Minimizing Migration Time of Virtual Machines by Serializing & Compressing Memory Pages in Xen Hypervisor

A Dissertation Proposal
submitted
By

Deeksha Abbi

to

Department of Computer Science and Engineering

In the partial fulfillment of the Requirement for the

Award of the Degree of

Master of Technology in Computer Science and Engineering

Under the guidance of

Pushpendra Kumar Pateriya

(May 2015)

Abstract

Cloud computing is an emerging research area that builds upon the foundation of various fields like service-oriented architecture, grid computing, utility computing, and virtualization technology. It offers various service models including infrastructure-as-a-service based on on-demand computing and pay-per-use models to the consumers. The base of this idea lies in provisioning infrastructure resources at the data centers. This provisioning is time-consuming and expensive. In order to efficiently provision resources, cloud employs two core services, namely, virtual machine provisioning service and virtual machine migration service. These are the main aspects of cloud computing explored in this report, along with some cost models, service-level agreements and the implementation of live migration on the Xen hypervisor.

A rule-based resource manager which allocates resources based on the priority of the incoming requests and a novel migration algorithm which aims to reduce the total migration time of a virtual machine by decreasing the amount of pages that are dirtied during transfer and in turn, make resource provisioning more effective and efficient. This is achieved by serializing the contents of the main memory and also compressing them reducing the number of dirty pages and the size of the data to be transferred over the network, leading to better resource utilization.

Acknowledgement

I would like to take this opportunity to express my deep sense of gratitude to all who helped me directly or indirectly during this dissertation work.

Firstly, I would like to thank my supervisor, **Mr. Pushpendra Kumar Pateriya**, for being a great mentor and the best adviser I could ever have. His advise, encouragement and critics are source of innovative ideas, inspiration and causes behind the successful completion of this dissertation. The confidence shown on me by him was the biggest source of inspiration for me. It has been a privilege working with him this past year.

I am highly obliged to all the faculty members of computer science and engineering department for their support and encouragement.

I would like to express my sincere appreciation and gratitude towards my friends for their encouragement, consistent support and invaluable suggestions at the time I needed the most.

I am grateful to my family for their love, support and prayers.

-Deeksha Abbi

Declaration

I hereby declare that the dissertation proposal entitled, "*Minimizing Migration Time of Virtual Machines by Serializing & Compressing Memory Pages in Xen Hypervisor*", submitted for the M.Tech degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date: May 04, 2015

(Deeksha Abbi)
Investigator Regn. No.:
11000994

Certificate

Certified that the work contained in the dissertation report titled "*Minimizing Migration Time of Virtual Machines by Serializing & Compressing Memory Pages in Xen Hypervisor*", by Deeksha Abbi, Reg.No. 11000994 has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Date: May 2015

(Mr. Pushpendra Kumar Pateriya)
Computer Science Dept.
L.P.U., Phagwara

Contents

List of Figures

List of Tables

Listings

List of Algorithms

Chapter 1
Introduction

1.1 Theoretical Foundation & Evolution of Cloud

Cloud computing aims at allowing access to a large number of resources on-demand using a virtualized single system view. [Buyya, Broberg (2011)] define it as,"a parallel and distributed computing system that consists of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified resources based on service level agreements." The important characteristics of a cloud-based service are:

- *On-demand self-service*: Cloud services can be used without any human intervention with the service provider, through a self-service interface.
- *Flexible resource provisioning*: Resources are dynamically assigned and reassigned on user demand. They can be scaled up or down accordingly.
- *Metered service*: Cloud uses the pay-per-use model where the users are billed in accordance with the usage of the service and don't have any long-term commitment with the provider.

Cloud computing arose from the growth in several other technologies that converged and contributed to the advent of cloud. The technologies that form the base of cloud computing are explained below.

1.1.1 Service Oriented Architecture

A service can be defined as a self-contained unit of functionality, which can exchange information with other services without the need of any human interaction. In a Service Oriented Architecture (SOA), software resources are packaged as services, which are well-defined, self-contained modules that provide functionality and are independent of the state or context of other services.

1.1.2 Grid Computing

Grid is a form of distributed computing in which each node is assigned a specific task. It uses middleware to divide and distribute pieces of a program among its nodes.

1.1.3 Utility Computing

Commonly referred to as the 'pay-per-use 'model, it is a service provisioning model in which users provide a utility value to their jobs based on various Quality of Service (QoS) constraints (deadline, satisfaction, importance) and are willing to pay a service provider to fulfill their demands [Buyya, Broberg (2011)]. Utility computing offers resources as a metered service. The main feature of this model is the fact that there is low or no initial cost to acquire the resources, as they are rented and after the fulfillment of the service of one user, can be provisioned to other users.

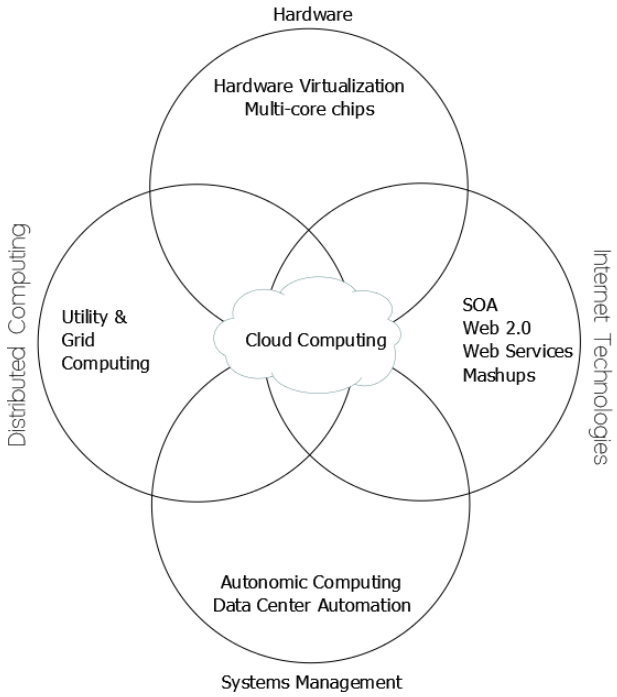


Figure 1.1: Convergence of various advances leading to the advent of cloud computing [Buyya, Broberg (2011)].

1.1.4 Hardware Virtualization

Virtualization is the abstraction of computing resources, mainly, storage, processing power, memory, and, network or I/O. It is similar to emulation but not the same as,

in emulation, a system pretends to be another system, whereas in virtualization it pretends to be two or more of the same system. Hardware virtualization allows multiple OS and software stacks to run on a single physical platform, increasing resource utilization from 20%, in traditional servers to 70% in virtualized servers as mentioned in [Imteyaz (2010)].

1.1.5 Layers of Cloud

Cloud services are divided into the following three layers, according to the levels of abstraction provided - Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).

1.1.5.1 Infrastructure as a Service (IaaS)

Offering virtualized resources (computation, storage, network) on demand is know as Infrastructure as a Service. Example, Amazon Elastic Compute Cloud (EC2) and Rackspace provide resizable resources on a pay-per-use basis.

