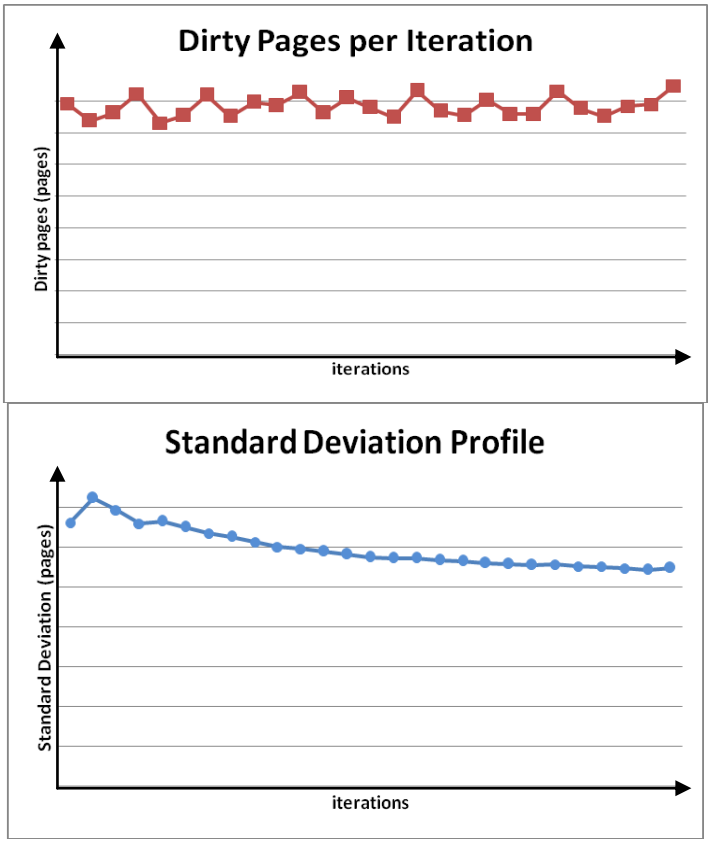


Figure 3-4: Correlation between Dirty pages and Standard Deviation Profile for n iterations.

With the above proposed use of the standard deviation, then Eq. (13) parameters are interpreted as follows:

- n denotes the predetermined number of iterations that will constitute the entire set of observed dirty pages from which the standard deviation profile will be calculated.
- x_i denotes the dirty pages observed at iteration i.

- $\bar{X}$ denotes the mean value of the observed dirty pages.

Standard deviation has been selected over the variance mainly because the units are the same as the raw data; which is the number of dirty pages generated.

3.3.2. CPU Credit Scheduling in Xen

As has been proposed in the related works [37], [33], the CPU resource plays an important role in the dirty page rate generation during the live migration process. In section 3.1, the pre-copy model shows that the dirty page rate is a factor in the downtime and migration time that is experienced by the VM. It was noted, through the authors' experience in [37] , that there was a linear relationship between the CPU and the workload (dirty page) generation. This was shown in the authors' work, particularly, in the mechanism design section that mathematically expressed this linear relationship.

The CPU scheduling has a direct impact on the dirty page rate. This in turn affects the number of dirty pages generated per iteration. The CPU scheduling reduces the amount of time the VCPU gets on the PCPU and this reduces the number of dirty pages generated in comparison to when there is no cap set. The stop conditions (h and N) are then met and this terminates the iterative process and proceeds to the stop-and-copy phase.

By reducing the time the VCPU gets on the PCPU, the dirty page rate generation can be managed and improve the performance of the live migration in terms of decreasing the downtime. This however will have an impact on the application or service running in the VM.

As described in section 2.2.2, Xen by default uses a credit scheduler. In this research, it is proposed to assess the impact of varying the credit scheduler's cap which determines the percentage of CPU time the VCPU gets. In so doing, it is envisioned that the downtime will be reduced significantly.

3.4. Dirty Page Standard Deviation Tracking with CPU Scheduling

The statistical tracking scheme proposed in combination with CPU scheduling discussed in the preceding section aims to optimize the live migration process. On one hand

the credit scheduler cap will be reducing the downtime experienced by the migrating VM, while on the other the standard deviation tracking will aim to limit the amount of iterations thereby reducing the migration time.

Fig.3-5 below illustrates how the standard deviation tracking will be used to select and assess the credit scheduler cap and thereafter the effect this will have on live migration performance in terms of downtime and migration time.

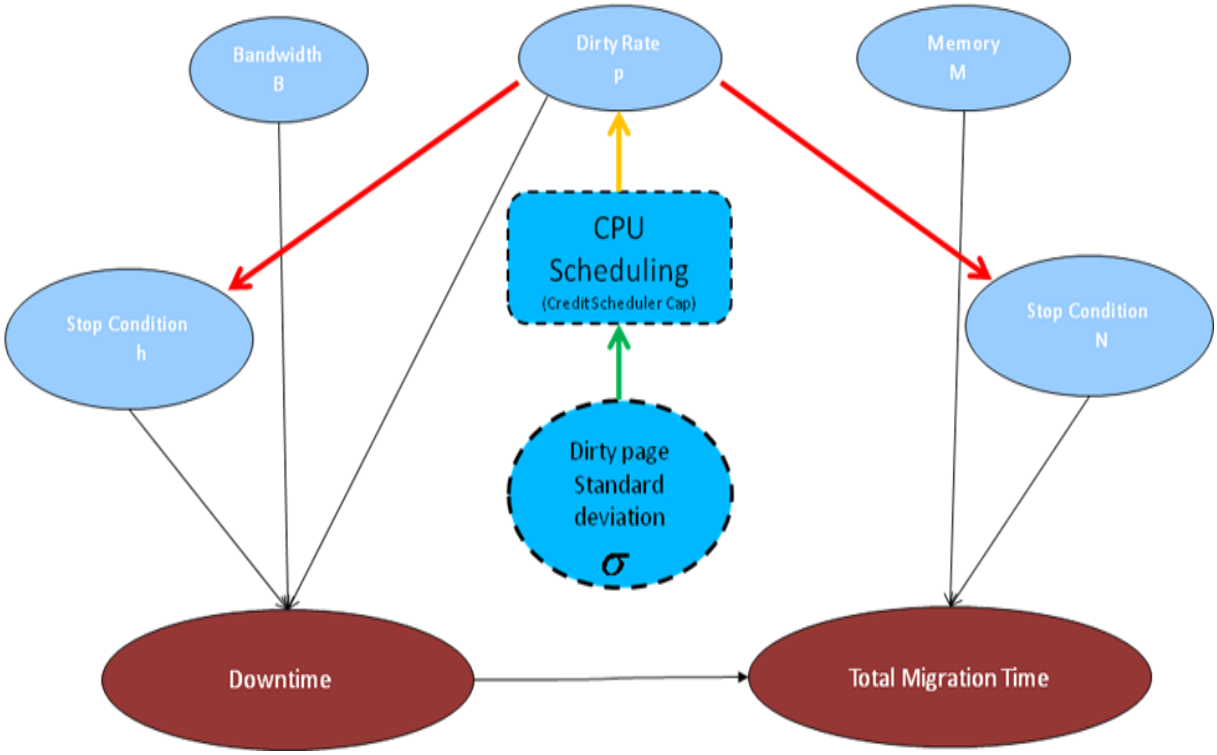


Figure 3-5: Cause-effect relationship showing standard deviation tracking with credit scheduler cap.

The following procedure will be used to achieve this goal:

1. A dirty page standard deviation profile for a range of credit scheduler caps is obtained and recorded after performing a series of live migration experiments.

2. From the above exercise, a cap is selected based on the dirty page standard deviation profile that is a suitable candidate for standard deviation tracking. A suitable profile for standard deviation tracking will be based on the stability of the standard deviation as well as the number of iterations that have been carried out.

3. To optimize the live migration process, the live migration has to be stopped once it detects that the stability point mentioned in the second point has been achieved.

4. To observe the improvement, service-specific QoS parameters will be recorded for the optimized pre-copy algorithm. This will then be compared with the default pre-copy algorithm.

3.5. Design Assumptions

The design assumptions that are made in proposing the following dirty page tracking approach are:

- The standard deviation calculations will not include the first iteration. This is because the first iteration copies the entire RAM and the actual pre-copying activity begins from the second iteration. Therefore, dirty pages that will be observed will be from the 2^{nd} iteration up to and including the predetermined n^{th} iteration.

- The predetermined number of iterations *n* that will constitute the data set used to compute the standard deviation will be greater than 10. This is so as to have a large enough data set and hence more confidence in the stability of the computed sigma value.

- The standard deviation range of values will be obtained after a series of live migration experiments. This will be used to determine the permissible standard deviation values that will trigger the end of the iterative pre-copy phase in the live migration.

- The stability criterion of the standard deviation will only be assessed over the maximum number of iterations which is 29 in this case. Any credit scheduler cap with a dirty page standard deviation profile that has less than 29 iterations is not considered as a viable candidate for tracking.

3.6. Modification of the Pre-Copy Algorithm

In order to implement the design, the open source virtualization platform, Xen will be installed from source. This installation process is discussed at length in Appendix A. The file responsible for the migration of dirty memory pages is *xc_domain_save.c* and is located in the *tools/xlibc/* directory of the unpacked source code.

With access to this file, the proposed standard deviation tracking design can be coded into the *xc_domain_save.c*. Once this is done it will be recompiled into the Xen source code and installed on the source host.

Fig.3-6 below illustrates how the standard deviation tracking design proposed will be incorporated into the original pre-copy algorithm during its execution. This in turn will then have an effect on the live migration performance through the stop conditions as has been proposed in the formulation of the tracking concept.

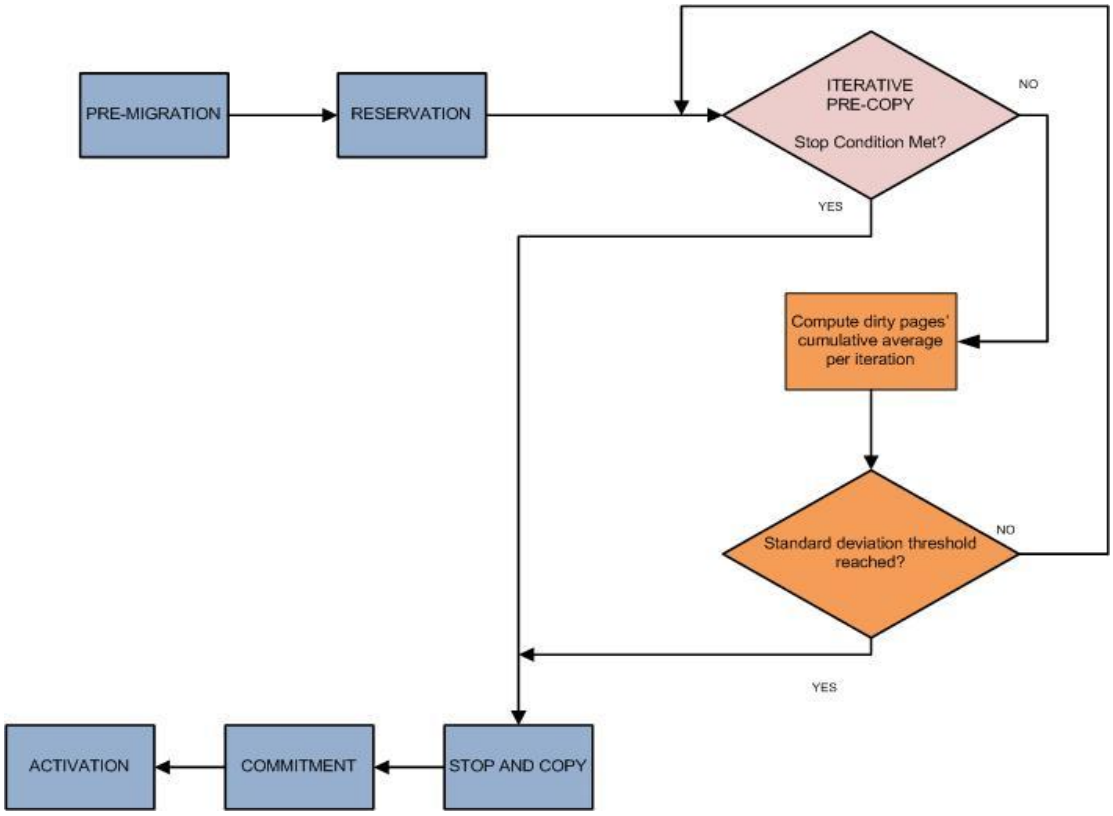


Figure 3-6: Standard deviation tracking in the modified pre-copy algorithm.

Since the modification of the original pre-copy algorithm may affect the overall efficiency of the initial code's execution, the changes proposed in this design will not be complex or highly demanding in terms of computational capacity.

3.7. Aggregated Service Consolidation Case Study Scenario

As a means of demonstrating the proposed design in this chapter, this research will evaluate real-world applications and carry out live migration experiments. As a case study, this research will consider the consolidation of both an email service and VoIP service onto

the same physical host. This will give a real-world scenario of a service provider offering multi-tenancy hosting of both real time and non-real time services within their data centre.

In consideration of the impact that live migration will have between the two classes of service stated above, this research will focus its attention on the effect and performance of the VoIP service while it is being live migrated. This primarily is because disruptions to these types of services are more easily perceived by end users. Moreover, the popularity of real-time services amongst everyday users currently makes this type of service a more worthwhile choice for impact assessment in live migration.

In summary, the VoIP service will be the focus of assessment while the email service will only be present to create resource contention on the host during the live migration from the source to the target host.

3.8. Design Topology for Live Migration

Given that live migration is between at least two hosts, connected on the same LAN as stipulated in the scope and limitations section, the topology proposed in this research will constitute no more than two physical hosts.

For the design topology, the Xen hypervisor of the source host will be modified to demonstrate the standard deviation tracking scheme as shown in Fig. 3-5. The hypervisor of the target host will remain unmodified. The VoIP and email service will both be hosted on the source host while the target will not be running any VMs.

A client will interrogate these services and thereby create a workload on both services. This workload will translate into the generation of dirty pages during the live migration. The intensity of the workload generated will then give rise to a spectrum of live migration performance metrics which then can be analyzed and assessed accordingly. And as has been mentioned, the focus will only be on the live migration of the VoIP service.

Fig.3-7 below illustrates the design overview incorporating the ideas discussed in this section.

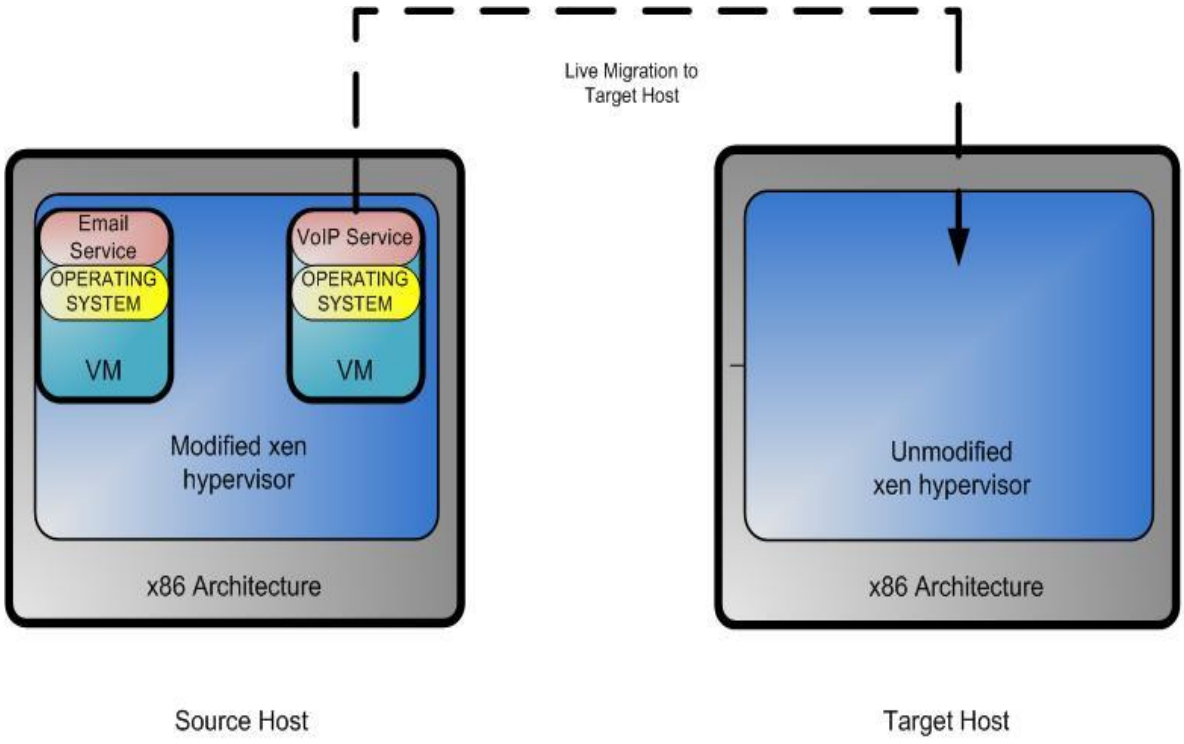


Figure 3-7: Design overview of live migration topology.

3.9. Summary

In this chapter, the correlation between key causes and their effects in terms of live migration performance have been drawn from a detailed pre-copy model. From this model, the rate at which dirty pages are generated plays a significant role in the downtime experienced during the live migration. From a service provider's point of view, the minimizing of the downtime is beneficial especially when considering the maintenance of SLA's with the end user.

The use of CPU scheduling is proposed to reduce the downtime of the migrating VM and intrinsically the service being migrated. This is achieved through setting a cap on the amount of time the VM gets on the CPU. A statistical approach is proposed in conjunction with the CPU scheduling to track the dirty pages generated per iteration and terminate once it is noted that further iterations will not reduce the dirty pages at the source host.

VoIP as a real-time service has been picked as a candidate to observe and quantify the improvements in service-specific parameters during the live migration before and after the proposed design.

In the next chapter a detailed evaluation platform is described to demonstrate the implementation of the design concepts discussed herein.

4. Testbed Implementation on Xen

In chapter 3, the design of a statistical approach to optimize the live migration of a service running within a VM was proposed. This chapter deals with the implementation of this concept. Furthermore, the chapter discusses the evaluation framework that is based on a practical testbed to get the sense of a more realistic performance of the proposed design.

This chapter begins by detailing the hardware and software tools used in the development of the evaluation framework to analyze the proposed design. This is followed by a look at the design framework integration into the testbed and finally a procedure of how the improved performance will be assessed.

4.1. Implementation Overview

The implementation of the proposed framework is carried out on open source software, which is licensed under the GNU General Public License and offers the choice of customising the architecture to suit the needs of the proposed design.

The test environment as shown in Fig. 4-1 below consists of 4 nodes.

2 dual-core Intel machines labeled source host and target host in which the VMs running the services are hosted. As stated in section 3.8, the source host contains both services from where the VoIP service is then migrated to the target host.

A client makes service requests to the hosted services and generates the workload towards these services. The workload generated then determines the live migration performance as it has a direct relationship with the dirty page generation.

All nodes are on the same /24 subnet and are interlinked with a gigabit Ethernet switch. Appendix B contains further information on how the VMs' networking was carried out. The high bandwidth demands for the live migration process require that the traffic be isolated to avoid negatively affecting other machines within the research lab's LAN. Also, this is done to ensure that there are no external network factors, such as congestion or network faults, that will impact the performance evaluation of the proposed design.

The iSCSI storage stores the VM's filesystem as a file image and offers them to both hosts. This is important for the live migration as it ensures a degree of robustness to a single

point of failure in comparison to having the filesystem on one of the hosts. Refer to Appendix C for further information pertaining to the setup.