1.1.5.2 Platform as a Service (PaaS)

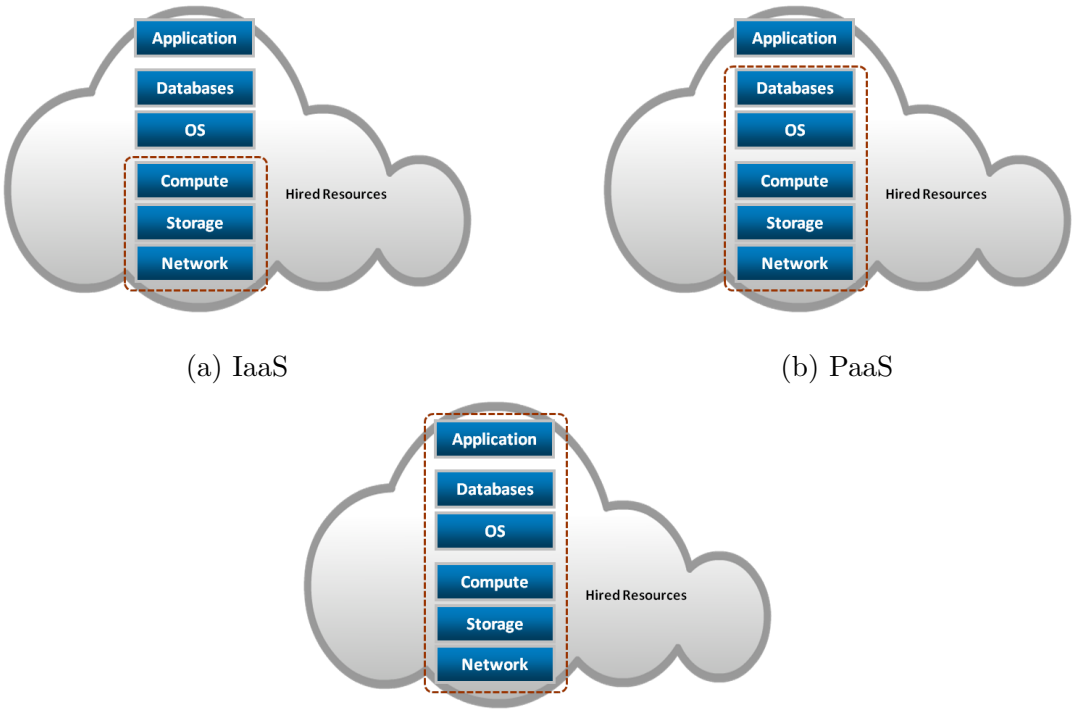
A cloud platform offers an environment to the developers on which they can create and deploy applications without needing to know how much memory or how many processors the application will be using. Google App Engine (GAE) and Microsoft Azure offer scalable environments for developing and hosting applications.

1.1.5.3 Software as a Service (SaaS)

Applications reside on top of the cloud stack. Users can access services provided by this layer via web browsers. This model reduces the burden of software maintenance for customers and makes development and testing easy for the service providers. That is why, majority of the consumers are shifting from locally installed programs to online software services. Example, Microsoft Office 365, Salesforce.com is a Software-as-a-Service solution for Customer-Relationship-Management (CRM) applications.

1.1.6 Types of Cloud

A cloud can be classified as public, private, hybrid or community, based on the model of development,



(a) IaaS

(b) PaaS

(c) SaaS

Figure 1.2: Cloud service models.

1.1.6.1 Public cloud

A cloud made available in a pay-as-you go manner to the general public. It is available on a subscription basis (pay as you go).

1.1.6.2 Private cloud

An internal data center of a business or other organization, not made available to the general public.

1.1.6.3 Hybrid cloud

A hybrid cloud is formed when a private cloud is scaled up by using resources from the public clouds. This is usually done to handle the increase in load, a phenomenon know as "*cloud bursting*".

1.1.6.4 Community cloud

In a community cloud, infrastructure is shared between organizations with a common interest, which can be managed internally or externally. The size is less than that of a public cloud, but more than that of private.

1.2 Dimensions

As a research area, cloud computing presents numerous fields to work upon, some of them being: resource provisioning, load balancing, quality management, security and privacy, and green cloud computing. Cloud aims at providing services on-demand to the consumers. These services utilize the underlying system resources which use virtualization so that they can be transparently accessed by multiple application users at the same time. Thus, the core functionality of cloud computing lies in the efficient allocation of resources.

This report focuses on the research done till date in the field of dynamic resource provisioning in cloud computing, with focus on four parameters. First, virtualization of resources and live migration of Virtual Machine (VM) which lets us allocate resources on the servers and overcome resource under-utilization and over-utilization problems, helps balance the load, and save energy by server consolidation.

Second, the various approaches for calculating the charges incurred by the consumer for using cloud services are examined. Cloud, being a utility-based computing, users only need to pay for the time they spend accessing the service, needs better revenue approaches so that users can also benefit.

Third, resource provisioning should be done keeping in mind the service level agreements. The user will be compensated for any additional time the service is down as mentioned in the SLAs.

Finally, we explore the Xen hypervisor, an open-source bare-metal Virtual Machine Monitor (VMM), responsible for the creation and management of VM and the present techniques for live migration support in Xen. Our work is focussed on improving the migartion time in Xen and providing faster resource allocation.

Chapter 2
Literature Survey

The related work is divided into four subcategories as follows: first section explores virtualization and virtual machine migration techniques for resource provisioning, second section is for the cost model, third section discusses approaches for Service Level Agreements (SLA) and finally, the fourth section follows the work done on Xen hypervisor.

2.1 Virtualization and Virtual Machine Migration

A key challenge faced by IaaS providers when building a cloud infrastructure is managing the physical and virtual resources and to provision them rapidly and dynamically as per the demand. Extensive research has been done in this field with various ways to improve provisioning. The following paragraphs provide a survey of the work done with regards to migration of VM and resource provisioning in cloud computing.

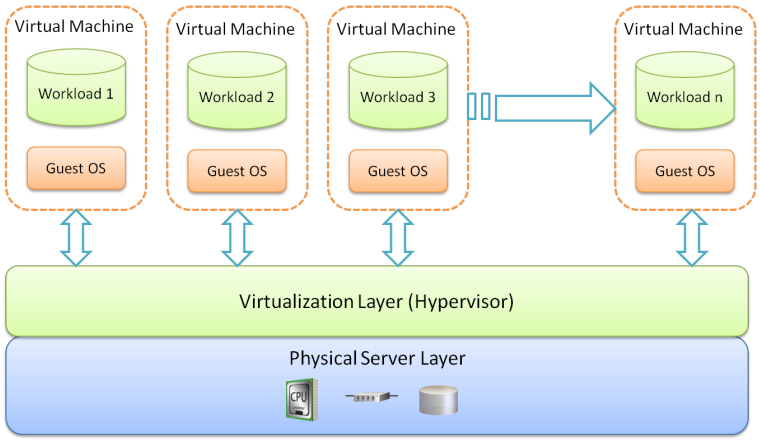


Figure 2.1: A layered virtualization technology architecture [Buyya, Broberg (2011)]

The core services which help the users get the best out of any cloud computing model are the VM provisioning services and migration services. The whole concept of allocating resources in cloud revolves around virtualization. When a user needs a resource, namely computation, storage, network, or any other, it is done so by provisioning a virtual server through a self-service interface. To start off, we begin with understanding the concept of

virtualization. Virtualization is defined as the abstraction of the four computing resources namely, computation (CPU), storage (HDD), memory (RAM) and network or I/O.

The VMM or Hypervisor, partitions the physical resources of a machine into multiple VM which are used to fulfill the allocation request of the users. They are also easy to manage and the resources are utilized efficiently. Changes in the workload conditions of a VM lead to the migration of VM by the hypervisor (depending on certain conditions) as described by [Mishra et al. (2012)]. There are two models of migration present - live migration (pre-copy and post-copy memory migration) and cold migration, but the one that is most generally preferred is *Iterative/Pre-copy Live Migration* [Buyya, Broberg (2011), Clark et al. (2005)]. Live Migration is preferred over cold migration because the memory pages are copied first and then the VM is stopped on the source and restarted at the destination with an overall downtime (time between stopping the VM on original host and resuming it on the destination) ranging from milliseconds to seconds. Whereas in cold migration, the VM is powered off firstly and then copied to the destination host, increasing the downtime to a significantly noticeable levels. Pre-copy is preferred over post-copy migration because of the following reasons,

(i) The memory pages are transferred while the VM is still running. If some pages become dirty, they are recopied in the successive iterations as shown in Fig.2.2. Whereas, post-copy memory migration is initiated by suspending the VM at the source. A minimal subset of the execution state is transferred and then the VM is resumed. When the clients access the service, page faults occur which are trapped at the target and redirected to the source.

(ii) Pre-copy retains up-to-date state of the VM at the source during migration but the state is distributed over both the source and destination machines in post-copy. If the destination machine happens to fail, VM can't be recovered in case of post-copy.

VM migration has three major goals as mentioned by [Mishra et al. (2012)], namely:

- **Server consolidation** : Consolidation is done to reduce the server sprawl. Many times the Physical Machine (PM) have underallocated resources, i.e., they are not efficiently utilized - know as coldspots. To avoid this problem, VM migration is done and all VMs from various PMs are hosted onto one or more PMs such that resources are fully utilized. The freed PMs can be switched off to save power.

- **Load balancing** : The goal here is to avoid a situation where there is large difference in the resource utilization levels of PMs. We balance the VMs such that the residual resource capacity on each PM is the same.

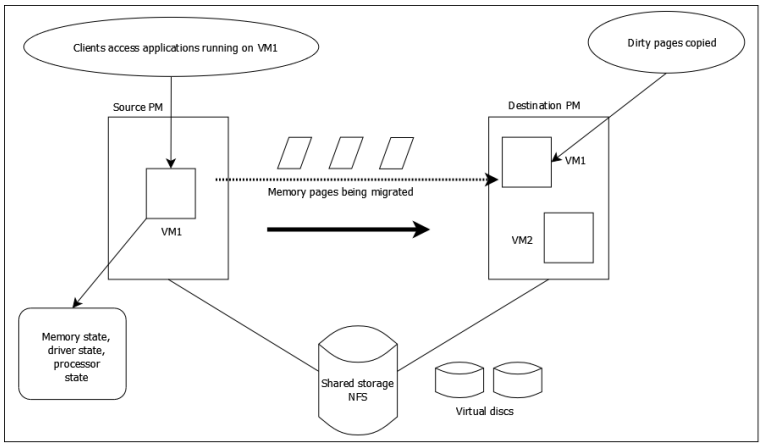


Figure 2.2: Live VM migration procedure [Mishra et al. (2012)]

- **Hotspot mitigation** : Hotspot is a condition in which there are not enough resources present on a PM to fulfill the incoming request for allocation. In such cases, more resources can be provisioned or the VMs are migrated to other hosts to make the required resources available.

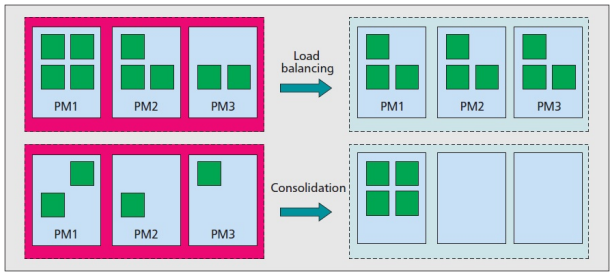


Figure 2.3: Load balancing and consolidation scenarios [Mishra et al. (2012)]

[Imteyaz (2010)] mentions the scaling up of a private cloud by using resources from the public cloud using a rule-based resource manager technique. The resources are allocated based on the priority of incoming requests. A request which performs critical data processing is high priority and one which doesn't is low priority. High priority requests should never leave the private cloud as it may have confidential information. If the private cloud has sufficient resources to fulfill the request, it provides the resources, else any incoming high priority request will cause a VM migration of a low priority request from private to public cloud. The proposed resource manager is shown in fig. 2.4.

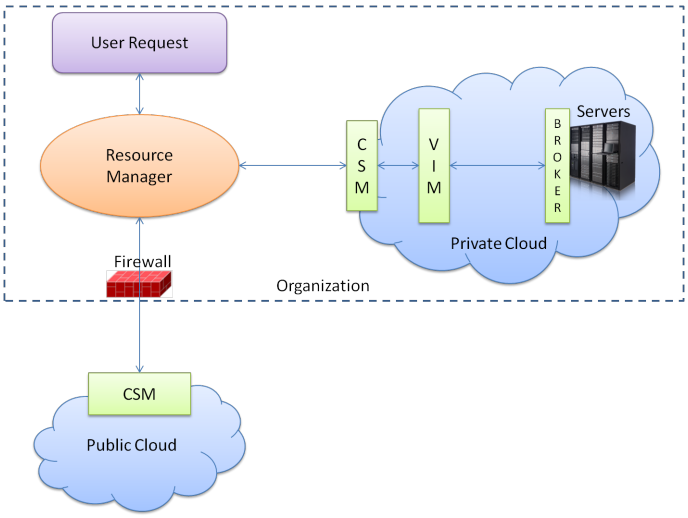


Figure 2.4: Resource Manager [Imteyaz (2010)].

[Rodriguez, Buyya (2014)] present the execution of workflow in cloud to be composed of two steps,

(i) Resource provisioning phase

(ii) Scheduling phase, where each task is mapped to a best-suited resource.

The problem is defined as finding a schedule to execute a workflow such that, the execution cost is minimized and the deadline is met, which can be formally written as,

$$\text{Minimize } TEC$$
$$\text{subject to } TET \leq \delta_W.$$

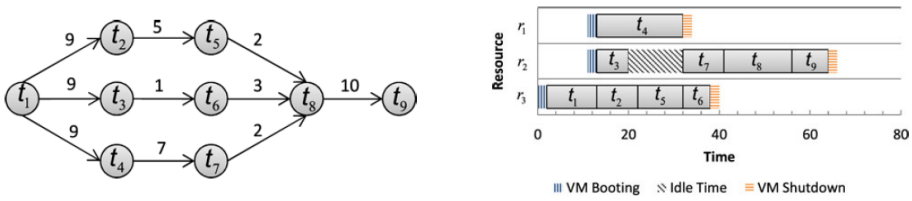
where TEC = Total execution cost, TET = Total execution time, and δ_W = deadline of work.

The proposed model, a meta-heuristic optimization technique based on particle swarm optimization (PSO), the workflow application is modeled as a directed acyclic graph (DAG) such that the nodes represent the tasks ($T = \{t_1, t_2, ..., t_n\}$) and the edges ($E$) represent the data dependency between two tasks, shown in figure 2.5a. This dependency is presented in the form of a parent-child relationship i.e.,

if t_i and t_j are dependent (an edge exists) then,

$t_i \leftarrow$ parent task

$t_j \leftarrow$ child task



(a) A sample workflow.

(b) Example of a schedule generation.

Figure 2.5: Deadline based resource provisioning and scheduling.