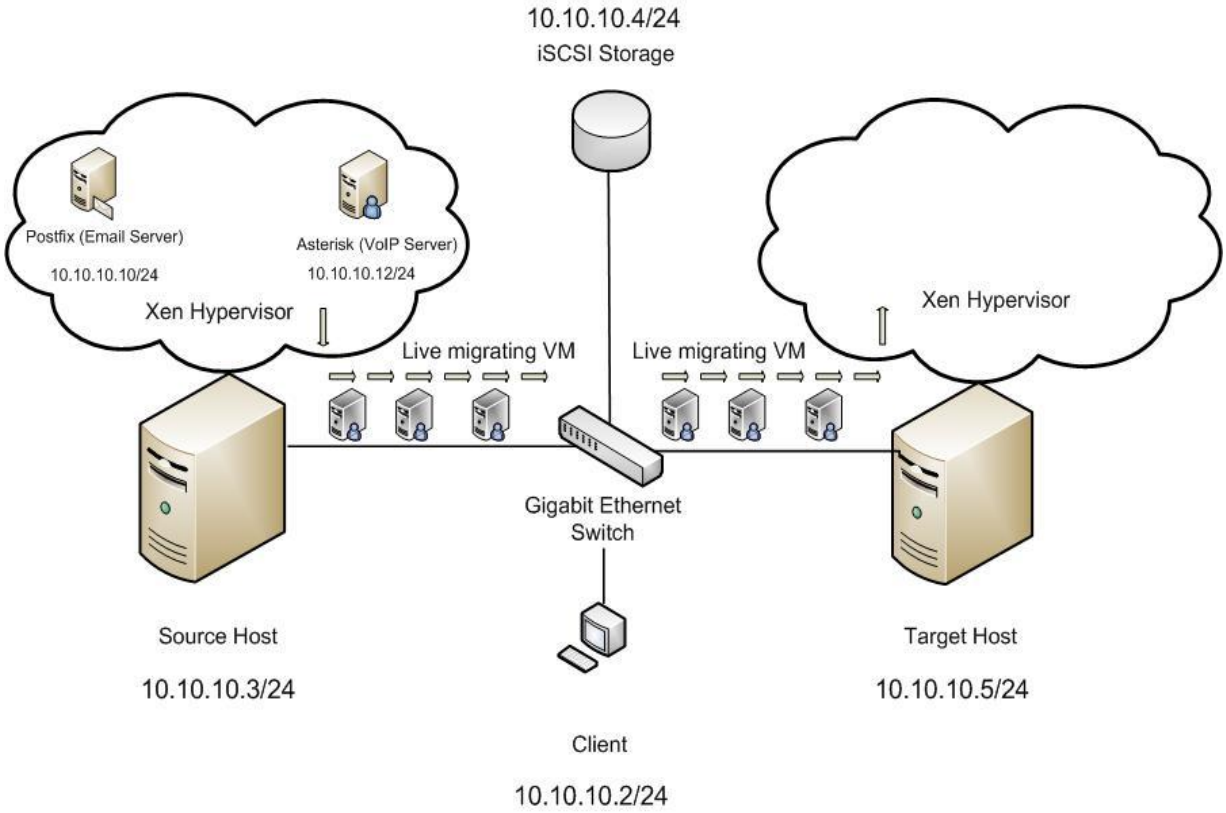


Figure 4-1: Testbed Implementation on a Xen hypervisor

4.2. Xen Hypervisor

The Xen kernel that is used in this testbed is version 3.4.3 and has been installed from source packages. The virtual machines or domUs are created through Xen's virtual machine manager application, known as virt-manager. During the VM creation process, the resource allocation of the virtual CPU, RAM, storage and networking parameters are setup.

4.2.1. virt-manager

Virt-manager is the administrative point from where the VMs are monitored and managed by the hypervisor. It is installed in the dom0 and therefore has privileged access to the underlying resources unlike the domUs hosting the services. The version that is installed in the testbed is 7.0.

The virtual machine creation is a step-by-step process whereby virt-manager requests for the following information:

- The unique name of the VM to be created.
- The location of the OS installation repository to be installed on the VM.
- The type and location of the VM's filesystem storage.
- To install the VM as paravirtualized or fully virtualized.
- Resource allocation (CPU and Memory) and networking setup.

Fig. 4.2 – 4.6 shows screenshots of the aforementioned steps.

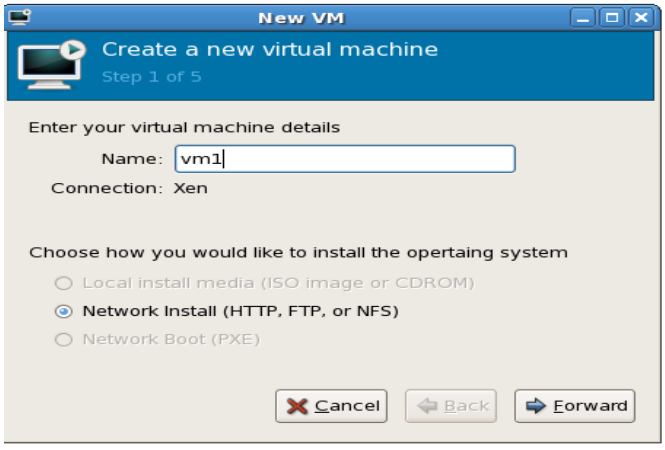


Figure 4-2: VM name creation using virt-manager.

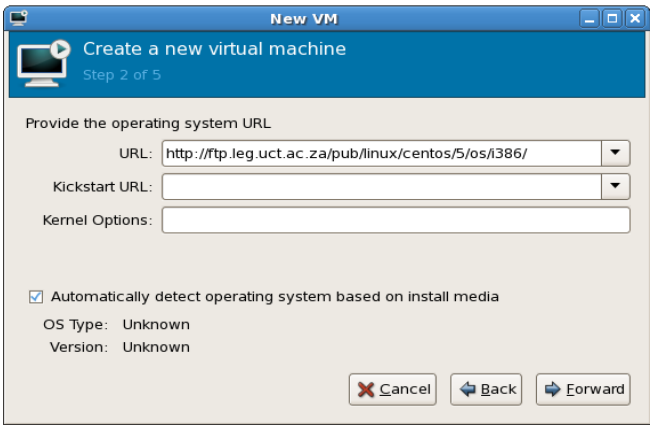


Figure 4-3: OS installation via network install option.

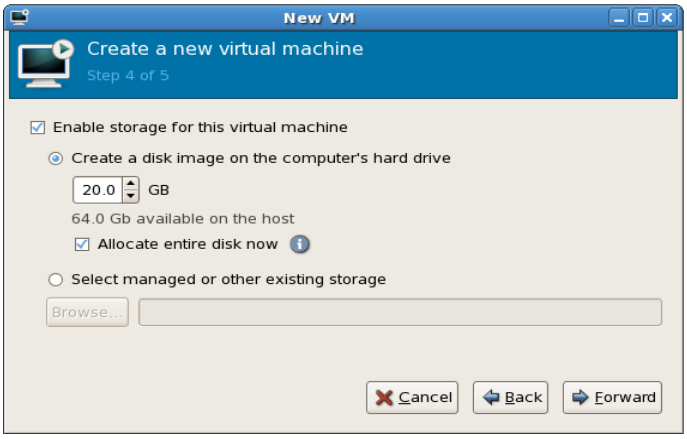


Figure 4-4: Storage allocation for VM filesystem.

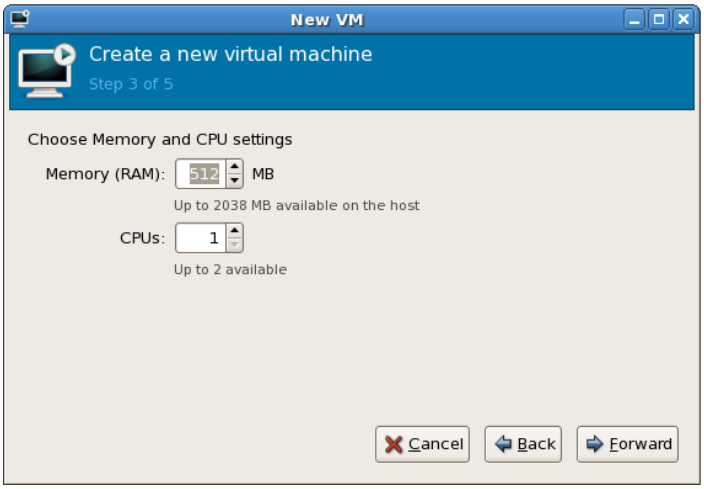


Figure 4-5: CPU and RAM Allocation.

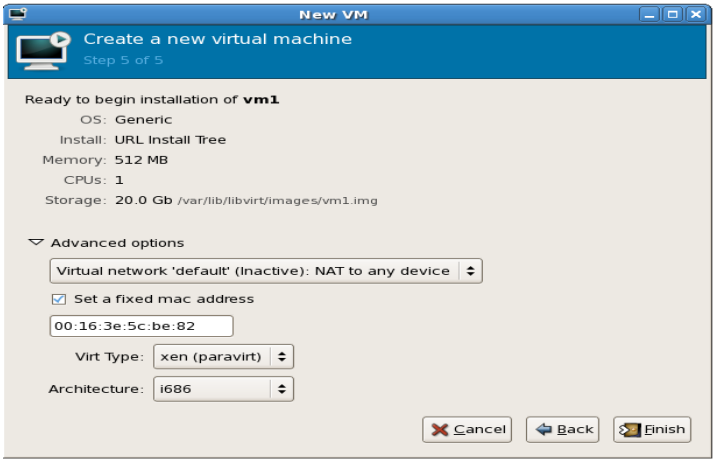


Figure 4-6: Networking setup and Type of Virtualization (para or full).

4.2.2. Live Migration Setup

For live migration to take place between two hosts, the following requirements must be met:

- Both the source and target hosts must be able to access the shared storage containing the VM's file system, which, in this case, is an image file located on an iSCSI storage.
- Hosts should be running compatible CPUs.
- There should be sufficient computing resources at the target host to accommodate the VM.
- Both the source and target must be on the same LAN.
- The versions of Xen running on both hosts must be compatible.

By default the live migration operation is disabled for security reasons. To enable live migration, the Xen control daemon, *xend* must be setup to listen for migration requests on both the source and target host by setting the relocation port to 8002. This operation is performed by editing the xend-config file located at */etc/xend-config.sxp* on both hosts. Below is an excerpt of the file highlighting the changes that need to be made.

```
# Port xend should use for the relocation interface, if xend-relocation-server
# is set.
#####Uncomment to listen for migration requests
(xend-relocation-port 8002)
```

Furthermore, the hosts that are permitted for live migration requests may also be specified as shown in the excerpt below.

```
(xend-relocation-address '')

# The hosts allowed to talk to the relocation port.  If this is empty (the
# default), then all connections are allowed (assuming that the connection
```

Since the testbed setup is on an isolated LAN from the research lab's network and only contains two hosts, it is opted to set it to listen to all hosts for requests.

4.2.3. Xen Management User Interface (xm)

The *xm* program is the main interface for managing Xen guest domains from the command line and may also be used as an alternative to the virt-manager for creating, pausing as well as shutting down VMs. All *xm* commands are executed with root privilege due to the fact that these are the communications channels used to talk to the hypervisor.

The *xm* command which is pertinent to this research is the *xm migrate* command which is used to perform the live migration.

```
#[root@phillip ~] # xm migrate –l 4 10.10.10.5
```

The - *l* denotes that it is a live migration and the 4 denotes the domU ID that is being migrated to the host with the IP address 10.10.10.5.

Other commonly used *xm* commands include:

- *xm-list:* which lists all domUs and their current status.
- *xm sched-credit:* which sets the cap and weight of the CPU credit scheduler.

4.3. Hosts

The source and target host are both x86 dual-core Intel machines with CPU 2.80GHz and 2.70GHz respectively. Both hosts have RAM (DDR2) of 2GB and have Realtek PCI Express Gigabit Ethernet controllers.

The dom0 kernel running on both hosts is CentOS 5.8, which is an RPM-based Linux distribution derived from Red Hat Enterprise Linux. The page size, as mentioned, depends on the computer architecture deployed. In this testbed, the value was obtained by typing the *getconf* command in the host's terminal.

```
# getconf PAGESIZE
```

This returns the value 4096 which is measured in bytes.

4.4. Hosted Services

A real-time and non-real time service in the form of VoIP and email service are setup within the VMs on the source host.

4.4.1. VoIP Service (Asterisk)

Asterisk is an open source IP telephony software which may be used to build a host of communication applications such as IP PBX, voice gateways and IVR(Interactive Voice Response) [38]. The source host is running a domU with assigned IP address 10.10.10.12 and a Linux distribution, CentOS 5.8. The Asterisk version installed is 1.6.2.20.

Asterisk also has a user friendly GUI, FreePBX, that is used for management and control. Fig. 4-7 below shows three extensions that have been created using the GUI. The three extensions/users' numbers are 2000, 2001 and 2002 with display names as John Doe, Jane Doe and sipp respectively.

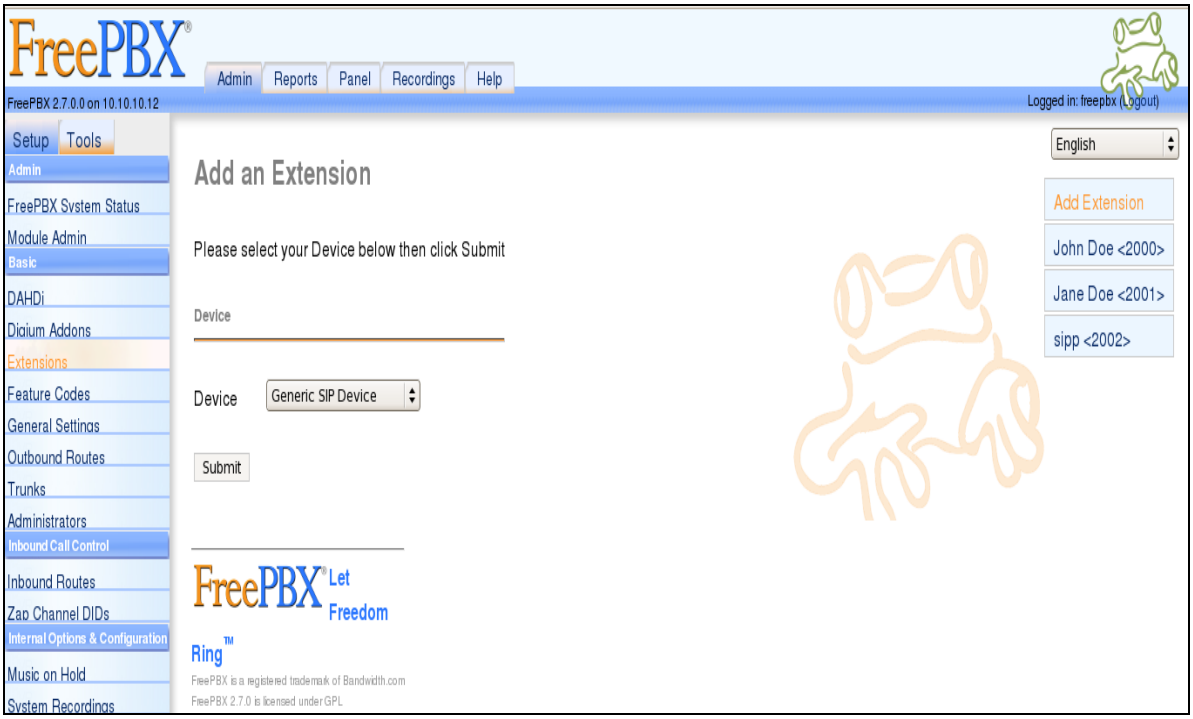


Figure 4-7: FreePBX showing extensions available.

4.4.2. VoIP Client

To interrogate the VoIP service, a VoIP client installed at the client machine is used. CentOS installation includes an IP telephony client application called Ekiga. The extensions

2000 and 2001 are setup to emulate a call session between two clients as shown in Fig. 4-8 below.

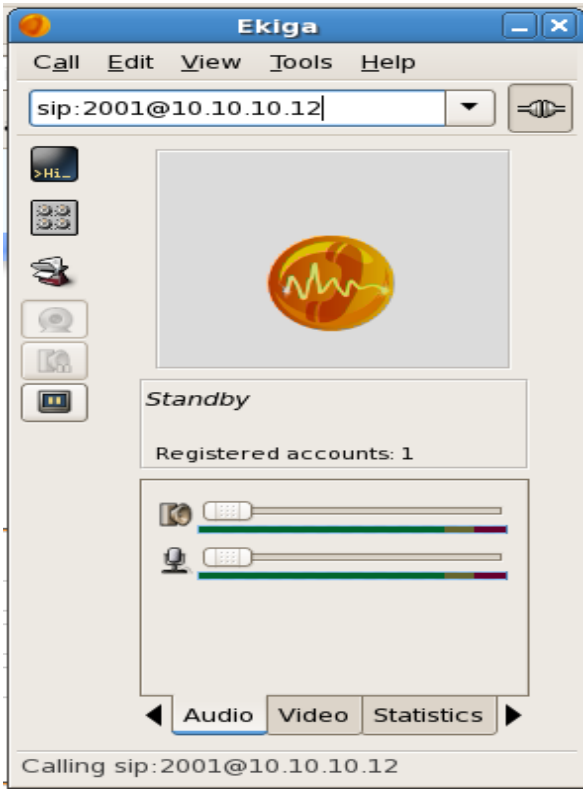


Figure 4-8: Ekiga showing a call session between extension 2000 and 2001.

The above illustration shows extension 2000 belonging to user John Doe calling and connecting to another client with extension 2001 belonging to Jane Doe. Jane's extension is registered to the Asterisk machine with IP address 10.10.10.12 denoted by sip:2001@10.10.10.12.

4.4.3. Email Service (Postfix)

Postfix [39] is an open-source email server that can be used to route and deliver electronic messages. The source host is also running domU with CentOS 5.8 and Postfix version 2.3.3 is installed. To be able to route electronic messages through the server and deliver them across the network, the Postfix configuration file */etc/postfix/main.cf* is edited by adding mailboxes, domain names and permissible clients' IP addresses that may route their electronic messages through the server.

The domain name setup in Postfix is *xen.com*. This is done by setting the "mydomain" parameter in the configuration file to be xen.com. This is illustrated in the following excerpt of the configuration file.

```
# The mydomain parameter specifies the local internet domain name.
# The default is to use $myhostname minus the first component.
# $mydomain is used as a default value for many other configuratio
# parameters.
#
mydomain = xen.com
```

Two email addresses are setup; jane@xen.com and john@xen.com from where emails will be sent and received respectively. Fig. 4-9 below shows the mailbox of john accessed using an email client application, Evolution.