Applications hosted on the cloud can automatically increase or decrease the amount of resources used by them. This process, known as auto-scaling, is possible because of the virtual nature of the resources offered to the applications. This concept is very useful during peak hours, when the amount of users suddenly increases (cloud bursting), and the resources need to be scaled up to keep up with the demand. [Calcavecchia et al. (2012)] proposed a self-organizing, self-adapting and fully decentralized solution known as DEPAS (DEcentralized Probabilistic Algorithm for Auto-Scaling), where the dependency on the central cloud provider is removed. The proposed architecture consists of the elements shown in figure 2.6,

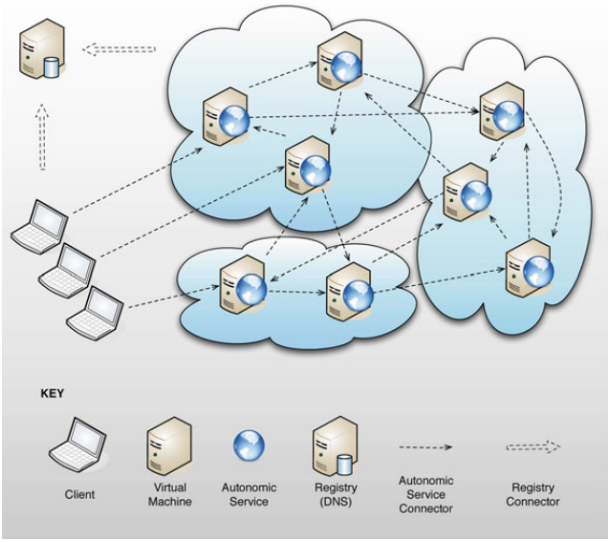


Figure 2.6: Architecture that supports the implementation of DEPAS [Calcavecchia et al. (2012)].

[Wuhib et al. (2013)] present a distributed middleware architecture and use gossip protocol to dynamically adapt the allocation to changes in load, ensure fair allocation among sites and scale the number of physical machines and sites. The work focuses on

the resource manager component with only two types of resources - CPU and memory, assuming that all machines belong to a single cluster. The problem involves placing identical instance of modules on machines and allocating resources to these modules such that the cloud utility is maximized. The protocol for distributed resource allocation works as follows,

(a) A round-based gossip protocol is executed, in which each node selects a subset of nodes to interact with.

(b) The nodes interact via small messages which are processed and trigger state changes.

The node interaction follows a *push-pull* paradigm, wherein the updates are sent to other nodes without them asking for it (push) and updates are pulled or requested by a client [Tanenbaum, Steen (2007)].

[Ramachandran et al. (2012)] explored the concept of dynamic provisioning in multi-tenant service clouds by proposing an overall system architecture called *User Interface-Tenant Selector-Customizer* (UTC), which enables cloud-based services to be provisioned as a variation of the existing service tenants in the cloud. Multitenancy is an important feature of cloud computing. Tenants are a group of users sharing the same view of the software they use, but in cloud, the user applications are run on separate VMs. Multi-tenancy helps providers save cost but incurs high reconfiguration costs. The proposed model aims to reduce this cost by reconstructing new tenants from the existing configurations by letting the clients exactly specify their requirements instead of searching and opting for a service from a catalogue provided by the service provider.

2.2 Cost Models

The resources in cloud computing environments are rented for a specific period of time and the user needs to pay for only that specified period. Calculating cost on a per-use basis is an important aspect in cloud and various models have been proposed on how to effectively calculate this cost, minimize it for user benefit and maximize it for service provider benefit.

[Zaman, Grosu (2011)] proposed a *Combinatorial Auction-Based Mechanism* for dynamic VM provisioning in the cloud. The proposed mechanism, called CA-PROVISION, treats the set of resources as liquid resources which can be configured as per the user request into various types of VM instances. The users *bid* on one of such bundles. The problem is called Dynamic Virtual Machine Provisioning and Allocation Problem (DVMPA), which states that m different types of VM instances are there which offer

Table 2.1: Comparison of Resource Provisioning Techniques employing Virtualization & VM Migration.

Parameters	**[Mishra et al. (2012)]**	**[Imteyaz (2010)]**	**[Rodriguez, Buyya (2014)]**	**[Calcavecchia et al. (2012)]**	**[Wuhib et al. (2013)]**	**[Ramachandran et al. (2012)]**
Objective	migration heuristics	scale up private cloud by using public cloud resources	minimize workflow execution cost while meeting deadlines	auto-scaling of services	make a middleware layer which adapts and scales	provisioning by reconfiguring already present tenants
Migration Feature	present	present	not mentioned	present	not mentioned	not mentioned
Provisioning Steps	allocate, migrate if needed	check priority of request and allocate accordingly	resource provisioning phase and schedule generation phase	uses autonomic services for allocation	gossip to select nodes	model client requirements, use customizer to improve matching
Features	reduce hotspots, coldspots and server sprawl, load balancing	security, scaling	cost minimization, meeting deadlines and heterogenity of hardware	load balancing, overlay management	scalable, maximize utility	reduction in matching time

computing resources (CPU) to the user. The user bids and specifies how many number of each VM resources he'd like and also tells the price he is willing to pay for the same. The CA-PROVISION algorithm then calculates a reserve price, which is the minimum amount of price every user has to pay for the resources to be allocated to him by creating a dummy user who is requesting VM instance one. If the price mentioned in the user's bid is lower than this reserve price, the user is not allocated anything, else the resources are allocated and the payment is calculated. The payment is equal to the bid density multiplied by the size of the bundle. The cloud provider's profit is calculated by subtracting the running cost of allocated VMs and cost of remaining idle resources from the revenue generated. The algorithm focuses on maximizing this profit but does not guarantee as to whether it will be maximized and if the allocation will be efficient.

[Jiang et al. (2013)] explored effective cloud capacity planning and instant VM provisioning. The goal of the problem is to reduce the prediction error.

$$E = \sum_{t} f(v^t, \hat{v}^t) \tag{2.1}$$

where E is the prediction error to be minimized, f(.,.) represents an arbitrary cost function that is used to quantify the prediction error, v is the future resource at a future time t in terms of VM units and $\hat{v}$ is the prepared resource at present time t. The proposed solution to this is a Cloud Prediction Cost (CPC) model which is an asymmetric and heterogeneous measure that models the prediction error of two different types of costs: the cost of SLA penalty (caused by underestimation of resources) and the cost of idle resources (caused by overestimation of resources). The total cost is represented as,

$$C = \beta P(v^t, \hat{v}^t) + (1 - \beta) R(v^t, \hat{v}^t) \tag{2.2}$$

where P is the cost of SLA penalty, R is the cost of idle resources and β is used to tune the importance between the two costs.

Another technique for benefit-aware on-demand provisioning approach for multi-tier applications. [Wu at al. (2013)] considers two types of costs,

- Transition cost - the transition latency caused in the VM by reconfiguration actions which ranges from 21 seconds to 105 seconds.
- Infrastructure cost - the rental cost of VM resources for tenants to host applications.

The problem is to decide the best resource provisioning strategy, taking into consideration the above two costs. This is done by the *Investment decision problem*, whereby the best candidate for selection is the one which maximizes the earnings to cost ratio. For this purpose, a balloon cloud platform is proposed whose provisioning engine consists of four components: monitoring & forecaster, evaluator, planner, executor. Using these,

benefit and cost functions for all possible reconfigurations are constructed. The cost and benefit are balanced and the action which minimizes the cost with maximum benefit is chosen. This approach was effective enough in reducing both, cost and SLA violations.

2.3 Service Level Agreements

SLA management is also an area of research in cloud. It provides a framework within which both buyer and seller can have a profitable service business relationship. The two types of SLA - infrastructure and application guarantee the availability of infrastructure like server, network connectivity and response time and the penalty incurred by the provider in case the agreement is not met because of any circumstances.

[Son, Jun (2013)] present an automated SLA negotiation mechanism and a workload and location aware allocation scheme (WLARA). The SLA negotiation mechanism is designed for the service price and response time issues. It allows the agents to make counter-offers (as shown in fig.2.7) to their opponents in alternating rounds until an agreement is reached or an agent's negotiation deadline is reached. The resource allocation strategy works as follows: a provider receives user requests and forwards them to the reservation manager which finds appropriate data center by using the utility function in equation 2.3. Then, the manager asks the coordinator agent, who manages the PMs in the DC, to allocate the request to the lightly loaded PM near the user. This guarantees a reasonable response time to the user.

$$U_m = \alpha.U_m^{UL} + \beta.U^{RT}m \tag{2.3}$$

There are two terms in the utility function: (1) the machine workload $\alpha.U_m^{UL}$, and (2) the expected response time $\beta.U^{RT}m$. Each term is multiplied by the preference weight. If $\alpha = 1$, then $\beta = 0$, the provider emphasizes workload in placing the VM and vice versa.

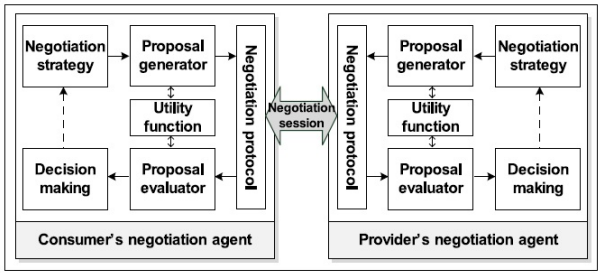


Figure 2.7: An automated negotiation mechanism [Son, Jun (2013)].

Table 2.2: Comparison of various cost models.

Parameters	**[Zaman, Grosu (2011)]**	**[Jiang et al. (2013)]**	**[Wu at al. (2013)]**
Objective	maximizing user utility by bidding and dynamic VM allocation.	capacity planning, instant VM provisioning, predicting future demands.	best resource allocation strategy considering infrastructure and transition costs.
Resource Type	bundles of VM instances	VM unit	VM (multi-tier application host environment)
Allocation Type	one bundle or none	one	one
Resource Provision-ing Parts	allocated if auction won, else bid again	plan capacity, instant provisioning	capacity resizing, add/remove VM replicas, live migration
Type of Cost	cost of running and idle VMs	cost of SLA penalty and resource wastage	transition and infrastructure costs

The idea of volunteer Clouds is explored by [Cuomo et al. (2012)] in their project *Cloud@Home* which aims at building an IaaS service provider by using resources from volunteer contributors. This model is effective from a cost perspective as the resources are borrowed from users not currently needing them, and not purchased. The system architecture is shown in figure 2.8. The main actors in the C@H management are users,

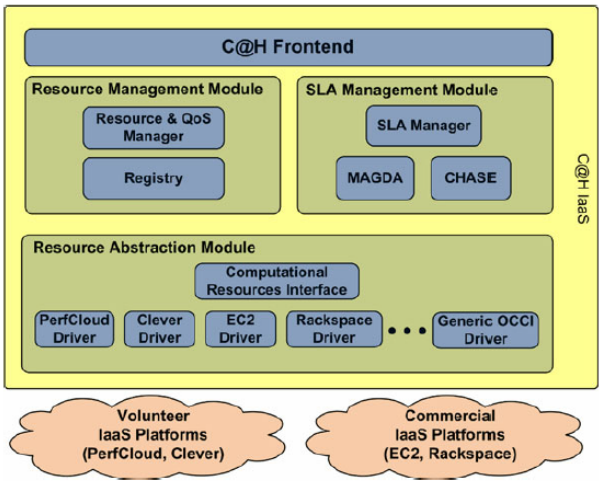


Figure 2.8: The C@H system architecture [Cuomo et al. (2012)].

admins and resource owners. The users interact with the provider and request resources and negotiate desired QoS. The admin manages the infrastructure and decides which services should be activated/deactivated. A resource owner shares its resources with the C@H system. It can be a public or volunteer contributor. The *Resource Abstraction* module provides an abstraction layer by hiding the heterogeneity of the resources from the users. The *SLA Management* module makes sure the SLA are not violated. Currently, C@H addresses only one QoS parameter of resource availability since most of the resources are from volunteer contributors. The core of the system is the *Resource Management* module which is in-charge or provisioning the resources. Lastly, the *Frontend* acts as an interface for the C@H actors. The resource provisioning process is carried out by the RQMCore (Resource and QoS Manager) component. It is an asynchronous event-based system which is responsible for acquiring resources from the providers and ensuring that the negotiated QoS is being met. It has two main tasks,

(1) Request Management

(1.1) Provider Selection: Retrieves a list of subscribed providers.

(2.2) Resource Acquisition: After finding a suitable provider, the drivers are set up for resource allocation.

Table 2.3: Comparison of SLA techniques.

Parameters	**[Son, Jun (2013)]**	**[Cuomo et al. (2012)]**
Objective	SLA-based framework considering workload and geographical location of DCs.	providing QoS and SLA on unreliable, intermittent cloud
Type of SLA Provided	service price and response time	only service availability
Resource Availability	stable, till deallocated	unstable, depends on volunteer customers offering resources
SLA Guarantee	proper working guarantee	no guarantee of fulfillment
SLA Choosing Scheme	negotiation	no user choice

(3.3) Logging: All the operations are logged.

(2) SLA Enforcement

(2.1) Availability Guarantees: The availability of a resource can be described as, $ProviderAvailability = \frac{MTBF}{MTBF+MTTR}; MTTR = T_{fd} + T_{boot}$
where MTBF is mean time between failure,
MTTR is mean time to repair a single resource,
T_{fd} is the time to detect a resource failure and,
RT_{boot} is the time to boot up a new VM as a substitute for the failed one.

(2.2) Alert Reaction: Heartbeat messages are used to verify if a node is alive. The variables HBfail, representing the number of failures and HBsuccess, representing the number of consecutive successful heartbeats, are used.

(2.3) Performance Guarantee

2.4 Xen Hypervisor

[Yu et al. (2014)] proposed a self-adaptive VM allocation technique that takes into account the network resource contention between physical nodes in the Xen virtualization environment. Xen primarily consists of three components,

1. Xen Hypervisor
   (a) It has direct access to hardware resources.
   (b) It schedules virtual CPUs onto physical CPUs.
   (c) It allocates and releases physical memory to/from VMs.
   (d) It supports isolated execution of VMs by executing hyper-call based trap handler codes.
2. Domain-U (Dom-U)
   (a) It is a VM running a guest Operating System (OS) and user applications.
   (b) It has no direct control over resources. The access is only permitted by exported hypercalls from Xen.
3. Domain-0 (Dom-0)
   (a) It is a control VM that supports network and disk I/O operations in Dom-Us.
   (b) Xen uses a split driver approach, where the front end driver lies in Dom-U and the back-end in Dom-0. The back-end driver forwards the requests from the front-end to the native driver and also relay the response back to the front-end.
   (c) Since Dom-0 is the critical path for all I/O requests, it is subject to performance bottleneck.