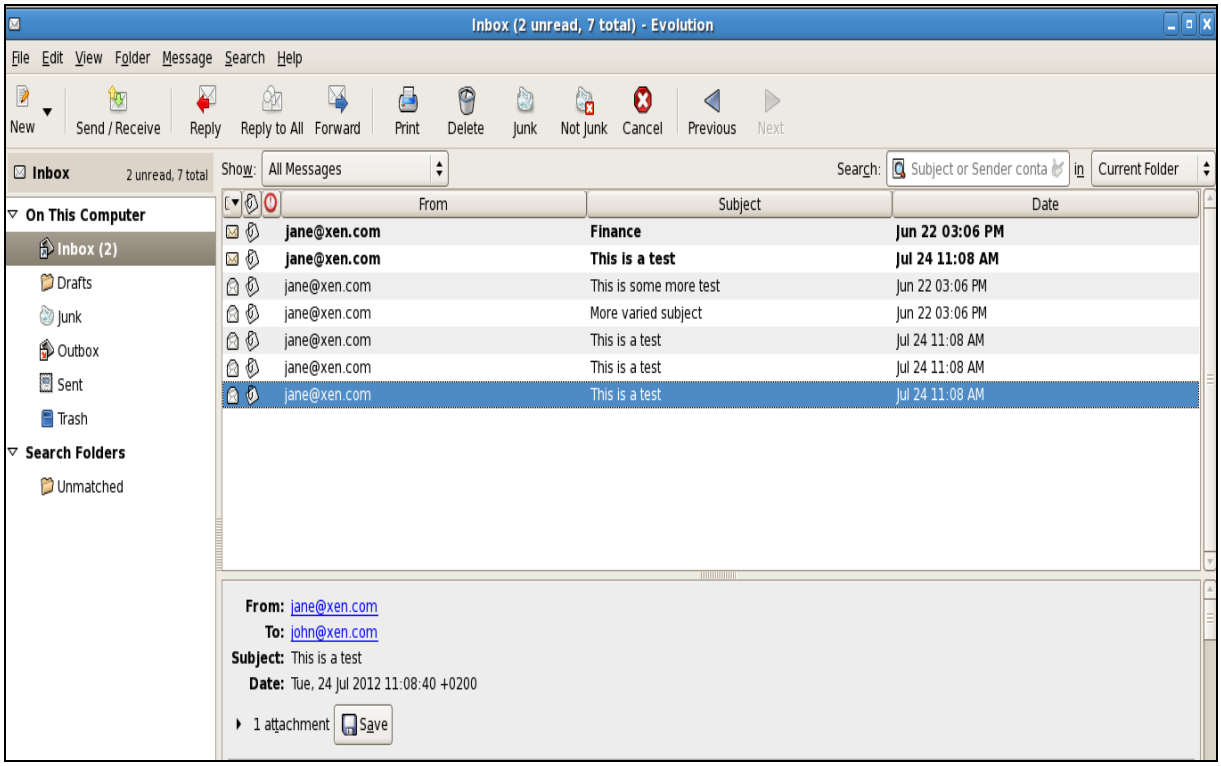


Figure 4-9: Inbox showing client's emails.

4.5. Traffic Generation Tools

To create the desired workload on the services above, traffic generators are installed at the client to interrogate the services and thereby create dirty page activity during the live migration process.

4.5.1. SIPp

SIPp is a free open source performance test tool for the Session Initiation Protocol (SIP) which is primarily used for communication session control for voice and video calls. It is used to generate signaling plane traffic. SIPp can also be used with limited capabilities to generate Real-Time Transport Protocol (RTP) traffic. [40].

SIPp contains embedded scenarios to suit different test requirements. The user agent client (UAC) scenario establishes and releases multiple calls with the INVITE and BYE methods [40]. This UAC scenario is used to generate signaling traffic to the Asterisk VM. SIPp is controlled through command line instructions to generate the traffic.

Below is an excerpt showing the UAC scenario being used in SIPp to generate signaling traffic to extension 2002 which is registered to Asterisk server at IP address 10.10.10.12.

```
# [root@localhost sip.svn]# ./sipp -sn uac -d 20000 -s 2002 10.10.10.12 -m 1000
```

A delay variable, measured in milliseconds and denoted by the option '-d', is used to control the call length which in turn creates a workload on the Asterisk application. Once a maximum of 1000 calls is reached, the traffic generator begins to gracefully terminate the calls to the extension. SIPp also collects other relevant statistics such as response time and call length distribution. These are further discussed and illustrated in Appendix D.

4.5.2. xstress

Xstress [41] is a free open source SMTP traffic generator that can be used to stress an email server by creating multiple connections to and sending multiple mails repeatedly over these connections to the mail server. Further customization of this tool includes the ability to send mails with varying attachments that may be binary files, plain text, html sources and images.

The variables used in xstress are:

- Server – Specify the SMTP mail server IP address.

- Port – The port that will be used by the SMTP server. By default it is 25.
- Threads – Specify the number of connections to the mail server.
- Mails per thread – Specify the number of mails to be sent per thread.
- Timeout –Specify connection timeout in seconds after which xstress will give up.

A log is kept as well showing timestamps of the sent mail and also a counter showing the number of mails that were successfully sent and how many failed.

4.6. Performance Monitoring

To assess the impact of the live migration on the VoIP service, monitoring tools are installed on the testbed to observe service-specific parameters. These are installed at the client machine as this is the point from where the VoIP client is running.

4.6.1. Wireshark

Wireshark [42] is an open source network packet analyzer that captures network packets and displays the data content of the packet in great detail. This tool is used by system administrators for troubleshooting issues on the network, debugging protocol implementations and examining for security problems among others. Fig. 4-10 is a screenshot example showing the network packet details that are collected during a capture; the packet source, destination as well as packet length are observed in real-time. This capture may also be saved into a file for future analysis.

With this said, Wireshark will be used to monitor the SIP and RTP packets sent during an audio call session between two extensions, 2000(John) and 2001 (Jane) to examine the packets and analyze them during the live migration. QoS parameters such as jitter are obtained from this analysis and demonstrate the effect of the live migration on the service quality of the voice call.

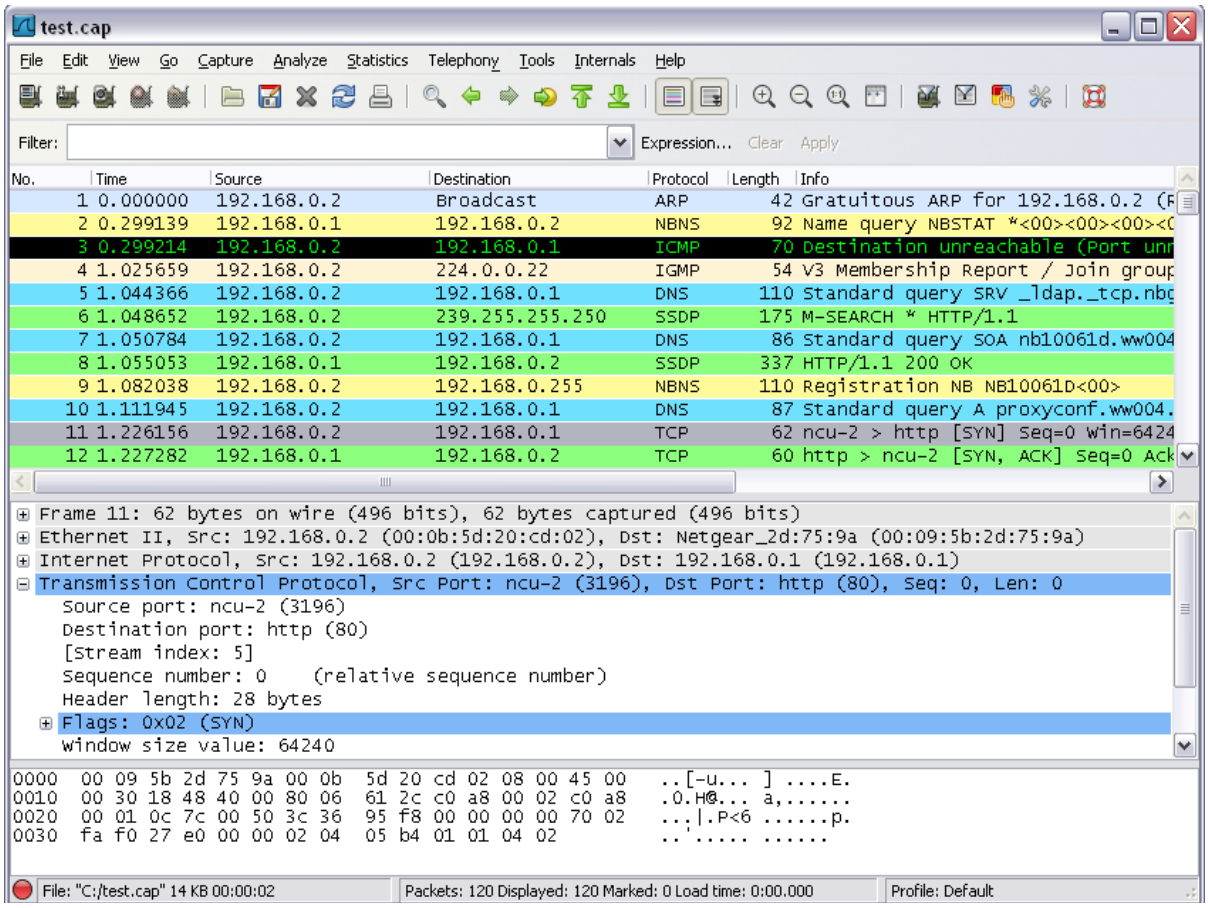


Figure 4-10: Wireshark showing captured packets [42].

4.6.2. Ping

As stated earlier, during the live migration process, a brief outage or downtime occurs while the pre-copy algorithm suspends the VM at the source and copies the remaining dirty pages and CPU state to the target host. The service at this point is not reachable and this is measured using a simple ping command during the live migration.

The ping test measures the amount of round trip time it takes for ICMP echo request packets sent from the client to the Asterisk VM. The downtime is considered to be proportional to the number of pings to the Asterisk VM which are unreachable during the live migration. Since we can also control the interval at which these pings are sent then the downtime is:

$$Downtime = No.\ of\ Unreachable\ Pings \times Ping\ Interval \quad (14)$$

The excerpt below is the ping command used in this testbed during the live migration of the real-time service.

```
#[root@phillip ~] # ping 10.10.10.12 –i 0.05
```

The Asterisk VM has IP address 10.10.10.12. The *-i* option is the ping interval in seconds between sending each ICMP echo request packet. This is set to 50 ms to get better granularity in the data collected, thereby ensuring a more accurate measure of the downtime when it occurs. Also it should be noted that by default the interval is 1 second and only the root user has the privilege of setting this value.

4.6.3. VoIPmonitor

VoIPmonitor [43] is a C++ program designed to monitor and analyse thousands of simultaneous calls by sniffing network packets for SIP and RTP VoIP protocol. The main purpose of this program is to analyze the quality of SIP VoIP calls based on qualitative parameters such as packet delay variation and packet loss according to ITU-T G.107 E-model which predicts quality on a Mean Opinion Score (MOS) scale. These parameters will be discussed at length in the following sub-section.

VoIPmonitor comes in two versions; an open source packet sniffer and a commercial packet sniffer which includes a Web GUI and support for more codecs. The open source version is selected for this research solely for the purpose of cost-saving.

There is similarity in the mode of operation between VoIPmonitor and Wireshark in that they can both sniff out SIP and RTP packets for analysis. The reason for including the VoIPmonitor as an added monitoring tool is that among other VoIP-related metrics, it also gives a MOS score which is important but is not included in the analysis provided by Wireshark.

To run VoIPmonitor the configuration file, */etc/voipmonitor.conf* needs to be called from the command line as illustrated in this excerpt.

```
# [root@phillip~]# voipmonitor –config-file /etc/voipmonitor.conf –k –v 1
```

This will immediately start monitoring the Ethernet interface every second for SIP and RTP packets in real-time and finally send the collected data to a MYSQL database. The database also has the option to export to Microsoft Excel as a comma separated value (CSV) file for further analysis.

The three relevant parameters for this research that will be collected by VoIPmonitor for analysis are:

- Packet loss
- Packet delay variation
- MOS

4.6.3.1. Packet Loss

According to the manual [43], VoIPmonitor detects packet loss and stores loss distribution to 10 loss intervals so it is able to find larger consecutive losses. This is useful to identify packet losses that are consecutively occurring and are perceived by the user more than those that are spread out randomly during the duration of the call.

4.6.3.2. Packet Delay Variation

In computer networking [43], packet delay variation (PDV) is the difference in end-to-end one-way delay between selected packets in a flow with any lost packets being ignored. The term PDV is defined in ITU-T Recommendation Y.1540, Internet protocol data communication service - IP packet transfer and availability performance parameters, section 6.2. This is also referred to generally as jitter, but by definition, it is imprecise for packet-switched networks, hence the adoption of using the PDV as a quality of service measure.

VoIPmonitor compares each RTP packet if the delay differs to optimal value (for most cases the delay between two RTP packets are 20ms). If the delay is higher than 50ms it will be counted to one of PDV intervals. The PDV intervals are - 50 – 70ms, 70 – 90ms, 90 – 120ms, 120 – 150ms, 150-200ms ,300ms and above.

4.6.3.3. MOS (Mean Opinion Score)

This is a subjective measure by a human listener to score the quality of a call as per ITU-T Recommendation P.800. This score is measured on a scale from 1 to 5; where 1 is the worst and 5 is the best. This is elaborated further as shown on Table 1 below.

Table 1: MOS rating scheme.

MOS	Quality	Impairment
5	Excellent	Imperceptible
4	Good	Perceptible but not annoying
3	Fair	Slightly annoying
2	Poor	Annoying
1	Bad	Very annoying

VoIPmonitor transforms PDV and packet loss into a MOS score according to ITU-T E-model which means that the MOS does not represent audio signal but network parameters. Because relation of PDV and MOS score depends on jitter buffer implementation, VoIPmonitor implements three MOS scores:

- MOS F1 – fixed jitter buffer simulator up to 50 ms buffer.
- MOS F2 – fixed jitter buffer simulator up to 200 ms buffer.
- MOS adapt – adaptive jitter buffer simulator up to 500ms buffer.

VoIPmonitor assumes that the call uses a G711 codec with maximum MOS score of 4.5.

4.7. Implementation of Standard Deviation Tracking

The file responsible for the migration of the dirty pages is *xc_domain_save.c* and is located in the *tools/xlibc/* directory of the unpacked source code of the Xen hypervisor. Since it is written in C++, a compiler, gcc, as well as a text editor, Geany [44], is installed on the hosts to edit and compile after affecting the desired changes to *xc_domain_save.c*.

Geany [44] is a text editor developed to provide a small and fast integrated development environment (IDE) and has been selected over the default editor that comes

with Linux such as *vi* due to its user friendly features such as symbol name auto-completion as well as code folding for easier organization of the written code.

First, the variables that will be required to compute and track the standard deviation are initialized as highlighted from line 255 to line 265 of the *xc_domain_save.c* file. In Fig. 4-11, the author's additional coding are commented and labeled as "//PN" to distinguish from the original lines of code.

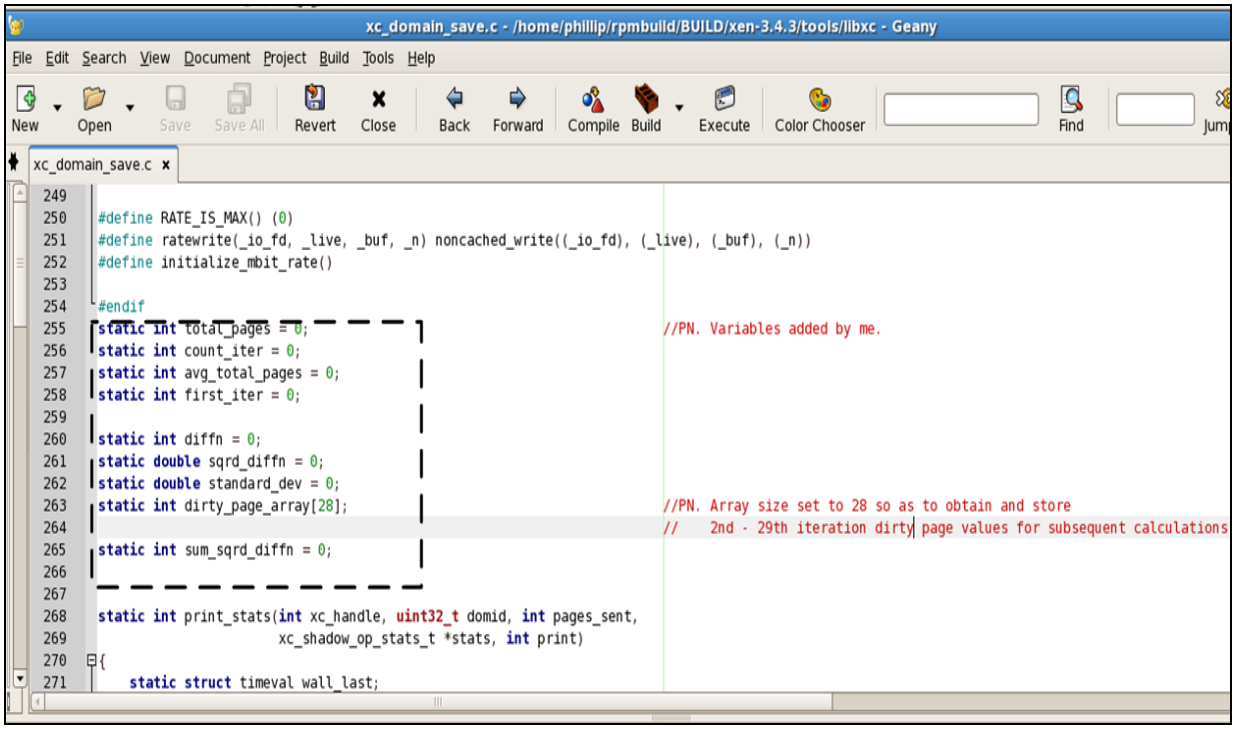


Figure 4-11: Initialization of variables used to calculate the standard deviation.

The calculation of the standard deviation as stated in the design assumption only considers dirty pages from the 2nd iteration onwards. To implement this, the number of dirty pages at each iteration is stored in an array under variable `dirty_pages_array` and is computed at each iteration for all dirty pages up to and including the current iteration. Fig. 4-12 below shows the implementation of this proposal from line 360 – 379.

Xen writes its logs into /var/log/xen/ directory. By default during the iterative pre-copying phase, information such as delta (duration of each iteration), sent rate, dirty rate and the number of dirty pages are recorded into the logs. This is shown in Fig. 4-13.

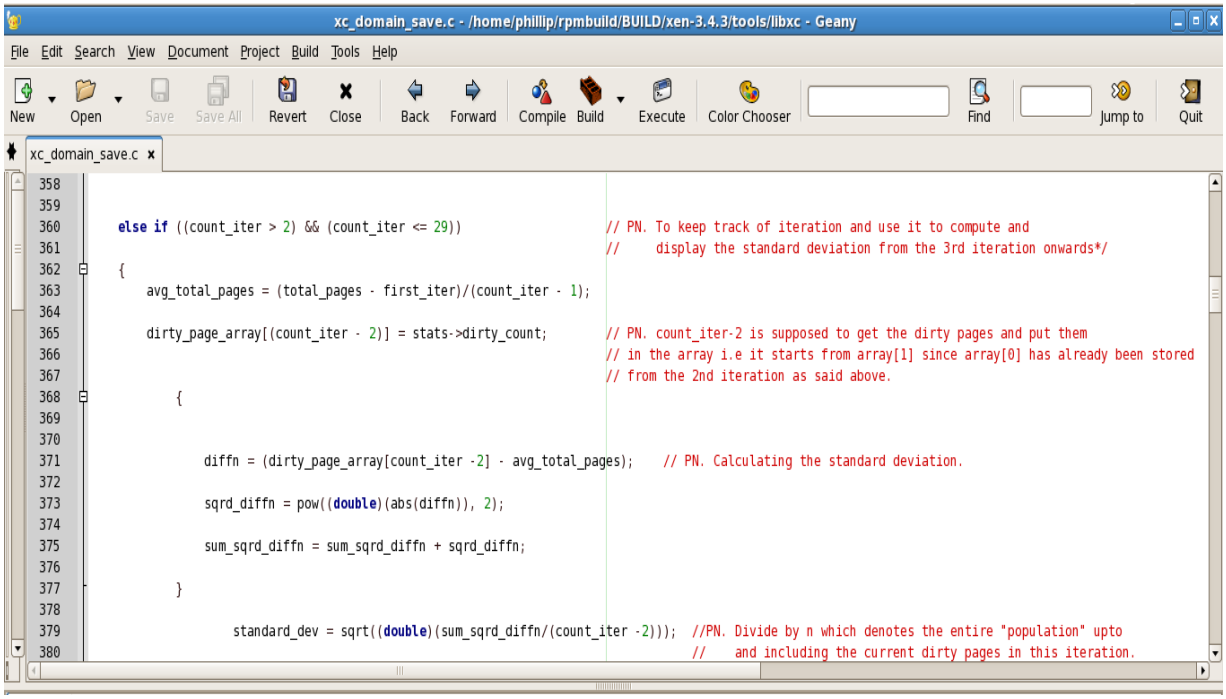


Figure 4-12: Standard deviation calculation at each iteration.

Since the proposed design is to track the dirty pages and subsequently compute the dirty page standard deviation, it is then required to modify this log output to show this result.

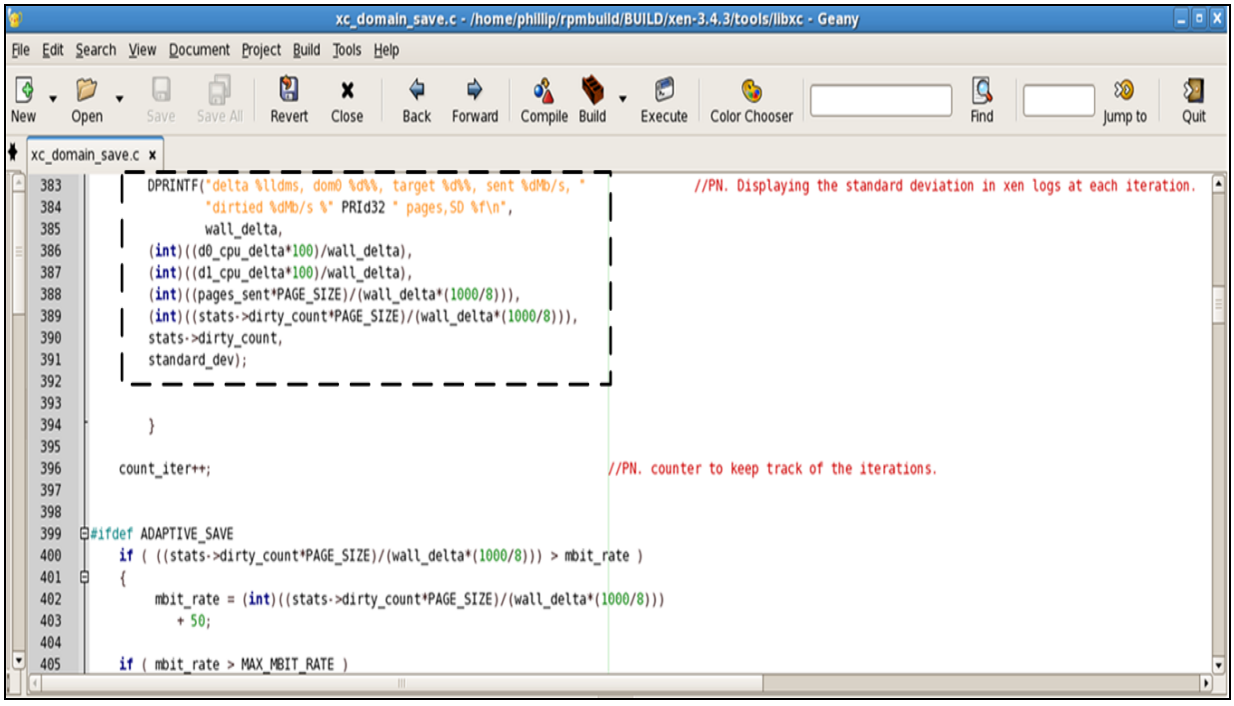


Figure 4-13: Modified Xen log showing standard deviation.

This has been appended on line 391 as shown in Fig. 4-13 to the `DPRINTF` command which is used to print out the information to the Xen logs. The data from the logs is collected and exported to a spreadsheet application such as Microsoft Excel for further analysis.

Fig. 4-14 below shows the section of code that deals with the stop condition of the pre-copying algorithm. The tracking of the dirty pages' standard deviation during the iterative pre-copying process is only considered after the 10th iteration as was stated in the design assumption. The permissible range that will trigger the stop of the iterative pre-copying of dirty pages is set at line 1418. This also shows that the tracking will only begin once the variable `count_iter` is equal or greater than 10.

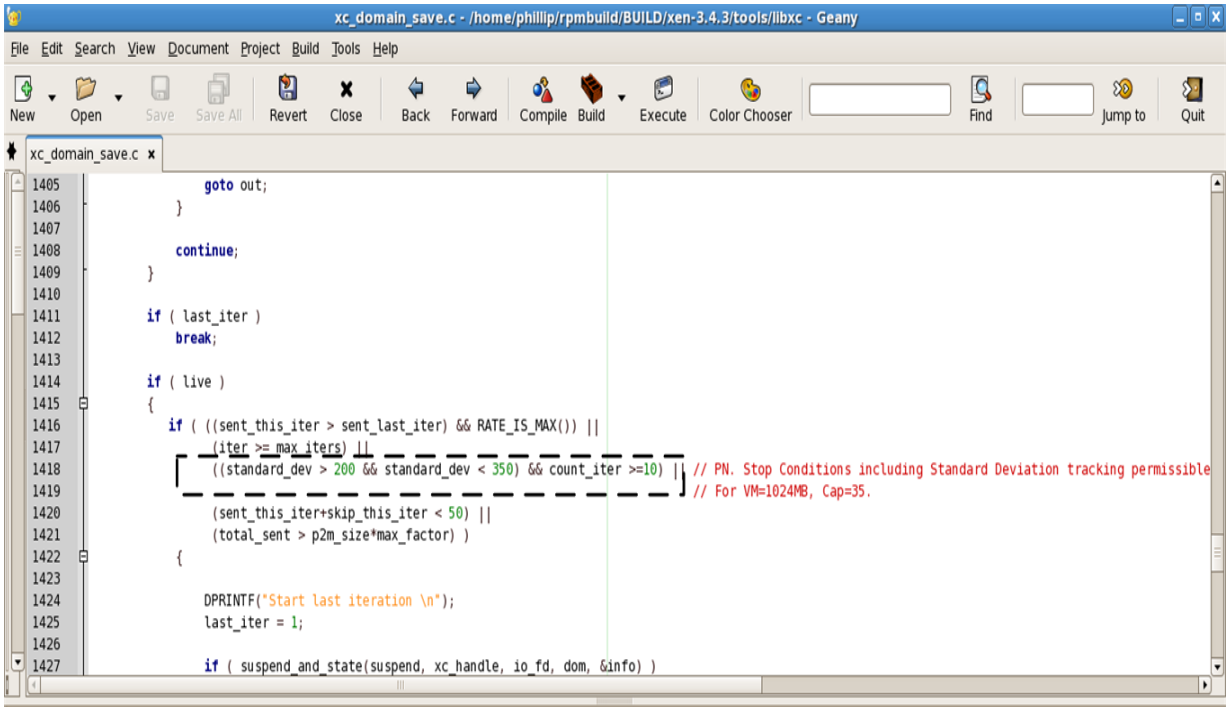


Figure 4-14: Standard deviation permissible range.

4.8. Experimental Procedure

To incorporate all the elements of the testbed discussed above, two sets of experiments are carried out:

- Pre-design experiments
- Design experiments

First, pre-design experiments are performed to establish the performance bounds of the VMs and the hosted services in the testbed. The VMs and their hosted services are put under stress from the traffic generators during the live migration process to get an idea of what parameters need to be set to obtain consistent and realistic outcomes.

The design experiments are then carried out to assess the improvements of the design modifications made to the original pre-copy algorithm.

4.8.1. Pre-design Experiments

The parameters obtained from this set of experiments will remain fixed for the design experiments carried out thereafter.

4.8.1.1. VM RAM Allocation/VM Size

The Asterisk VM RAM allocation is varied to establish how much RAM will be sufficient to run the application while it is under stress from SIPp. The allocation is varied from 64MB to 1024MB while running tests when the VM is idle as well as when it is active. Idle state implies that no workload is being generated to Asterisk VM whereas the active state is during the workload generation by SIPp.

4.8.1.2. Traffic Generation

Signaling traffic using SIPp is used to create stress on the Asterisk VM to determine what the maximum number of calls the application can handle. The delay variable that controls the length of the calls is varied between 5 – 30 seconds. Once the right value is obtained, it will be fixed for the design experiments.

Xstress will concurrently be generating a workload to the Postfix VM. Since the focus is on the migration of the VoIP service, this will be simply set to send out a total of 1000 emails from jane@xen.com to john@xen.com. The presence of this service is to create host and network resource contention scenario of a multi-tenancy model.

4.8.2. Design Experiments

To assess the design proposed in this research, live migration experiments are carried out to set a benchmark performance of the original pre-copy algorithm. Once this is established, then the design modifications are made to the pre-copy algorithm and the same

set of live migration experiments is done to observe the optimization of the live migration process and the improved service performance.

4.8.2.1. Standard Deviation

The implementation of the standard deviation tracking of the dirty pages will be carried out and thereafter experiments to verify the improvements it has made as compared to the benchmark performance of the original pre-copy algorithm.

The permissible range of the standard deviation is set based on the observations made after extensive experiments of migrating the Asterisk VM.

4.8.2.2. Credit Scheduler Cap

The varying of the cap has a significant impact on the downtime. Experiments will be carried out by reducing the cap and observing live migration performance metrics such as downtime as well VoIP service-specific QoS metrics such as packet loss, packet delay and MOS.

First, experiments will be carried out to determine the standard deviation profile for each cap and from this a selection of the suitable cap will be made. Once this cap is fixed, further experiments will be carried out using standard deviation tracking and the improvements will be recorded.

4.9. Summary

In this chapter, the proposed implementation of the solution methodology to the stated research problem is demonstrated. First, a detailed explanation of each component used to build the testbed is extensively discussed, with the aim of creating a clear picture of the environment in which this research is being carried out.

A variety of performance monitoring tools as well as traffic generators that will be interrogating the hosted services are installed in the testbed environment. The implementation of the proposed standard deviation tracking scheme is carried out in C++ and incorporated into Xen's pre-copy algorithm.

Finally a procedure of the experiments that need to be carried out to demonstrate the optimization of the live migration process and effect on service quality using the proposed

design is made. The results of these experiments are all discussed in the next chapter with the aim of justifying the proposed design.

5. Performance Evaluation

In chapter 4, the solution framework as well as the testbed environment in which it will be conceptualized was introduced. In this chapter, the effectiveness of this design implementation is the main concern. The evaluation of the design framework is based on relevant performance metrics that will highlight the performance optimization of the live migration process as well as its effect on the service quality during the same process.

The experiments mentioned in section 4.8 are designed to demonstrate, at a small scale, a real-world scenario of what a service provider would expect in terms of performance and service quality during a live migration of a real-time service.

The results of this chapter will then pave way for what needs to be considered for future work in this area of research.

5.1. Performance Monitoring

In order to evaluate the performance of the hosted VoIP service during the migration, the following parameters will be monitored to determine the live migration performance as well as the quality of the service to the end user.

- Downtime
- Total migration time
- Response time
- Packet loss
- Packet delay variation
- Mean Opinion Score (MOS)

By considering these parameters, deductions can be made on the performance of the proposed design framework.

5.2. Pre-design Experiment Results

In this experiment, the RAM allocation of the VM was varied from 64MB to 1024MB as illustrated in Fig.5-1 to assess the migration time while it was either idle or active. The active state in this case is when SIPp was sending signaling traffic using the UAC scenario to extension 2002.

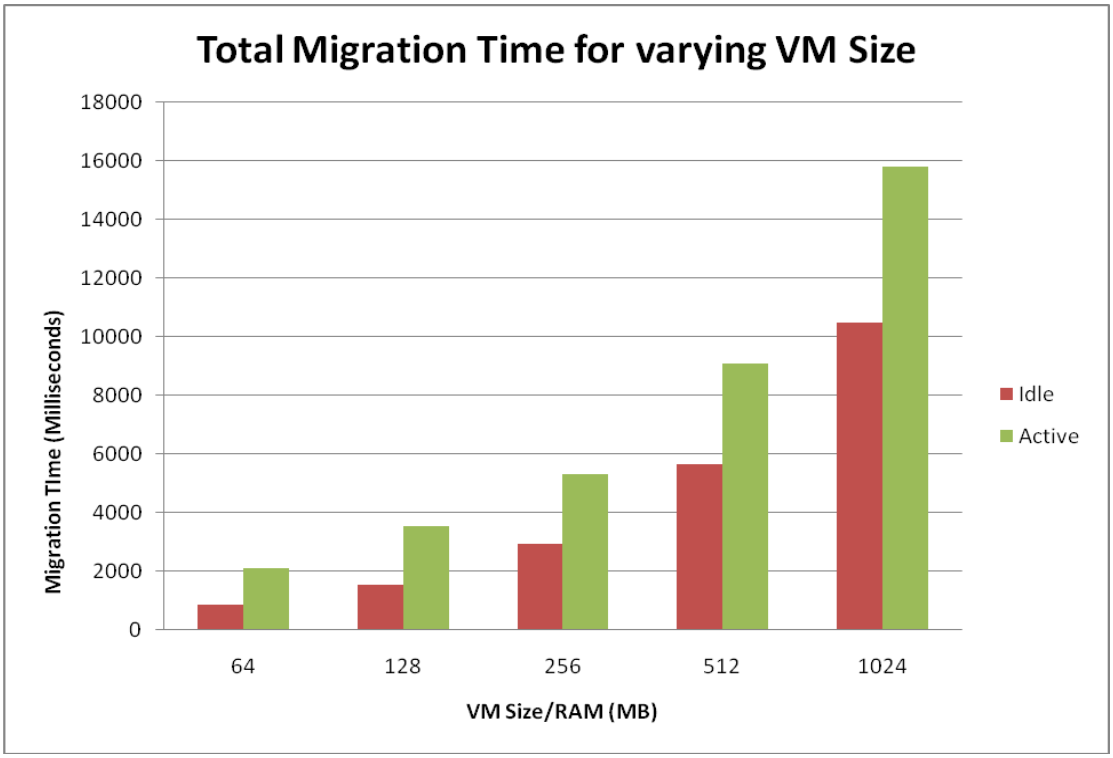


Figure 5-1: VM size effect on migration time.

As expected, it took a longer time to migrate a larger VM i.e. 1024MB than it did a smaller one in this case 64MB. This primarily was because the first iteration copied the entire memory during this round before the iterative copying actually began.

It was observed during the active state experiments that the VM would crash and become unresponsive when set to 64MB. The reason is because the memory allocation was grossly insufficient to handle the workload being generated by SIPp. The Asterisk application was relatively stable as the RAM was increased to suit the needs of the application.

From this series of experiments, it was established that to ensure performance consistency, the RAM allocated to the VM is to be set to 1024MB. This RAM allocation was

more than enough for the application and the stress being put on it during traffic generation. This remained the case for all the subsequent experiments.

To establish the traffic generation parameters, the UAC scenario of SIPp was used to generate signaling traffic to extension 2002 which is registered to Asterisk server at IP address 10.10.10.12. A maximum of 1000 calls were made before the traffic generator terminated the test. The delay parameter, d was varied while observing the downtime during the live migration process. Fig. 5-2 shows the results that were observed.

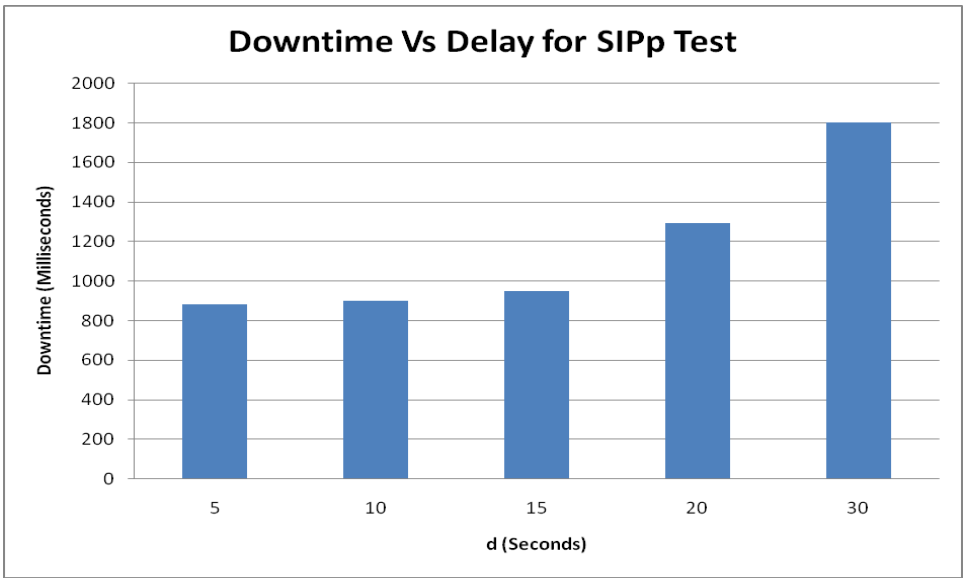


Figure 5-2: Downtime for varied delays, d.

The d parameter, as earlier mentioned, controls the duration of the call and indirectly the number of calls that the Asterisk application can concurrently hold. It is observed that as the delay parameter is increased from 5 seconds to 20 seconds, there is an increase in the dirty pages generated as there are more and more simultaneous calls being held by Asterisk. This is shown in Table 2.

Table 2: SIPp test results showing failed calls and downtime after live migration.

d (s)	Dirty Rate (Mbps)	Downtime (ms)	Failed Calls
5	718.73	881	45
10	835.29	900	95
15	1116.7	950	145
20	1263.65	1295	195
30	1552.77	1800	1000

SIPp also keeps a record of failed calls for the duration of the test. During the stop-and-copy phase during the live migration, the brief loss of network connectivity between the traffic generator and Asterisk resulted in failed calls. This is shown in Table 2 which tabulated the number of failed calls for each corresponding d value.

It is noted that when d was set to 20 seconds and the SIPp test is run but with no live migration taking place, the Asterisk application could handle a maximum of 200 simultaneous calls and no failed calls were observed. Fig. 5-3 shows the total active calls in the system to be the maximum which is 200 calls.