To overcome the bottleneck problem, a network contention aware job management framework was proposed, which works as follows,

- As new jobs arrive, they are input to the job parser, which analyzes their specifications based on the computing resource (e.g., number of nodes, memory size) needed to fulfill the request and puts them in a job queue.
- The jobs are passed onto the scheduler, which allows the implementation of allocation algorithms. It provides a {host:VMs} map.
- The host manager keeps the information about the I/O degree of Dom-0 for each host. It also collects and manages the network contention information, defined as follows,
  $NETCONT_k = (\omega \times CPU_k) + ((1 - \omega) \times \frac{1}{IDLE_k})$

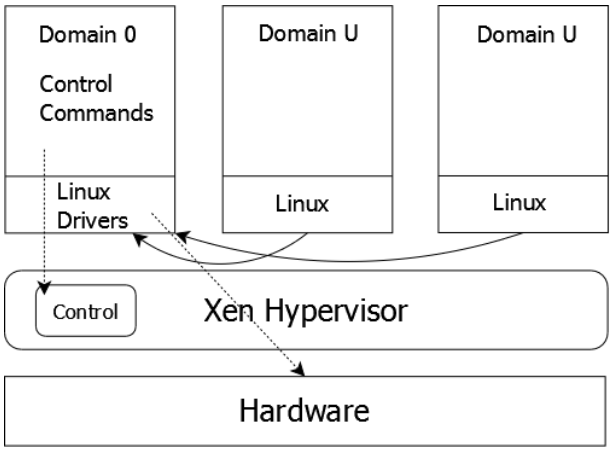


Figure 2.9: Architecture of Xen Hypervisor.

- Upon receiving provisioning request, the VM scheduler who is aware of the NETCONT, greedily selects physical hosts based on it.
- The allocation and booting of VMs is handled by the VM Manager, and the applications are executed on the dynamically allocated virtual clusters with minimum contention. This whole process can be represented as in Figure 2.10,

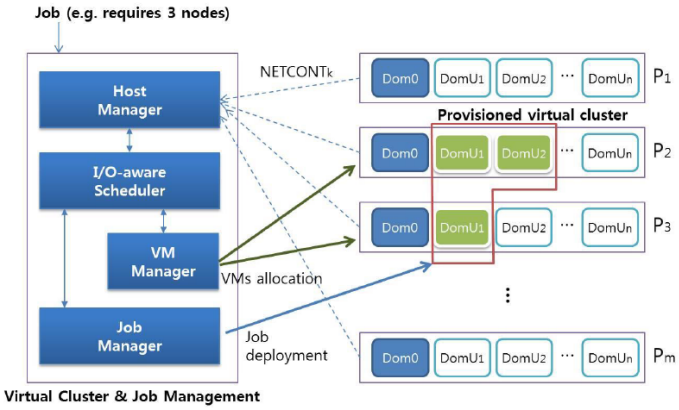


Figure 2.10: Network Contention Aware VM Allocation Technique.

[Akoush et al. (2010)] discuss the parameters that can be used to predict migration times with focus on the Xen virtualization platform. Xen employes the pre-copy memory migration technique in which we iterate through multiple rounds of copying and any memory pages that get modified (dirtied) in the process are copied in the next iteration again until the dirty rate becomes small enough for the VM to be halted and restarted at the destination host. This design minimizes total migration time as well as downtime.

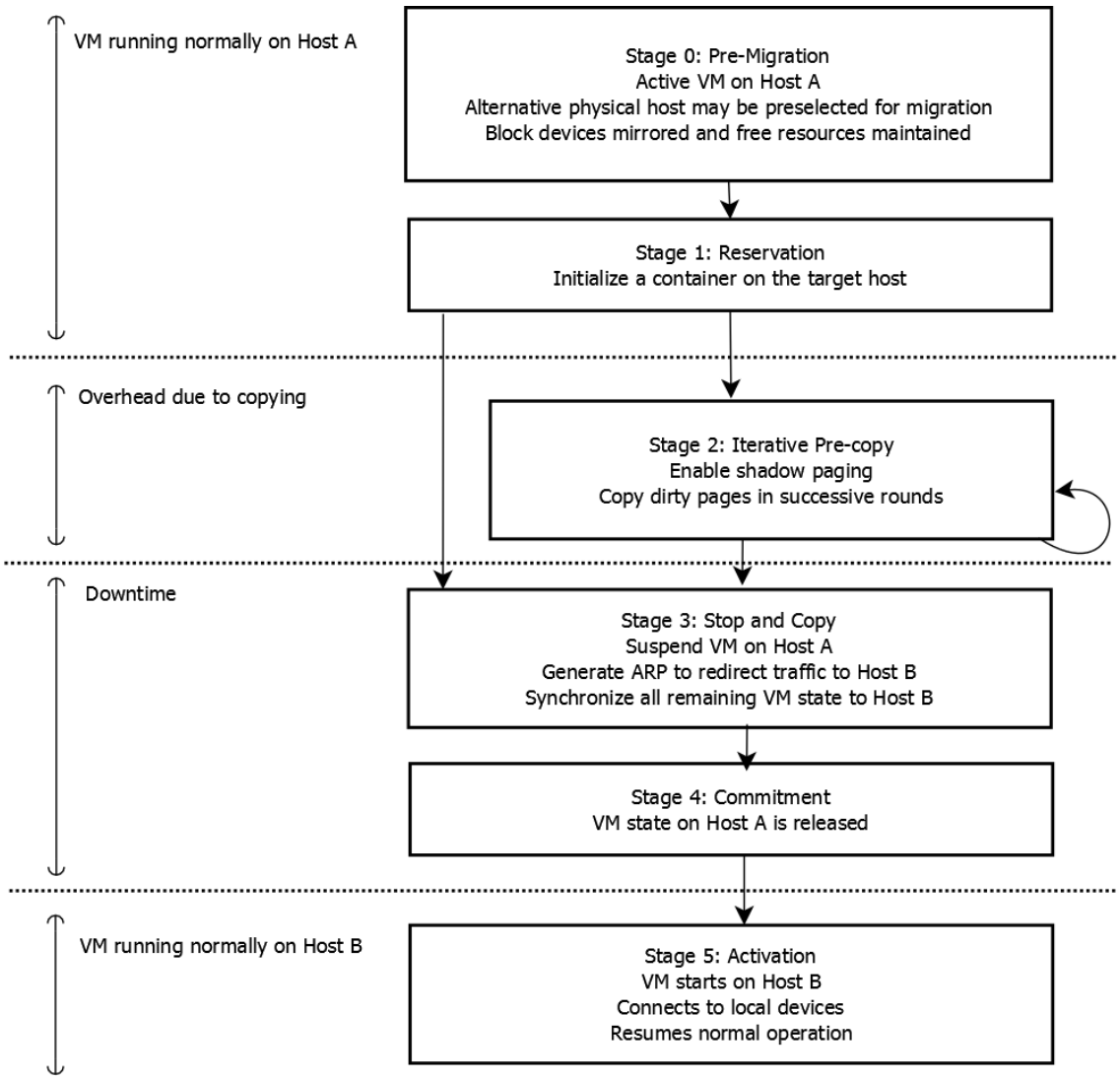


Figure 2.11: Live Migration Timeline [Buyya, Broberg (2011)].