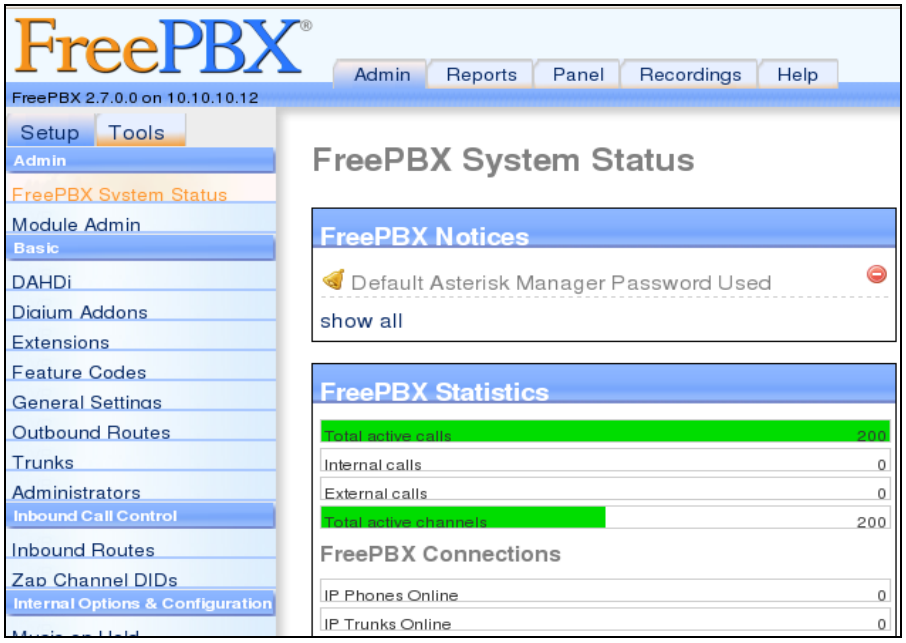


Figure 5-3: Simultaneous calls when d = 20 seconds.

A further increase to set d to 30 seconds and running the same test (i.e. no live migration) resulted in all the calls being dropped. This is further illustrated in Table 2 where during the live migration all 1000 calls that were made, failed to be established. This is attributed to the insufficient computing resources available such as RAM and CPU to meet the demand to maintain more than 200 concurrent calls.

In this series of experiments, it was also established that a d value of 20 seconds was optimal. This is because it gave the performance benchmark when the system was running at full capacity.

To also get an understanding of the effect live migration had on co-hosted services, the response time of Postfix was monitored during the live migration of the VoIP service.

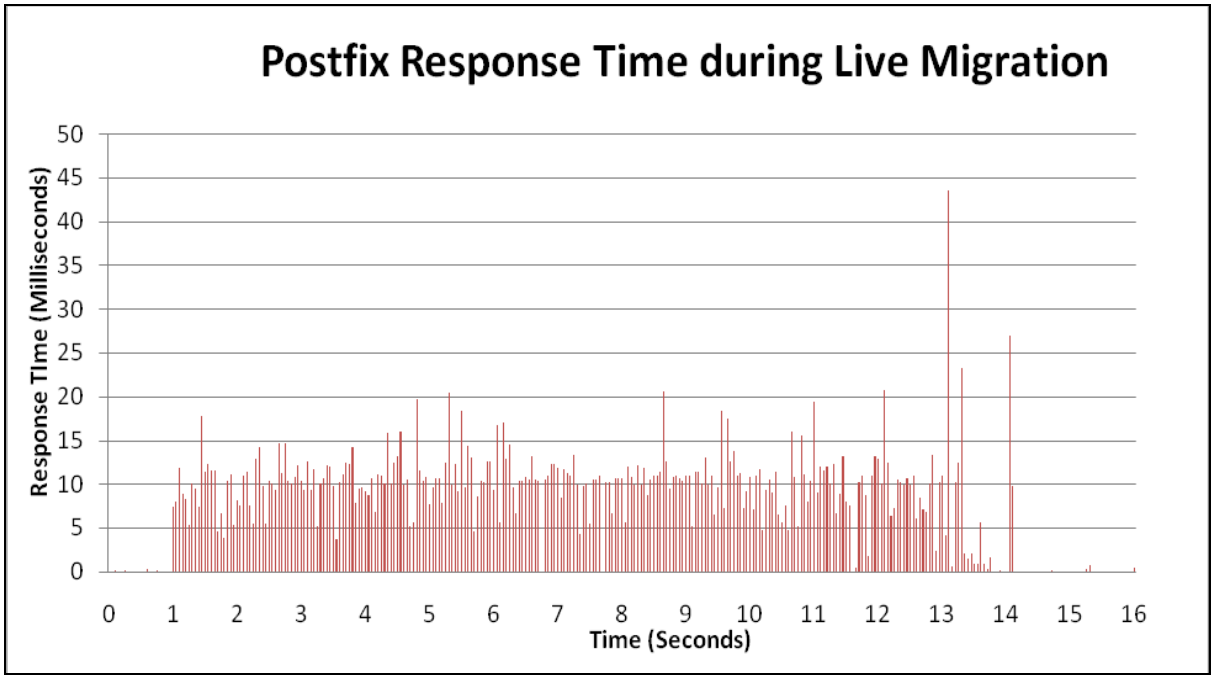


Figure 5-4: Postfix response time during live migration.

The iterative process began at the 1st second mark and ended at approximately the 13th second mark, at which time the average response time was 10.35ms. After the migration was complete the response time reduced to an average of 0.13ms as shown in Fig.5-4 after the 14 second mark.

Having Postfix running concurrently with Asterisk resulted in a mixed workload scenario.Using xstress to generate traffic during the live migration contributed to an increase in input/output (I/O) utilization on the host as shown in Fig. 5-5. More specifically this was disk I/O as this involved writing and storing mail to the mailbox directory of the receipient. Also the network I/O showed only incoming traffic to the email server. This was attributed to the mail being sent by xstress to Postfix. Its is important to note that emails contained a text body and no attachments. The low outgoing traffic may be attributed to control traffic required to maintain the SMTP connection between xstress and the email server. In Fig. 5-5, the I/O throughput figures shown for Disk and Network, i.e.,18MB/s and 42KB/s are instantaneous values observed during the polling of the respective performance metric.

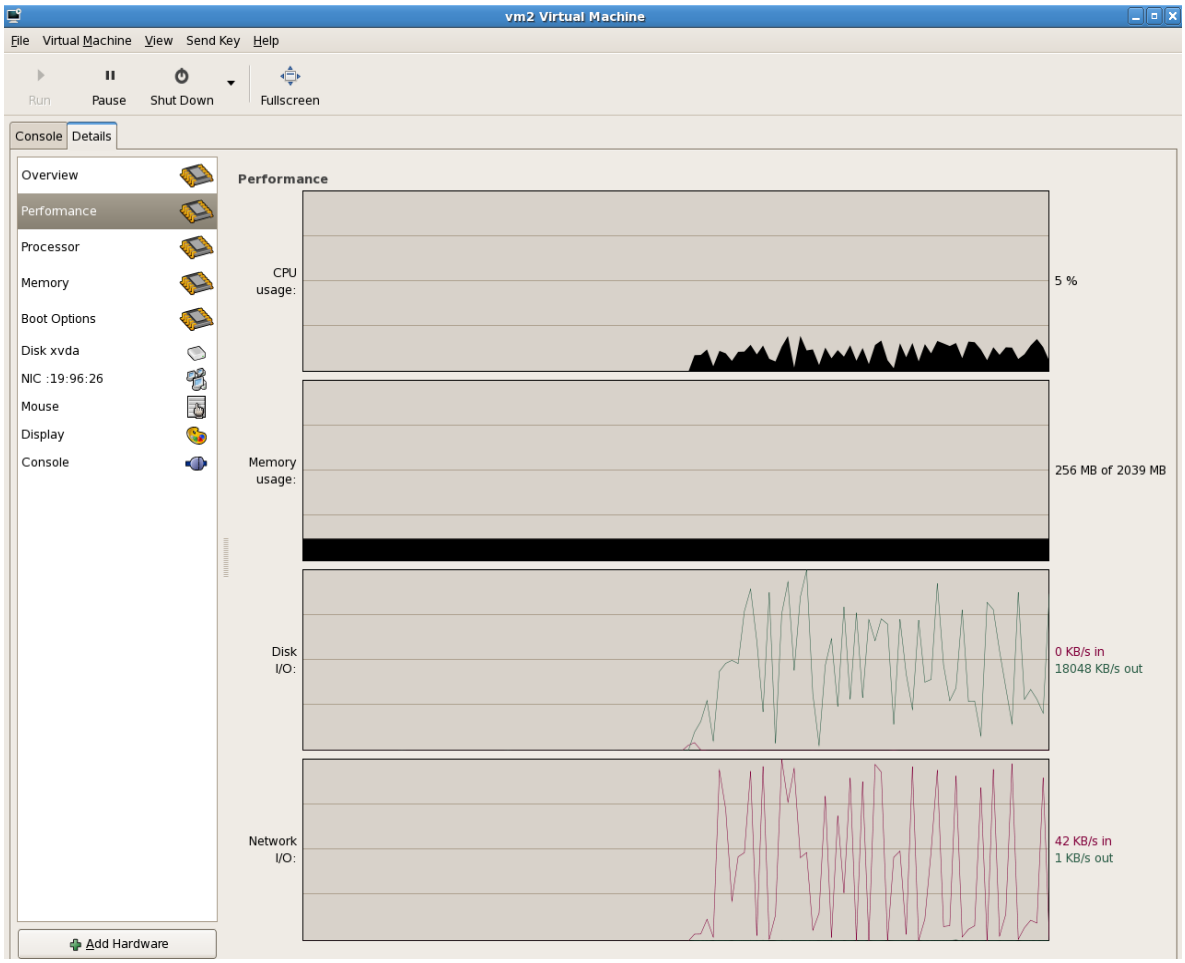


Figure 5-5: Snapshot of I/O generation by xstress on VM running Postfix.

A timeline was drawn to illustrate the various stages of the pre-copy algorithm as discussed in section 2.4, in this case, during the live migration process of the VoIP service. Figure 5-6 shows 3 stages A, B and C.

Stage A in this case represented the pre-migration and reservation phase. Stage B was the iterative copying of the dirty pages from the source to the target host and it is observed that there is a noticeable increase in the response time as a result of increased CPU usage. Stage C is the stop-and-copy as well as commitment phase. Migration completion confirmation and network reattachment to the target host takes place at Stage C.

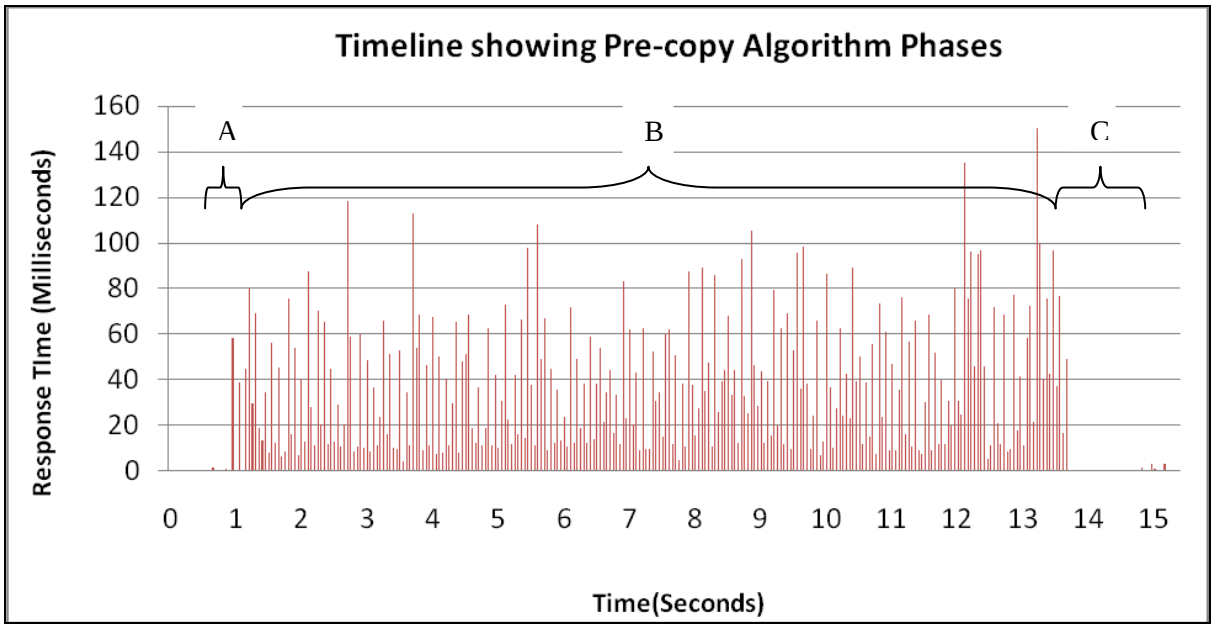


Figure 5-6: Timeline Showing Pre-copy Algorithm phases.

5.3. Design Experiment Results

In these series of experiments, the credit scheduler cap was varied and the downtime, dirty rate and the ping response time were recorded and tabulated into Table 3 below.

Table 3: Credit scheduler cap experiment results.

Cap	Downtime (ms)	Dirty Rate (Mbps)	Service Response Time (ms)
20	550	364.1	61.62
25	500	457.52	48.62
30	650	642.37	50.73
35	800	860.57	38.55
40	1000	869.32	33.24
50	1100	1015.54	17.78
60	1150	1262.57	13.75
100	1050	1297.23	10.93

It was observed that there was an inverse relation between CPU limitation of the VM and the response time. At lower caps, the VM's processes get less time on the physical CPU and hence an increase in response time of the VoIP service. At higher caps, there is an increased availability of processing capacity and thus a lower response time. As had been mentioned in chapter 2, by default, Xen has a cap set to 0. This implies that the VM will utilize available CPU cycle and this is represented as 100% in Table 3. A 6 fold increase in

the response time from the default is noted by setting the cap to 20%. This decrease in the cap has an impact on the quality of the call as will be shown later in this section.

The CPU limitation also had a direct relation with the downtime. This is primarily because CPU plays a pivotal role in the dirty page generation of a VM. Allocating low caps implied less time was given to the VM to generate dirty pages and hence a decrease in the downtime was recorded. This means that the downtime experienced was reduced by half when the cap is set to 20% as compared to the default (represented by 100%).

To establish a trend line in Table 3, the parameters were represented graphically as shown in both Fig.5-6 and Fig.5-7, where the cap was drawn vs. response time and downtime respectively. The trend line equations were automatically obtained by MS Excel.

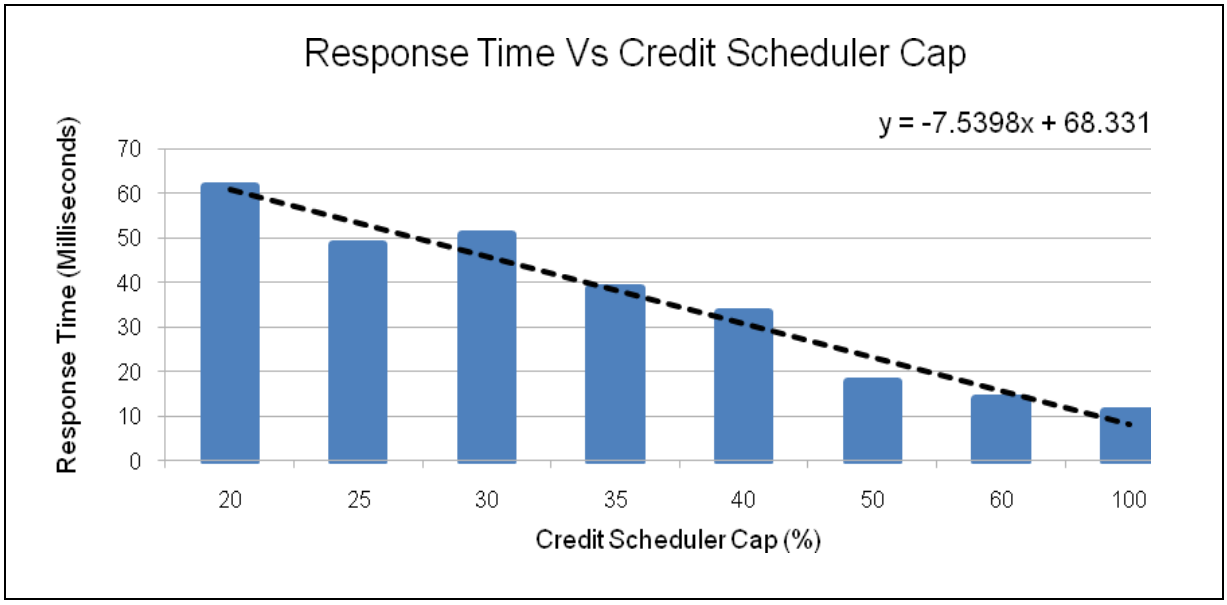


Figure 5-7: Response time of varying caps.

The trend line in Fig.5-7 as discussed showed the general decline in response time as the cap was increased from 20% to when it was at the default. Also it was observed in Fig.5-8 that there was an increasing trend of the downtime with each cap increment.

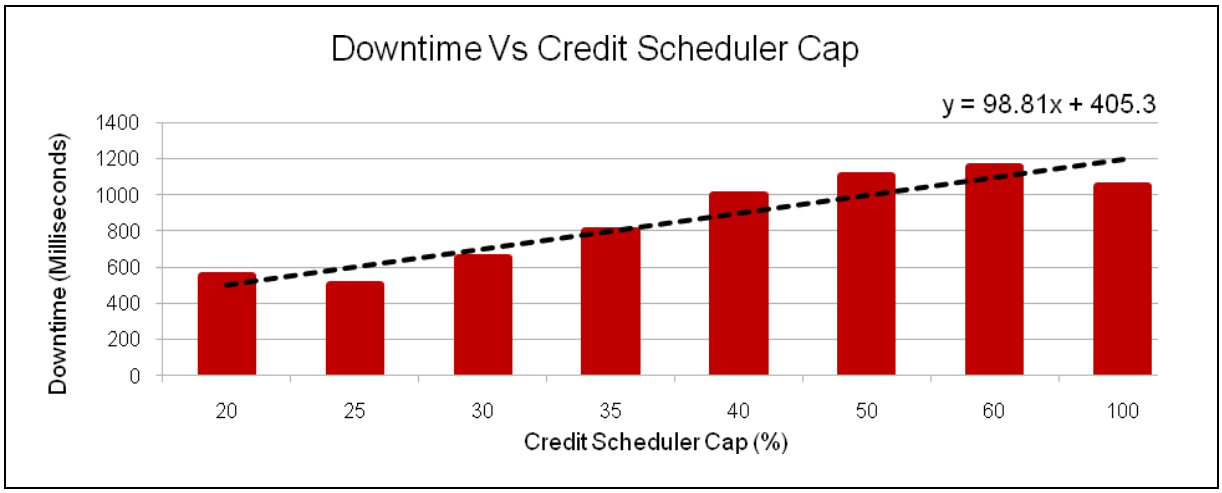


Figure 5-8: Downtime of varying caps.

The trend lines of both response time and downtime were then used to establish an optimal cap that balances the two parameters. In order to do this, the downtime trend values were given a weighting of 0.05. The downtime was considered to be the value of importance and therefore a weighting assignment of 0.05 was used so as to draw a comparative trend with the response time. A weighted comparison was then made between the downtime and the response time trend values from which a maximum was obtained at each cap as shown in Table 4. The best cap that achieved the above goal is when the cap was set to 35.

Table 4: Weighted trend lines of response time and downtime.

Cap	Response Trend (ms)	Downtime Trend (ms)	Max. Comparison
20	60.79	25.20	60.79
25	53.25	30.14	53.25
30	45.71	35.08	45.71
35	**38.17**	**40.03**	**40.03**
40	30.63	44.97	44.97
50	23.09	49.91	49.91
60	15.55	54.85	54.85
100	8.01	59.79	59.79

This was also graphically shown in Fig. 5-9 where it is seen that the optimal cap that would balance the downtime and the response time is when the cap was set to 35%. According to Table 3, at a cap set to 35 the downtime is decreased by 23.8% from the default (cap =100) whereas the response time was increased by 252.3% from the default.

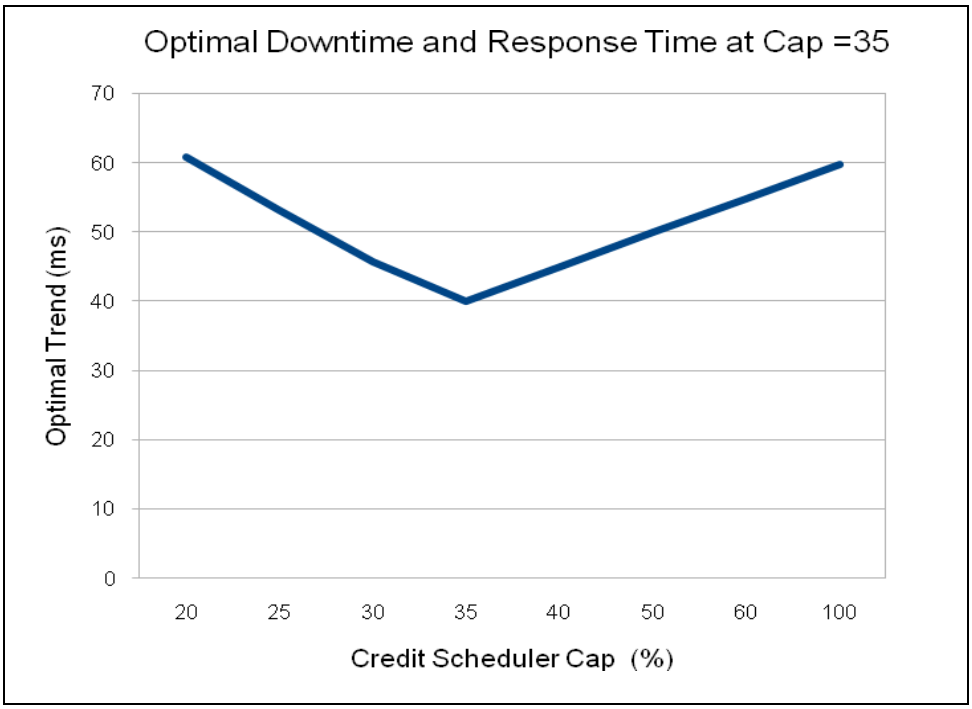


Figure 5-9: Optimal downtime and response time at cap = 35.

In the following experiments, the design framework results were obtained showing the contribution of the standard deviation scheme in conjunction with the credit scheduler cap. By varying the cap and taking note of the standard deviation at each iteration a profile was obtained for each cap.

This is shown in Fig. 5-10 and Fig.5-11. The standard deviation profiles shown in Fig.5-10 are for caps ranging from 20 – 35% whereas in Fig.5-11 the standard deviation profiles are for caps ranging from 35 – 100%.

The standard deviation profiles for the caps 20%, 25% and 30% as shown in Fig. 5-10, do not have a stability region and only 4, 9 and 12 iterations respectively are carried out during the live migration. This is because the cap limits set reduced the dirty page generation significantly such that the stop condition of less than 50 dirty pages remained thereby triggering the termination of the iterative pre-copy phase.

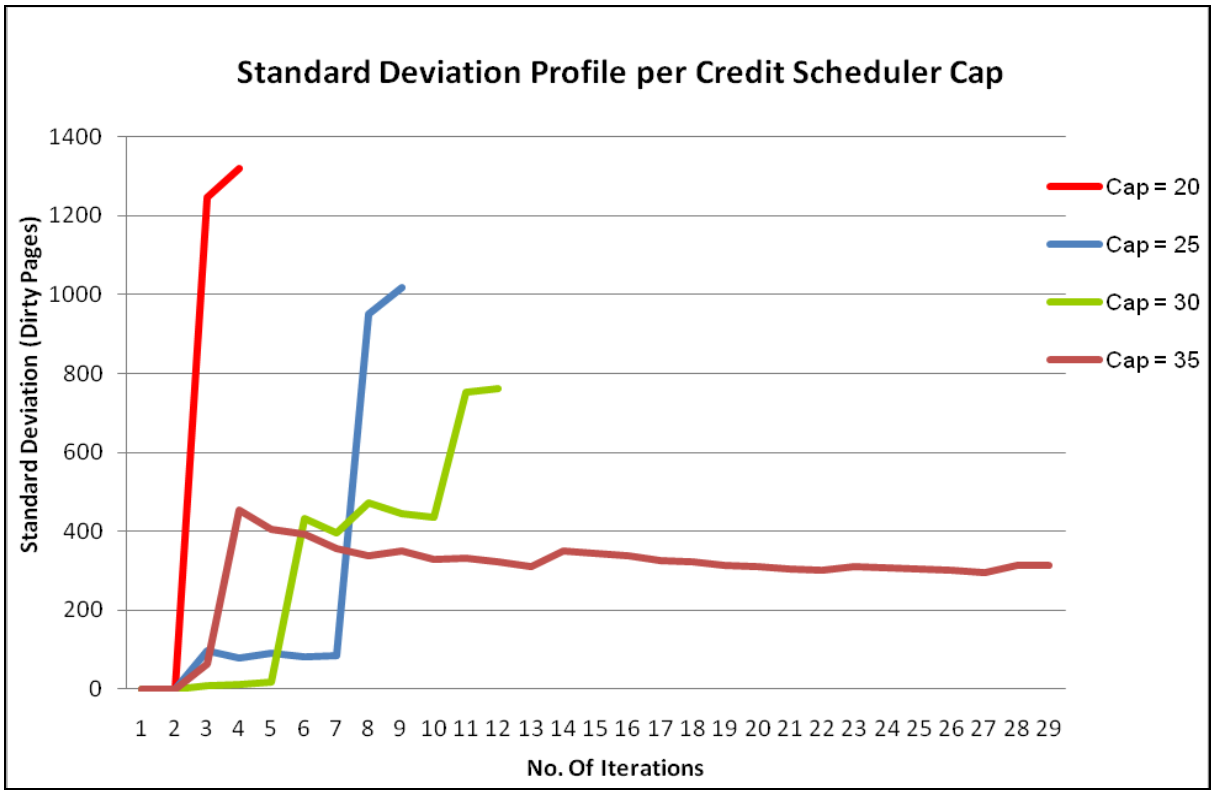


Figure 5-10: Standard deviation for varied caps from 20 – 35%.

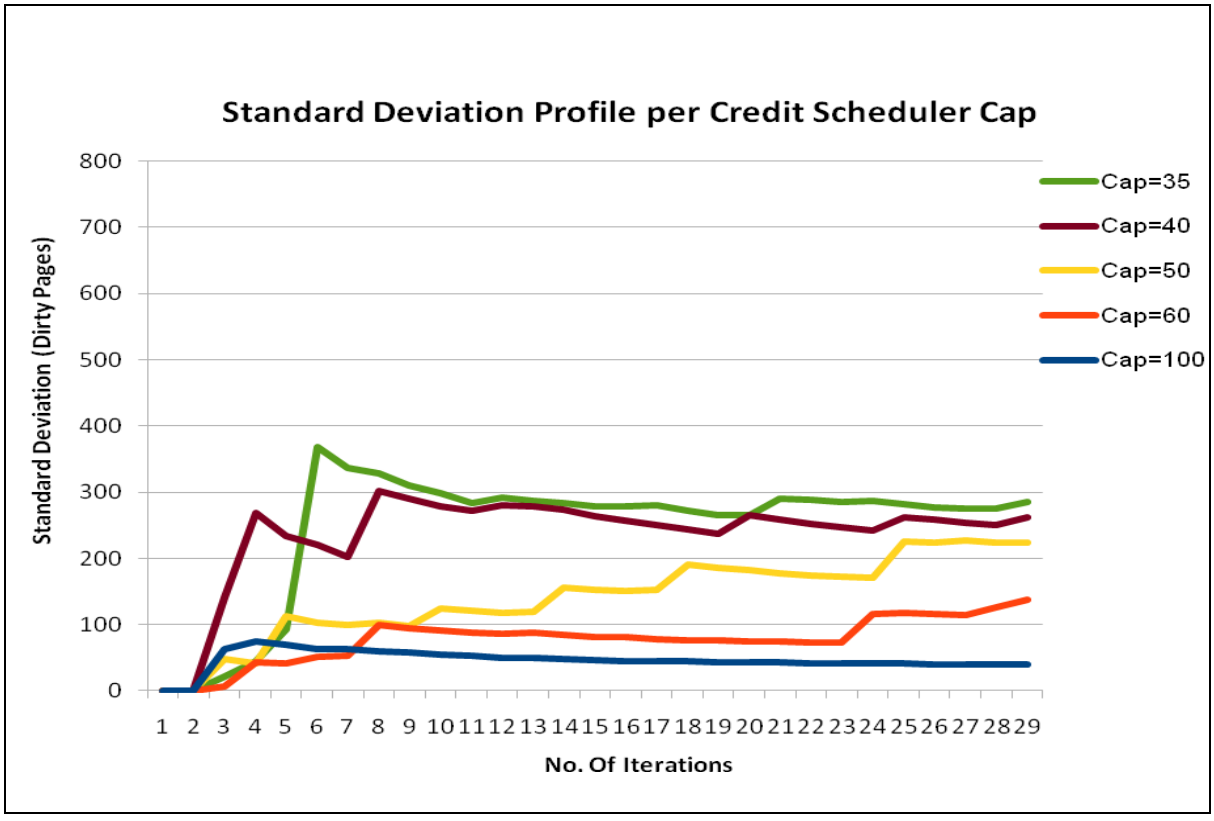


Figure 5-11: Standard deviation for varied caps from 35 – 100%.

The standard deviation profiles for the caps 35 – 100% took the maximum number of iterations which is 29. This is shown in Fig.5-11. This was because the dirty pages generation was faster than the rate at which they were being copied to the target host thereby the stop condition of 29 iterations triggered the termination of the iterative pre-copy phase.

As was stated in the design assumptions, a stability criterion would have to be met so as to qualify for standard deviation tracking. By considering all the profiles, the lower caps (20 – 30) did not qualify as viable candidates. The rest of the profiles qualified as viable standard deviation tracking as they displayed a region of stability over the 29 iterations carried out.

As was discussed earlier, a cap of 35% was considered to be the most optimal cap that balanced downtime and response time and was therefore considered to be the most suitable candidate for standard deviation tracking. A series of live migrations as depicted by each line in Fig. 5-12, were carried out to determine the permissible range of standard deviation that was in the modified pre-copy algorithm. The range of standard deviation values that were considered to lie within the region of stability are between 150 – 300 dirty pages. Also based on the design assumption, the tracking of the dirty pages standard deviation only started from the 10^{th} iteration.

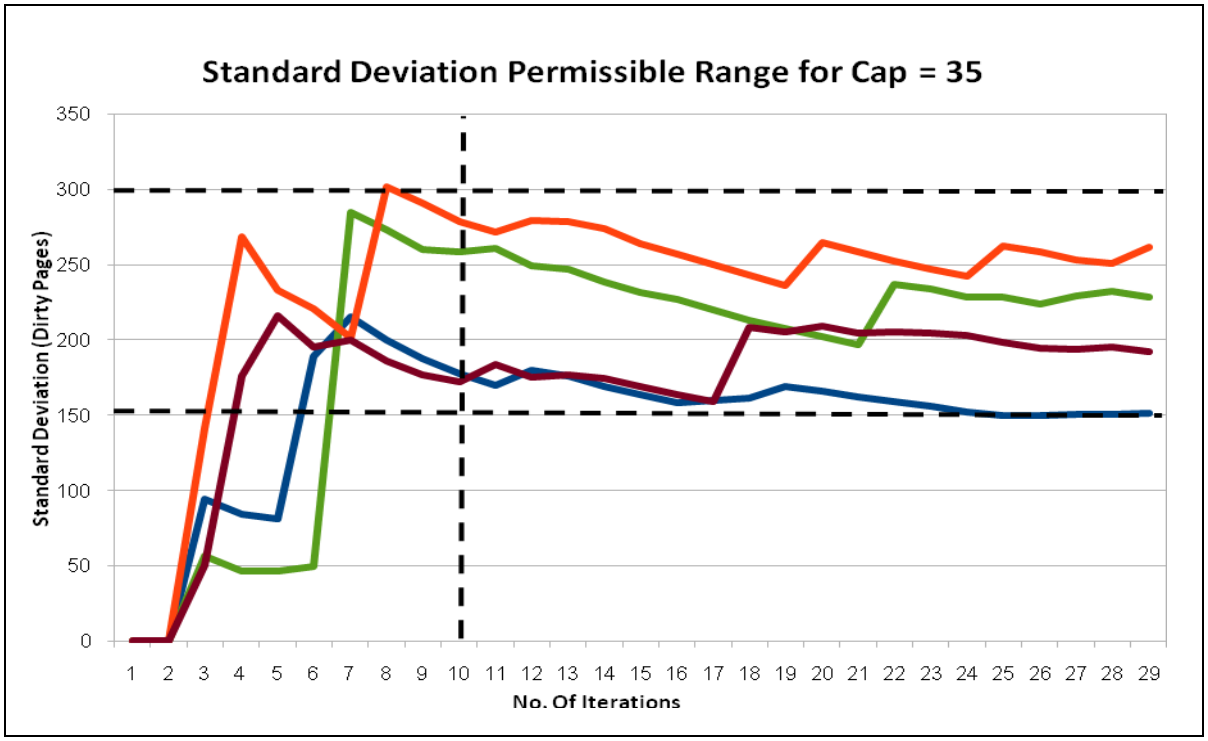


Figure 5-12: Standard deviation permissible range for cap = 35.

Once the permissible range was established, the pre-copy algorithm was modified to reflect these changes and recompiled into the source code. This recompiling procedure has been discussed in Appendix A.

Experiments were then carried out to observe the improvements that were made by implementing the standard deviation tracking scheme with a cap of 35. Fig. 5-13 below shows that after a series of experiments the tracking terminates at the 10th iteration as the permissible range i.e. a standard deviation between 150 – 300 dirty pages was met.

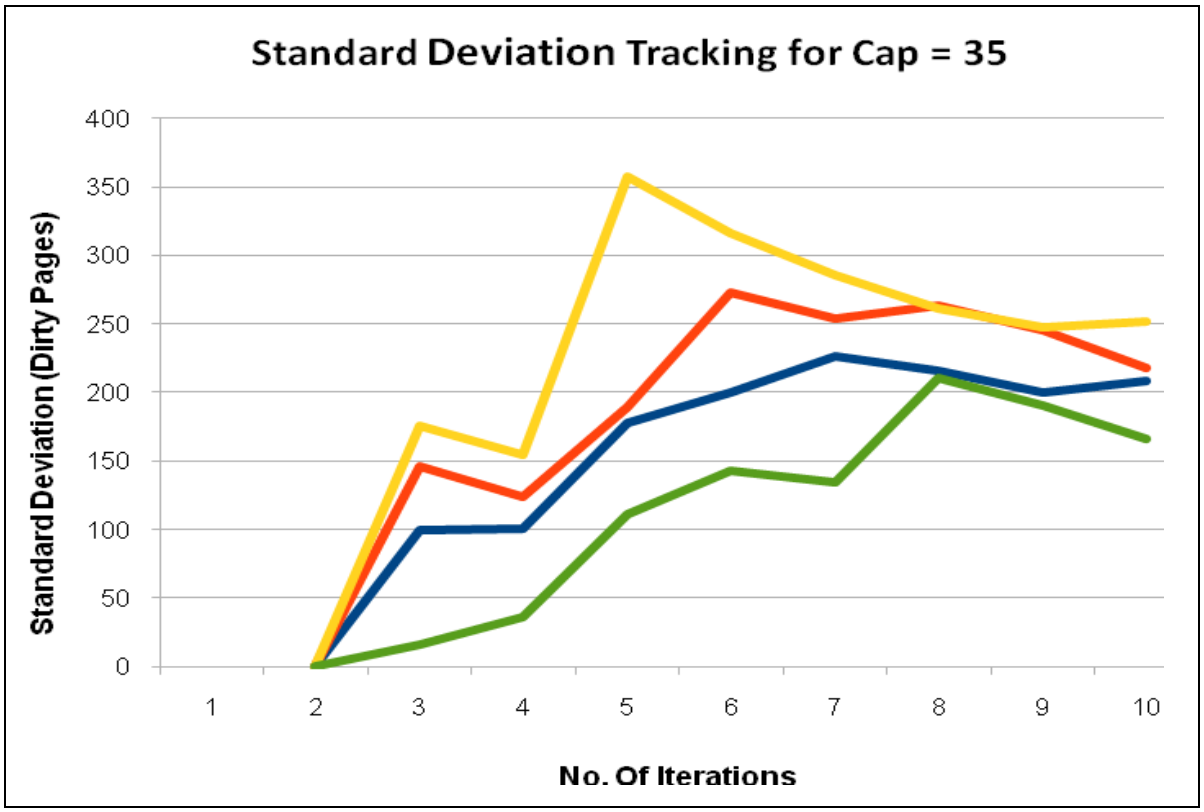


Figure 5-13: Standard deviation tracking results.

The live migration performance implications are that of the 29 iterations that would have normally been carried out only 10 are necessary hence an improvement in the migration time. Table 5 shows the comparison of the migration times prior to and after standard deviation tracking modifications to the pre-copy algorithm.

Table 5: Migration time comparison.

Avg. Migration Time (ms)	Avg. Migration Time - Optimized (ms)	% Change
13441	12369	7.98

The reduction of the migration time by 7.98% from the default is primarily because the tracking process eliminated the need for carrying out the remaining 19 iterations. Another important observation is that the migration time largely constituted the duration it takes to copy the entire VM memory in the first iteration. In this experiment the first iteration took on average 11.574 seconds on the default setup and 11.585 seconds on the modified setup which translates to 86.1% and 86.2% respectively of the migration time.

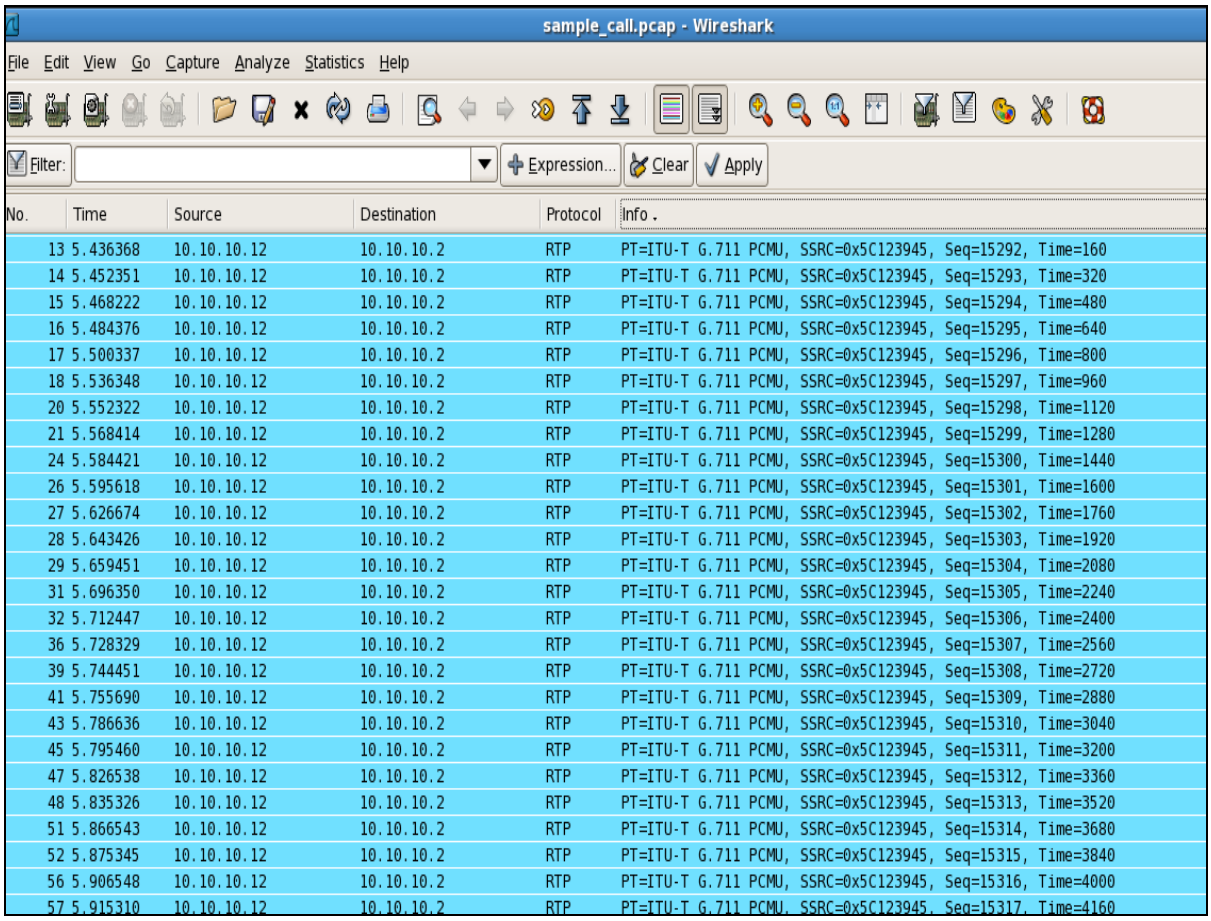
If this first iteration is taken into account and deducted from the migration time, then what remains is the time it takes to iteratively copy the dirty pages from the second iteration onwards. This yields a 58.01% reduction in iterative copying time (Table 6).