Pre-copy migration involves 6 stages, [Clark et al. (2005)], as shown in the figure 2.11. The iterative pre-copy will continue indefinitely unless stopping conditions are present. Xen has three stopping conditions as follows,

1) Less than 50 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 RAM allocated to the VM has been copied to the destination host.

The first condition gives a smaller downtime as compared to the other two, which force the migration into the stop-and-copy phase even when there may be a large number of dirty pages left to be copied.

2.4.0.4.1 Migration link bandwidth influences migration more than any other parameter. Link capacity is inversely proportional to total migration time and total downtime. If the link speed is high, the transfer rate for the pages will be faster.

2.4.0.4.2 Page dirty rate is the rate at which the memory pages are modified. Higher the page dirty rate, more the number of pages that need to be transferred per iteration and hence, more the total migration time. It also increases the total downtime as the number of pages that need to be transferred during the stop-and-copy phase increase.

2.4.0.4.3 Pre-migration and Post-migration overheads refer to the operations that are not part of the actual transfer process and can be either static or dynamic overheads. Static overheads are operations related to initializing a container on destination, mirroring block devices, maintaining free resources, advertising moved IP address. These overheads are unavoidable. There are two proposed solutions for predicting migration as described by [Akoush et al. (2010)],

(a) The AVG Simulation Model assumes a constant or average page dirty rate, which is sufficient as an approximation only and fails if the page dirty rate is a function of time.

(b) The HIST Simulation Model is a specialization of the AVG Model. It works on the idea that the page dirty rate for deterministic processes will approximately be the same as the previous runs of the same workload running in a similar environment.

[Wood et al. (2007)] address the problem of manual hotspot detection and migration by proposing automated strategies for the same. The proposed *black-box* technique makes decisions by observing VMs from the outside and the *gray-box* strategy assumes access

to a small amount of OS-level statistics for better migration. The proposed model, Sandpiper 2.12, is a system for automated migration of virtual servers. It assumes that there is a cluster of heterogeneous servers whose hardware configuration is known to it. Each PM runs a virtual machine monitor (Xen Hypervisor) and one or more VMs. Each VM runs an application and is allocated a slice of the physical resources as follows,

- **CPU:** A weight is assigned to the VMs and the underlying Xen scheduler allocates CPU bandwidth proportionally to the allocated weight.
- **Network Interface:** Best effort FIFO scheduler is used.
- **Memory:** A certain amount of RAM is allocated to each VM.
- **Storage:** There is no need to move disk state during migration as all the storage is on a storage area network (SAN).

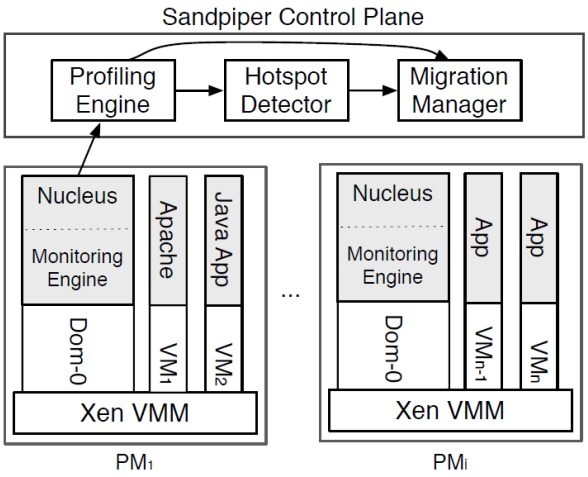


Figure 2.12: The sandpiper architecture [Wood et al. (2007)].

Sandpiper runs a nucleus on each server, which runs inside Dom-0 of the Xen hypervisor and gathers resource usage statistics using a monitoring service. For the gray-box approach, a daemon is implemented inside each VM to get OS-level statistics. The nuclei sends these statistics to the sandpiper control pane which consists of three components, a *profiling engine* which constructs resource usage profiles for each VM, a *hotspot detector* which monitors the profiles for hotspots and invokes the *migration managers* when the need arises. The migration manager determines what to migrate, where to and how much of a resource to allocate.

[Liu et al. (2011)] estimate the live migration cost in terms of performance and energy. The performance model tries to determine which VM should be migrated with the

minimum cost. It considers various migration performance factors - size of VM memory, memory dirtying rate, network transmission rate amd migration algorithm. This model states that a VM with small size of memory image an small memory dirtying by transmission ratio is the better candidate for migration, as it will generate lesser network traffic and will have shorter migration latency. A second scenario considers that the memory dirtying rate is larger than the memory transmission rate i.e., the workload consists of a set of hot pages that are updated extremely frequently, known as Writable Working Set (WWS) as described by [Clark et al. (2005)]. These pages are used to store stack and local variables of running processes. As these pages are dirtied the fastest, they are transferred during the stop-and-copy phase of pre-copy migration. For the energy model, the power drawn by the physical server consists of a static portion - stable power consumption even if the server is idle, and dynamic portion- consumption my resources working on behalf of VM. The power drawn by VM migration is determined by the data transmission rate, which in turn has effect on the migration latency. Therefore, the energy cost equals the energy drawn at the source and destination hosts.
The final cost comes down to,

$$C(VM_i) = aT_{down} + bT_{mig} + cV_{mig} + dE_{mig} \tag{2.4}$$

where a, b, c, d are the cost metrics, and $a + b + c + d = 1$.

T_{down} is the application downtime, T_{mig} is the total migration time, V_{mig} is the total network traffic during migration and E_{mig} is the energy consumption by the migration process.

Table 2.4: Comparison of various allocation techniques for the Xen hypervisor.

Parameters	**[Liu et al. (2011)]**	**[Yu et al. (2014)]**	**[Akoush et al. (2010)]**	**[Wood et al. (2007)]**
Objective	VM migration cost (performance and energy)	self-adaptive VM allocation	workload specific prediction of service interruption	automated VM migration
Run as/inside	inside host process	inside host manager	as host process	as a daemon
Component being monitored	memory access pattern of running workload	I/O degree of Dom-0	page dirty rate	Dom-0 resource usage statistics
Performance evaluation	reduced migration latency (72.9%), reduced downtime (93.5%)	reduction of average response time (37%), average execution time (20%)	migration time predicted with 90% accuracy	singe server hotspot resolution in 20s

Chapter 3
Present Work

3.1 Problem Formulation

Resources are allocated to the users transparently and on-demand, in the form of VMs which exist as independent entities on top of a host operating system. The virtual machines make it easy to allocate, deallocate, scale up and scale down the resources. Many of these operations require the virtual machine to be transferred (migrated) to a different host on the network. Our aim is to decrease the live migration time of these VMs running specifically on the Xen hypervisor to decrease the allocation delay.

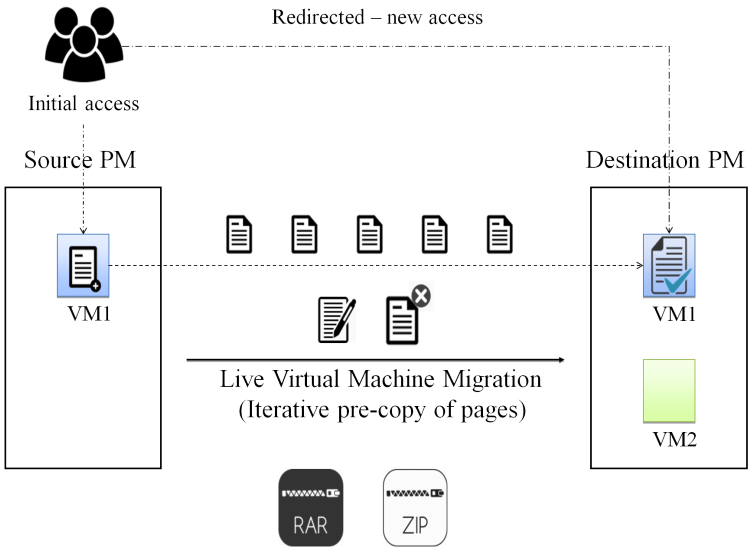


Figure 3.1: Live VM migration procedure with modifications.