Table 6: Iterative copying time comparison.

Avg. Iterative Copying Time (ms)	Avg. Iterative Copying Time - Optimized (ms)	Change %
1867	784	58.01

The effect of reducing the cap allocated to a VM had an impact on the quality of the VoIP service. This was quantified using performance monitoring tools such as Wireshark and VoIPmonitor to detect service-specific QoS parameters. The VM with the VOIP service was then migrated from the source to the target host during which time a call was being made between two extensions.

Wireshark was used to capture the RTP packets during the VoIP call. Wireshark was initiated to monitor the Ethernet port, eth0 of the client machine to capture all the packets traversing through it. Then a 30 second call was initiated at the client on extension 2000 to a client with extension 2001. Fig. 5-14 below shows an excerpt of the captured RTP packets demarcated on the diagram under the protocol tab. Also the codec that was used by the client is also shown to have been G711.



No.	Time	Source	Destination	Protocol	Info
13	5.436368	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15292, Time=160
14	5.452351	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15293, Time=320
15	5.468222	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15294, Time=480
16	5.484376	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15295, Time=640
17	5.500337	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15296, Time=800
18	5.536348	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15297, Time=960
20	5.552322	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15298, Time=1120
21	5.568414	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15299, Time=1280
24	5.584421	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15300, Time=1440
26	5.595618	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15301, Time=1600
27	5.626674	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15302, Time=1760
28	5.643426	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15303, Time=1920
29	5.659451	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15304, Time=2080
31	5.696350	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15305, Time=2240
32	5.712447	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15306, Time=2400
36	5.728329	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15307, Time=2560
39	5.744451	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15308, Time=2720
41	5.755690	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15309, Time=2880
43	5.786636	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15310, Time=3040
45	5.795460	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15311, Time=3200
47	5.826538	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15312, Time=3360
48	5.835326	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15313, Time=3520
51	5.866543	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15314, Time=3680
52	5.875345	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15315, Time=3840
56	5.906548	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15316, Time=4000
57	5.915310	10.10.10.12	10.10.10.2	RTP	PT=ITU-T G.711 PCMU, SSRC=0x5C123945, Seq=15317, Time=4160

Figure 5-14: Wireshark showing captured RTP packets during call.

After the call was terminated, Wireshark was then used to analyze the captured packets. Fig. 5-15 and 5-16 illustrate the jitter profile of the call. At the 10th second mark the live migration process began and lasted approximately 11 seconds at which point the downtime occurred at the 20th second mark. This was denoted by the gap in the diagram. After the live migration was completed the target host took over hosting the VoIP service.

In both graphs it was noted that during the period of the live migration i.e. from the 10th second until approximately at the 21st second mark, the jitter was unstable and the maximum jitter value occurred during this period. This occurred during the iterative copying phase which illustrated the effect the live migration has on the call quality. From Fig. 5-15, when the cap was left as the default, the maximum jitter that was observed was 13.16ms which occurred during the iterative process of the live migration. In Fig.5-16 when the cap was reduced to 35, the maximum jitter was 14.26ms and there was degradation in the call quality during this period due to this instability.

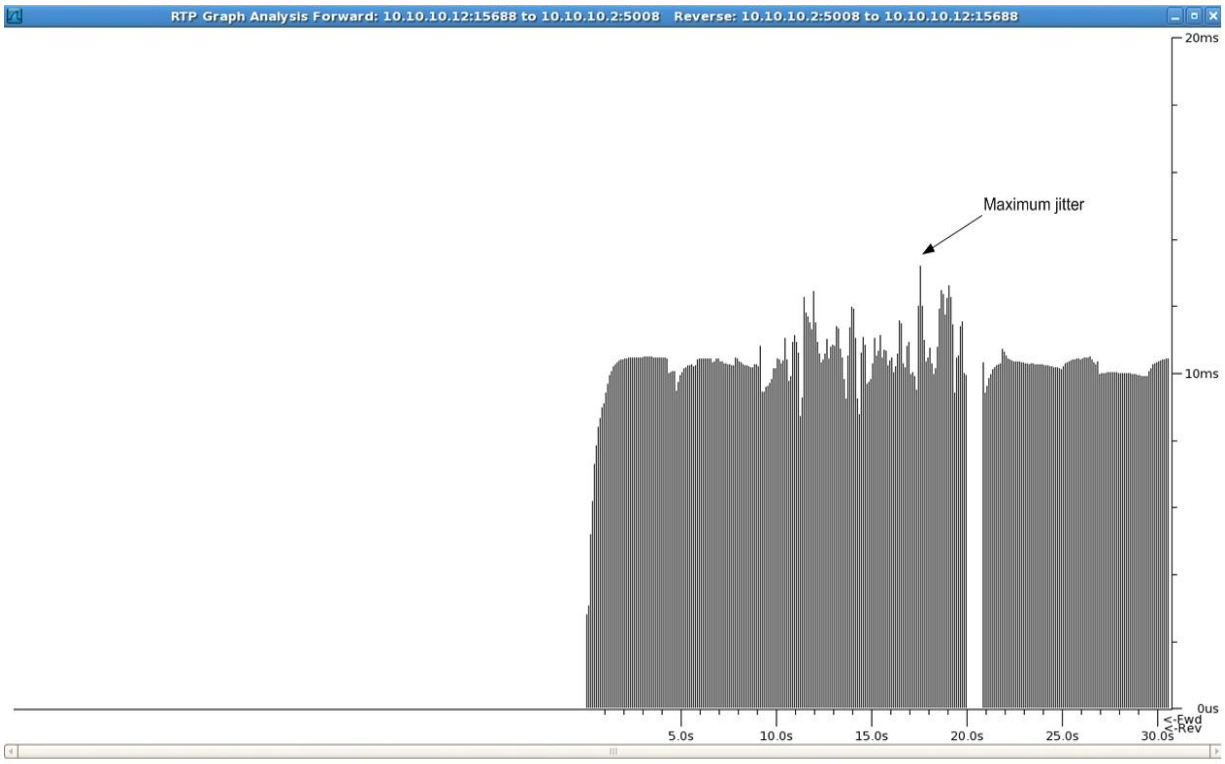


Figure 5-15: Wireshark graph showing jitter profile when cap = 0.

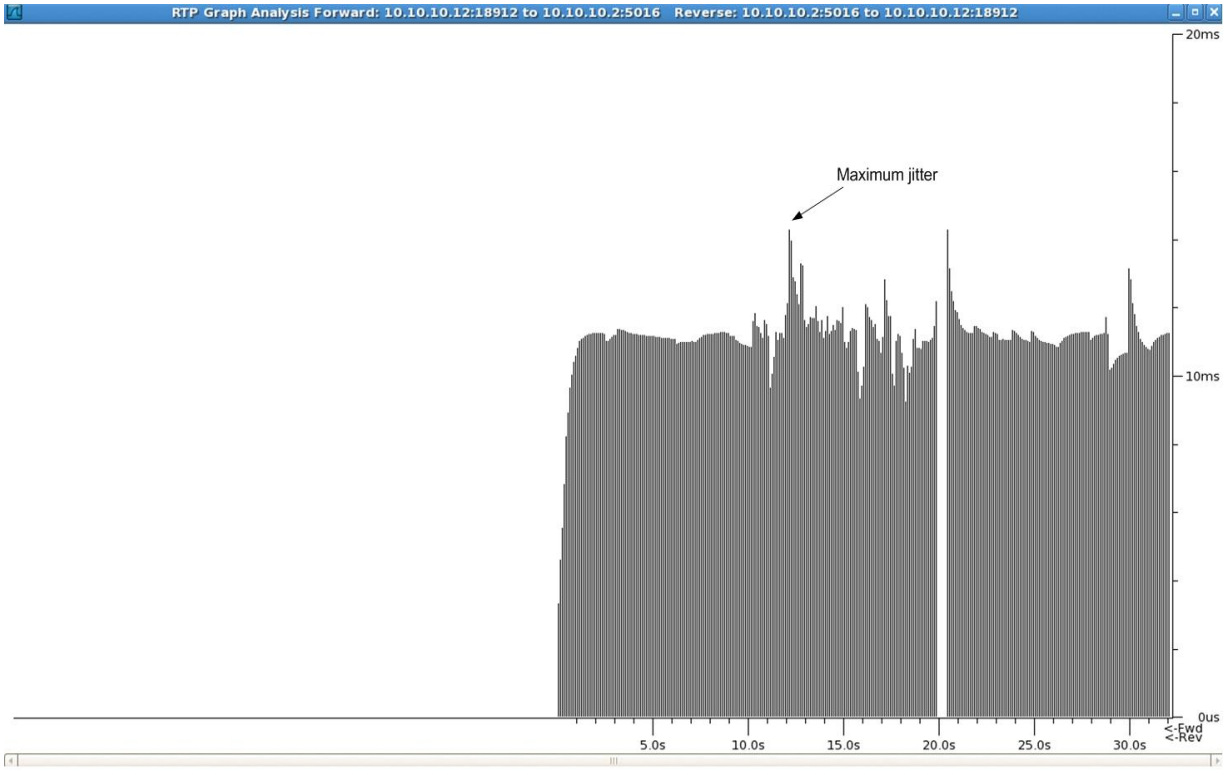


Figure 5-16: Wireshark graph showing jitter profile when cap = 35.

To give further insight into the call quality during this period, VoIPmonitor was also used to analyze the packet loss distribution, the packet delay variation as well as the MOS.

As mentioned in section 4.6.3, VoIPmonitor uses packet loss intervals to determine the distribution of packets lost during the call duration. This is helpful in identifying consecutive packet losses which would be more detrimental in the call quality rather than having the packet losses occurring sparsely over the duration of the call.

A call was made and the live migration was carried out with the default cap parameter and also with the cap set to 35. Table 7 below shows the results of the packet loss distribution for both cases. The default scenario showed that only 1 packet was lost and this occurred in the second loss interval. When the cap was set to 35, the packet loss occurs in almost all intervals most notably at the 10^{th} loss interval, hence degrading the call quality in comparison to the default.

Table 7: Comparison result for packet loss distribution.

	a_sl1	a_sl2	a_sl3	a_sl4	a_sl5	a_sl6	a_sl7	a_sl8	a_sl9	a_sl10
Cap=100	0	1	0	0	0	0	0	0	0	0
Cap=35	1	0	0	0	0	1	1	1	5	73

Also the packet delay variation (jitter) was used to assess the quality of service of the VoIP call during the live migration. With this metric measure, the variation in the delay of the arrival of the RTP packets was tabulated for both cap scenarios (Table 8).

Table 8: Comparison result for packet delay variation.

	a_d50	a_d70	a_d90	a_d120	a_d150	a_d200	a_d300
Cap=100	0	0	2	0	0	0	0
Cap=35	43	18	14	5	5	0	3

The default scenario had only 2 packets with delay between 90 - 120ms whereas when the cap was set to 35 the packet delays occurred in almost all of the delay intervals. The highest number of delayed packets occurred at the 50 – 70ms interval. This packet delay variation negatively impacted on the quality of the call during the live migration period.

The Mean Opinion Score (MOS) was also calculated using VoIPmonitor as shown in Table 9. As was mentioned, VoIPmonitor transforms PDV and packet loss which were obtained above into a MOS score according to ITU-T E-model. VoIPmonitor gives three scores based on the jitter buffer of 50ms, 200ms and above 300ms. The results obtained showed that irrespective of the buffer used, the default scenario got a better MOS score than when the cap was reduced.

Table 9: Comparison result for MOS.

	a_mos_f1	a_mos_f2	a_mos_adapt
Cap 0	4.28252	4.5	4.22431
Cap 35	3.53985	1	0.0905172

5.4. Summary

In this chapter, the theoretical discussions of the proposed design framework were verified through the analysis of the results and the implications it had on the VoIP service during the live migration. The pre-design experiments carried out established the performance bounds of the testbed's nodes and their hosted services so as to not set unrealistic parameters. The design experiments aimed at assessing the standard deviation tracking scheme and CPU scheduling concept were successfully undertaken to illustrate the theory discussed in the design chapter.

Results showed that the live migration performance parameters such as downtime and total migration time were optimized for the VoIP service as a result of applying the design conceptualized in this research. The trade off to this came at a minimal cost of call degradation as perceived by the user for the live migration period.

6. Conclusions and Recommendations

This research work highlighted the importance of optimally managing the live migration process with the aim of diminishing the service downtime and migration time. This would be beneficial to service providers who have consolidated their services through multi-server virtualization and whose business survival is highly dependent on satisfying SLA with their clients.

6.1. Conclusions

With the increase in demand for service provisioning in the cloud, business enterprises and service providers alike benefit from having VM migration as a central resource management tool in maintaining of SLAs. Hence, there is a need to understand the migration process implications on the specific service being migrated and customize the underlying virtualization platform accordingly.

The thesis presented a solution framework that addressed the problem by using a statistical approach to optimize the live migration process. Also the understanding of CPU resource management and the effect it had on VoIP service was used to get a suitable value that allowed for a balance between the service response time and service downtime.

The contributions of the work done in this research were mainly pertaining to the setup and design implementation used to optimize the live migration of the VoIP service and outlined as below:

- The work was carried out in a testbed environment that emulated, at a small scale, a real-world scenario of a service provider's data centre. The services that were being hosted in the testbed used real-world applications such as Asterisk and Postfix as voice and email services.

- By tracking the statistical behaviour, in this case, the standard deviation of the dirty pages, it was illustrated that it is possible to optimally judge and terminate further iterations that would be considered unnecessary in reducing the dirty pages. In comparison with statically setting the number of iterations to 29 by default, the standard deviation tracking showed a marked improvement of 58.01% in the iterative copying time through the design framework implemented in the pre-copy algorithm.

- It was demonstrated that the primary contributing factor to the duration of the migration time was the VM size and the amount of available bandwidth to carry out the live migration. In a gigabit link network, it was shown that the first iteration took as much as 86% of the migration time.
- Coding in C++ was successfully undertaken to implement the envisioned design concept by customizing the pre-copy algorithm of the Xen hypervisor.
- The extensive experiments using the CPU credit scheduler cap highlighted the effect resource management has on real time services and in particular voice traffic quality.

Service providers use live migration primarily for high availability of their services and for fault tolerance purposes. The design solution proposed in this research is best suited for a scenario where the hosted service is running at its peak demand and needs to be live migrated to another host in order to maintain the SLA agreed upon. This implies that during the live migration, the dirty pages that are being generated as the hosted service runs at its peak, will exceed the bandwidth available. By tracking these dirty pages and terminating the process earlier means less iterations are carried out. Since the live migration process is resource intensive, the benefit of fewer iterations translates to more resource availability to other co-hosted services that would otherwise have been affected negatively during the process.

One of the drawbacks with the proposed standard deviation tracking scheme is its insensitivity to very low workloads. For very low workloads or low dirty page rates, the available bandwidth would be sufficient to transfer and reduce the dirty pages at the source within less than 29 iterations. This was shown to be the case for the experiments that had credit scheduler caps set below 35. The tracking scheme would not be suitable as it requires that all 29 iterations be present for the scheme to detect a region of stability.

6.2. Future Work Recommendations

The initial literature review carried out in this thesis as well as the implementation of the concepts discussed within the body of this work, gave rise to prospective areas that may be of further research interest.

The scope of this work considered performing live migration within a LAN which meant that all the experiments were carried out in a controlled and isolated environment. Consideration should be made in assessing live migration across a Wide Area Network (WAN) as well to assess the process when the variables are dynamic and not directly controllable.

The setting of the credit scheduler cap was a static process in which it was manually set prior to triggering the live migration process. In future, a scenario may be considered in which the standard deviation tracking would determine the cap to set dynamically based on a desired outcome such as a pre-determined downtime or QoS parameter. This may be achieved by introducing in the control code a periodic check of the CPU cap at each iteration. It may then be increased or decreased in real-time accordingly to suit a desired outcome.

With the ever growing interest in cloud related services, Xen also includes a cloud platform known as XCP (Xen Cloud Platform) that may be used to setup a private or public cloud for research purposes. XCP includes a Xen hypervisor, a Xen API as well as integrated networking and storage for cloud based services.

Although the research focused primarily on the live migration of VoIP, it would also be of interest to assess the impact of the proposed design framework for a wider range of services such as video streaming or even non real-time services.

References

[1] P. G. Mell T. The NIST Definition of Cloud Computing. [Online]. *2011(April 14)*, 2009. Available: http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf.

[2] Minqi Zhou, Rong Zhang, Dadan Zeng and Weining Qian, "Services in the cloud computing era: A survey," in *Universal Communication Symposium (IUCS), 2010 4th International*, 2010, pp. 40-46.

[3] M. Adolph, E. Sutherland and A. Levin. Distributed computing: Utilities, grids & clouds - ITU-T technology watch report 9 Telecommunication Standardization Policy Division. 2009-04-01[Online]. Available: http://www.itu.int/oth/T2301000009/en.

[4] ITU-T Study Group 13: Question 28/13 – Cloud Computing Resource Management and Virtualization. [Online]. *2012(04/24)*, Available: http://www.itu.int/ITU-T/studygroups/com13/sg13-q28.html.

[5] M. P. Papazoglou and D. Georgakopoulos, "Introduction: Service-oriented computing," *Commun ACM*, vol. 46, pp. 24-28, Oct, 2003.

[6] E. Castro-Leno and J. He, "IT and business integration through the convergence of virtualization, SOA and distributed computing," in *E-Business Engineering, 2008. ICEBE '08. IEEE International Conference on*, 2008, pp. 615-620.

[7] J. Sahoo, S. Mohapatra and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, 2010, pp. 222-226.

[8] Press Release: Gartner Says Virtualization Will be the Highest-Impact Trend in Infrastructure and Operations Market through 2012. . [Online]. *2011(10/24)*, Available: http://www.gartner.com/it/page.jsp?id=638207.

[9] W. Voorsluys, J. Broberg, S. Venugopal and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Proceedings of the 1st International Conference on Cloud Computing*, Beijing, China, 2009, pp. 254-265.

[10] Bing Wei, Chuang Lin and Xiangzhen Kong, "Energy optimized modeling for live migration in virtual data center," in *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, 2011, pp. 2311-2315.

[11] T. Wood, P. Shenoy, A. Venkataramani and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proceedings of the 4th USENIX Conference on Networked Systems Design \&\#38; Implementation*, Cambridge, MA, 2007, pp. 17-17.

[12] D. Marshall, W. A. Reynolds and D. McCrory , *Advanced Server Virtualization : VMware and Microsoft Platforms in the Virtual Data Center.* Boca Raton: Auerbach Publications, 2006.

[13] v. W. Hagen, *Professional Xen Virtualization.* Indianapolis: Wiley Publishing, 2008.

[14] M. Chebiyyam, R. Malviya, S. K. Bose and S. Sundarrajan. Server Consolidation: Leveraging the Benefits of Virtualization. *7(1),* 2009.

[15] D. Patnaik, A. S. Krishnakumar, P. Krishnan, N. Singh and S. Yajnik, "Performance implications of hosting enterprise telephony applications on virtualized multi-core platforms," in *Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications,* Atlanta, Georgia, 2009, pp. 8:1-8:11.

[16] Hanfei Dong, Qinfen Hao, Tiegang Zhang and Bing Zhang, "Formal discussion on relationship between virtualization and cloud computing," in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2010 International Conference on,* 2010, pp. 448-453.

[17] C. N. Hoefer and G. Karagiannis, "Taxonomy of cloud computing services," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE,* 2010, pp. 1345-1350.

[18] Amazon Elastic Compute Cloud - Amazon EC2. [Online]. *2012(04/23),* Available: http://aws.amazon.com/ec2/.

[19] Xen. [Online]. *2012(04/24),* Available: http://www.xen.org/.

[20] T. J. Bittman, G. J. Weiss, M. A. Margevicius and P. Dawson. Magic Quadrant for x86 Server Virtualization Infrastructure. [Online]. *2012(10/31),* Available: http://www.gartner.com/technology/reprints.do?id=1-1B2IRYF&ct=120626&st=sg.

[21] Kernel-Based Virtual Machine (KVM). [Online]. *2012(04/24),* Available: http://www.linux-kvm.org/page/Main_Page.

[22] Microsoft Hyper-V. [Online]. *2012(04/24),* Available: http://www.microsoft.com/en-us/server-cloud/hyper-v-server/default.aspx?hdrFo=mthdr02.

[23] How can I Install Xen 3.4.2 from Source Code? [Online]. Available: http://old-list-archives.xen.org/archives/html/xen-devel/2011-07/msg01020.html.

[24] A. Silberschatz, P. B. Galvin and G. Gagne, *Operating System Concepts.* New Jersey: John Wiley & Sons, 2005.

[25] S. Akoush, R. Sohan, A. Rice, A. W. Moore and A. Hopper, "Predicting the performance of virtual machine migration," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on,* 2010, pp. 37-46.

[26] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam and M. Rosenblum, "Optimizing the migration of virtual computers," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, Massachusetts, 2002, pp. 377-390.

[27] E. Zayas, "Attacking the process migration bottleneck," in *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, Austin, Texas, United States, 1987, pp. 13-24.

[28] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design \& Implementation - Volume 2*, 2005, pp. 273-286.

[29] B. Jiang, B. Ravindran and C. Kim, "Lightweight live migration for high availability cluster service," *Stabilization, Safety, and Security of Distributed Systems*, pp. 420-434, 2010.

[30] Y. Wu and M. Zhao, "Performance modeling of virtual machine live migration," in *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, 2011, pp. 492-499.

[31] Bolin Hu, Zhou Lei, Yu Lei, Dong Xu and Jiandun Li, "A time-series based precopy approach for live migration of virtual machines," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, 2011, pp. 947-952.

[32] M. Atif and P. Strazdins, "Optimizing live migration of virtual machines in SMP clusters for HPC applications," in *Network and Parallel Computing, 2009. NPC '09. Sixth IFIP International Conference on*, 2009, pp. 51-58.

[33] Zhaobin Liu, Wenyu Qu, Weijiang Liu and Keqiu Li, "Xen live migration with slowdown scheduling algorithm," in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2010 International Conference on*, 2010, pp. 215-221.

[34] M. Nelson, B. Lim and G. Hutchins, "Fast transparent migration for virtual machines," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, Anaheim, CA, 2005, pp. 25-25.

[35] 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 *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, USA, 2003, pp. 164-177.

[36] M. R. Hines, U. Deshpande and K. Gopalan, "Post-copy live migration of virtual machines," *SIGOPS Oper.Syst.Rev.*, vol. 43, pp. 14-26, jul, 2009.

[37] H. Jin, W. Gao, S. Wu, X. Shi, X. Wu and F. Zhou, "Optimizing the live migration of virtual machine by CPU scheduling," *Journal of Network and Computer Applications,* vol. 34, pp. 1088-1096, 7, 2011.

[38] Asterisk - the Open Source Telephony Projects. [Online]. *2012(03/21),* Available: http://www.asterisk.org/.

[39] Postfix. [Online]. *2011(10/12),* Available: http://www.postfix.org/start.html;.

[40] SIPp. [Online]. *2011(11/12),* Available: http://sipp.sourceforge.net/.

[41] Xstress - SMTP Stress Tool. [Online]. 2011(11/11), Available: http://xstress.sourceforge.net/.

[42] Wireshark. [Online]. *2012(8/12),* Available: http://www.wireshark.org/download/docs/user-guide-us.pdf;.

[43] VoIPmonitor Manuals. [Online]. *2012(05/10),* Available: http://www.voipmonitor.org/downloads/VoIPmonitor-Sniffer-manual-v5.3.pdf.

[44] Geany. [Online]. *2012(09/20),* Available: http://www.geany.org/Main/About;.

[45] C. Takemura and S. L. Crawford, *The Book of Xen.* San Francisco: No Starch Press, 2009.

[46] Xen Networking. [Online]. *2012(02/11),* Available: http://wiki.kartbuilding.net/index.php/Xen_Networking.

Appendix A: Installing Xen from Source

In this section, the installation process of Xen is discussed. This is obtained from online resources and a Xen book [45].

Download CentOS 5.8 from http://mirror.ufs.ac.za/centos/5.8/isos/i386/.

Install the OS using the normal procedure as instructed. When requested about which packages to include, tick the "Virtualization" package. This will install the default Xen hypervisor which is shipped with the Linux distribution. In addition to the above, the following packages should be installed:

Development:

- gnome software development
- development libraries
- development tools
- legacy software development
- virtualization

Base System:

- legacy software development
- system tools
- admin tools
- x windows system

When this is complete, you will have to restart the computer as instructed.

To install the hypervisor from source code, go to http://www.gitco.de/repo/src/ and install xen-3.4.3-4.el5.src.rpm. Follow the below instructions as stated on a Xen-devel mailing list [23]

1. Setup the rpm build environment.

```
#mkdir -p ~/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
#echo '%_topdir %(echo $HOME)/rpmbuild' > ~/.rpmmacros
#yum install transfig tetex-latex texi2html libaio-devel iasl xz-devel
e4fsprogs-devel rpm-build redhat-rpm-config
```

2. install xen-<version>.src.rpm to get .spec file.

```
#rpm -i xen-<version>.src.rpm
```

This will create xen-3.4.3.spec file under /root/rpmbuild/SPECS/

3. go to /root/rpmbuild/SPECS/ and execute following command to build the rpm sources

```
# rpmbuild -ba xen-3.4.3.spec
```

this may report some missing dependencies. Please do "yum install" of those and do "rpmbuild" again.

4. Once this is complete, the entire source code tree will be unpacked and located in /root/rpmbuild/BUILD/xen-3.4.3 directory.
5. ***This is the point where any modifications to the source code can be made prior compiling and installing the hypervisor. You may use a text editor such as Geany to modify sections of the code as required.***
6. Once this is complete, go to /root/rpmbuild/BUILD/xen-3.4.3/ and install xen and tools.

```
[root@localhost xen-3.4.3]# make install-xen install-tools
```

7. This will then run the compiler and install the modified Xen kernel. During start up it will now be listed on the grub entry along with the initially shipped hypervisor installed in the previous section.

Appendix B: Networking for Virtual Machines

In this research the routed network setup was opted as it gave the flexibility of assigning the VMs IP addresses in the same subnet as the host. This is done by un-commenting the following line as shown below in the */etc/xen/xend-config.sxp* configuration file. The other options available are bridged network or NATd network.

```
(network-script network-route)
(vif-script vif-route)
```

The following wiki [46] gives instructions on the type of available scenarios one may use to setup their VMs networking. In this research the two-routed scenario was chosen.

1. Configure xend-config.sxp

```
vi /etc/xen/xend-config.sxp
(network-script network-route)
#(network-script network-bridge)
#(network-script network-nat)

(vif-script vif-route)
#(vif-script vif-bridge)
#(vif-script vif-nat)
```

2. Configure vm2 configuration file which is located in /var/lib/xend/domain/. Locate the section that deals with networking denoted by ‘vif’. Change the ‘ip’ section to the static IP address you would like to assign the VM. In this example it is 10.10.10.10.

```
(vif
( ip 10.10.10.10 )
```

3. Configure dom0 for forwarding packets of the VMs as shown below.

```
echo 1 > /proc/sys/net/ipv4/conf/all/proxy_arp
```

4. Configure /etc/network/interfaces on the VM itself to match the IP address that was set in the configuration file e.g. in this case the VM was assigned IP address 10.10.10.10. This may be done from the network management of the desktop environment or through the command line as shown below.

```
vi /mnt/etc/network/interfaces

auto eth0
iface eth0 inet static
address 10.10.10.10
netmask 255.255.255.0
```

Appendix C: iSCSI Storage setup

iSCSI (internet Small Computer Systems Interface) storage stores the VM's filesystem as a file image and offers them to both hosts. This is imperative for the live migration as it ensures a degree of robustness in comparison to having the filesystem on one of the hosts. In iSCSI terminology, the dedicated PC holding the filesystems of the VMs is known as the target whereas the dom0 of both hosts as the initiators. The dom0 then offers the shared filesystem to the VM as storage location [45].

iSCSI Server Setup

The target setup can be downloaded from http://sourceforge.net/projects/iscsitarget/

1. Download the target software from the website and unzip it:

```
# tar xzvf Desktop/iscsitarget-0.4.16.tar.gz
# cd iscsitarget-0.4.16
```

2. Then proceed to build all components using the 'make' process.

```
# make
# make install
```

3. The main config file is located at /etc/ietd.conf. The target section defines an iSCSI Qualified Name (iqn) which will be used to uniquely identify the different storage filesystems that will be exported to the dom0 and their respective VMs.

   e.g. target: iqn.2011-08.server.san.storage.img.vm.3driver

   e.g. target: iqn.2011-08.server.san.storage.img.vm.2

   Lun 0 Path=/nfs/home/phillip/images/vm3driver.img,Type=fileio

4. Now the iSCSI devices can be exported:

```
# service iscsi restart
```

iSCSI Client Setup

For the hosts to have access to the target an initiator is required which may be obtained from http://www.open-iscsi.org/. Both Red Hat and Debian have versions in their package manager which may be used. These are iscsi-initiator-utils and open-iscsi, respectively. Once the initiator has been downloaded and installed, work needs to be done to allow the dom0 to mount the iSCSI devices at boot time such that the VMs are able to access them.

1. Using the *iscsiadm* command one is able to perform target discovery (PC with IP address 10.10.10.4) and login.

```
# iscsiadm --mode discovery --type sendtargets --portal 10.10.10.4
```

The output of the command is:

```
iqn.2011-08.server.san.storage.img.vm.3driver, portal: 10.10.10.4,3260
iqn.2011-08.server.san.storage.img.vm.2, portal: 10.10.10.4,3260
```

2. Restart iscsid to propagate your changes.

```
#service  iscsid restart.
```

Now that the host has access to the exported iSCSI devices, one may proceed to install the VM in these filesystems whose paths are at the following location.

```
#/dev/disk/by-path/
```

Appendix D: SIPp Monitoring Screens

SIPp monitors SIP traffic and displays information on the following screens:

- Scenario screen
- Statistics screen
- Repartition screen

Scenario Screen

Once SIPp is started this is the first screen that appears showing the call-rate and the

```
[root@localhost sipp.svn]# ./sipp -sn uac -d 20000 -s 2002 10.10.10.12 -m 1000
Resolving remote host '10.10.10.12'... Done.
------------------------------ Scenario Screen -------- [1-9]: Change Screen --
  Call-rate(length)   Port   Total-time  Total-calls  Remote-host
10.0(20000 ms)/1.000s   5060     120.01 s         1000  10.10.10.12:5060(UDP)

  Call limit reached (-m 1000), 0.000 s period  0 ms scheduler resolution
  0 calls (limit 600)                  Peak was 201 calls, after 20 s
  0 Running, 332 Paused, 0 Woken up
  0 dead call msg (discarded)          0 out-of-call msg (discarded)
  1 open sockets

                                Messages  Retrans   Timeout   Unexpected-Msg
     INVITE ---------->         1000      0         0
        100 <----------         1000      0         0         0
        180 <----------         0         0         0         0
        183 <----------         0         0         0         0
        200 <----------  E-RTD1 1000      0         0         0
        ACK ---------->         1000      0
      Pause [    20.0s]         1000                          0
        BYE ---------->         1000      0         0
        200 <----------         1000      0         0         0

------------------------------ Test Terminated --------------------------------
```

Figure D-1: Scenario Screen.

messages that have been transmitted thus far. This screen displays real-time data as the SIP messages are being transmitted and also takes note of any retransmissions and timeouts that occur during the duration of the calls.

The SIP responses are as a result of SIP requests such as INVITE, BYE and ACK in this case. The response codes shown in Fig.D-1 are as a result of the corresponding requests and have the following meaning:

100	Trying
180	Destination has received INVITE request and user is being alerted of the call.
183	Session is in progress.
200	Request was successful.

Table D-10: SIP response codes.

Statistics Screen

This screen is accessed by pressing the numerical key '2'. A cumulative counter is kept of all statistics pertaining to the calls.

- Start Time: Date and time when the test has started.
- Last Reset Time: Date and time when periodic counters where last reset.
- Current Time: Date and time of the statistic row.
- Elapsed Time: Elapsed time.
- Call Rate: Call rate (calls per seconds).
- Incoming Call: Number of incoming calls.
- Outgoing Call: Number of outgoing calls.
- Total Call Created: Number of calls created.
- Current Call: Number of calls currently ongoing.

- Successful Call: Number of successful calls.
- Failed Call: Number of failed calls (all reasons).
- Response Time: Time between two SIPp commands (send, recv or nop).

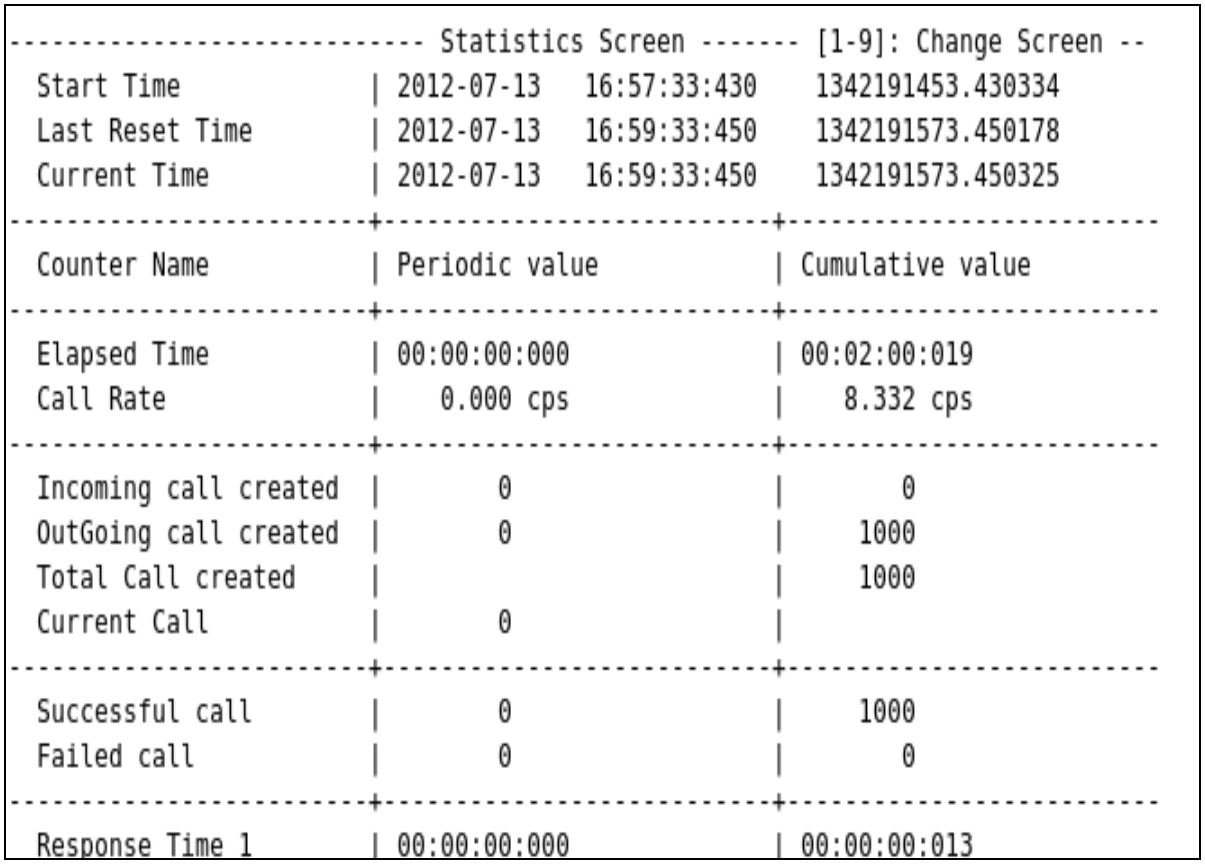


Figure D-2: Statistics Screen.

Repartition Screen

This screen is accessed by pressing the numerical key ‘3’. The response time and call length distribution is logged on this screen.

```
---------------------------- Repartition Screen ------- [1-9]: Change Screen --
  Average Response Time Repartition 1
           0 ms <= n <        10 ms :        326
          10 ms <= n <        20 ms :        558
          20 ms <= n <        30 ms :         95
          30 ms <= n <        40 ms :         20
          40 ms <= n <        50 ms :          0
          50 ms <= n <       100 ms :          1
         100 ms <= n <       150 ms :          0
         150 ms <= n <       200 ms :          0
                  n >=       200 ms :          0
  Average Call Length Repartition
           0 ms <= n <        10 ms :          0
          10 ms <= n <        50 ms :          0
          50 ms <= n <       100 ms :          0
         100 ms <= n <       500 ms :          0
         500 ms <= n <      1000 ms :          0
        1000 ms <= n <      5000 ms :          0
        5000 ms <= n <     10000 ms :          0
                  n >=     10000 ms :        941
------- Waiting for active calls to end. Press [q] again to force exit. -------
```

Figure D-3: Repartition Screen.

Appendix E: Accompanying CD-ROM

The CD-ROM contains the following directories:

Research Literature contains all the relevant reading material used in the course of this research work.

Software contains setup files required to install the monitoring and traffic generation software used in this research. The modified pre-copy algorithm is located in *xc_domain_save* sub-folder and the source RPM of the Xen hypervisor is contained in its respective folder.

Thesis contains an electronic copy in PDF format of the thesis.

Results contain all the results obtained during the course of the thesis and appended to the final thesis in Microsoft Excel format. Also it contains screen capture videos from the research.