Xen hypervisor is an open-source bare-metal VMM. Its current stable version is 4.4.2 and unstable release is 4.5. The code for migration of the VMs is contained in the xen/tools/libxc/xc_domain_save.c and xen/tools/libxc/xc_domain_restore.c. The current

technique of migration involves selecting and making a batch of 1024 pages of memory and transferring it to the destination host. The further iterations repeat this process by resending the pages dirtied in the earlier rounds. Xen supports *remus compression*, also known as checkpoint compression, which live migrates a copy of the running VM to a backup server which loads in case the VM running on our local system fails. This is done to restore the VM in case of any unknown error or crash.

Our problem can be defined as a problem of fast resource provisioning which involves live migration of virtual machines in the Xen hypervisor. Time involved in the live migration depends on the page dirty rate. More the number of pages dirtied, more number of iterations will be there, increasing the time for transfer of all the pages. Xen automatically stops the iterative pre-copy phase of migration and forces the VM to shut down and go into the post copy phase on meeting any one or more of the given conditions,

1) Less than 50 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 RAM allocated to the VM has been copied to the destination host.

Problem Definition: The problem is to reduce the amount of dirty pages or dirty bits of the VM memory during transfer and reduce the size of data being sent over the network to reduce migration latency in case of Xen hypervisor, shown in fig.3.2. This has been achieved by serializing the bits and compressing a batch of pages before transfer over the network. This work is an extension from [Imteyaz (2010)].

3.2 Objectives

1. To scale up our private cloud by using resources from the public cloud during peak workload hours.

2. To increase the performance of resource provisioning process by reducing the migration latency (decrease of total migration time and total downtime).

3. To reduce the cost incurred by the users for using resources of the private cloud, in comparison to the public cloud.

4. To make sure that service-level agreements for a user are not violated by the allocation of resources to another user.

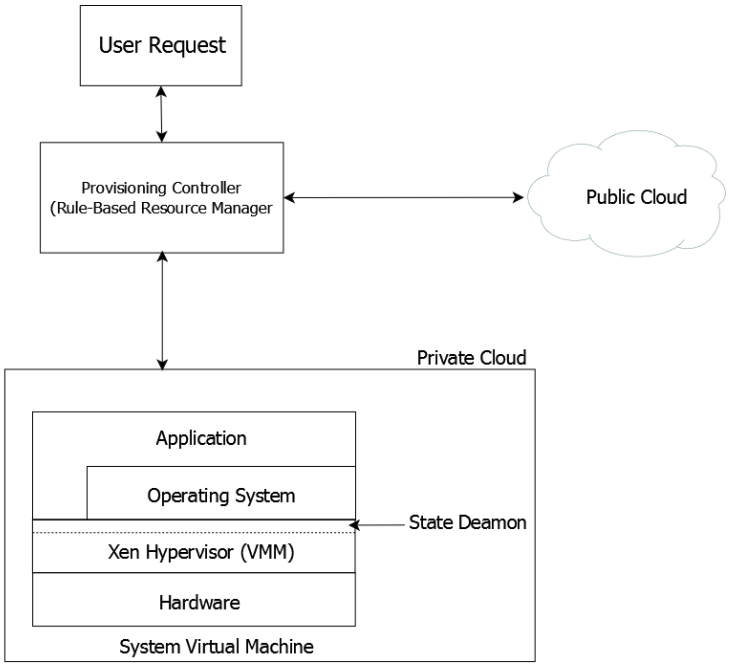


Figure 3.2: Proposed Solution.

3.3 Methodology

In order to perform live migration of virtual machines, the setup shown in fig. 3.3 was used. Our setup consisted of two physical machines with configuration described in table 3.1.

Both machines were connected through DLink DSL-2730U router and also through a 100 Mbps CAT.5 UTP cable to estalish a wired LAN connection for fast migration. Each machine acted as a Network File System (NFS) server for sharing the virtual machine saved image file. The creation, destruction, and migration of virtual machines was done via both, graphical interfaces and command-line interfaces.

3.3.1 Virtualization Tools

Virtualization and VM migration using Xen hypervisor was tested in the following tools,

1. **Console Tools**

   1.1 **libvirt**: A toolkit for management of VM guests, storage and virtual networks. It provides an API, a daemon and a shell (virsh).

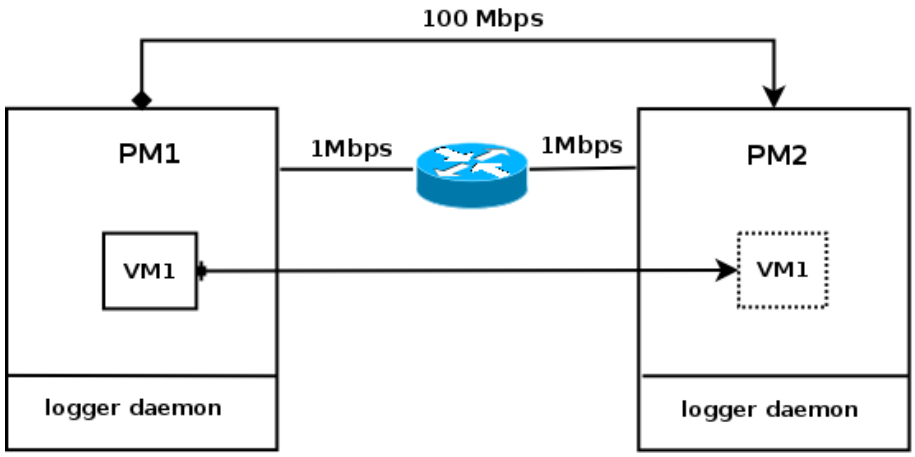


Figure 3.3: System setup.

2.2 **virt-install**: Supports both text-based and graphical installations.

2. **GUI Tools**

2.1 **virt-manager**: A graphical management tool which is most commonly used on workstations.

2.2 **virt-viewer**: An X viewer client for VM guests with TLS/SSL encryption.

3.3.2 Installing Xen

Installion of Xen can be done by executing the given commands in the sudo mode,

Listing 3.1: Installation of Xen

```
# yum -y install xen xen-hypervisor xen-runtime xen-libs
# systemctl enable xendomains.service
```

The status of a service can be checked by executing,

Listing 3.2: Check status of service (active or dead).

```
# systemctl status xendomains.service
```

Installing these packages creates new entries in the GRUB2 as showin in fig.3.5. We need to boot into the **'Fedora, with Xen hypervisor'** option to work with Xen.

Table 3.1: Configuration of physical machines

	PM1	**PM2**
Machine	Dell Inspiron N4010	Dell XPSl501X
Operating system	Fedora 21	Fedora 21
Linux kernel release	3.18.7-200.fc21.x86_64	3.17.4-301.fc21.x86_64
Processor	Intel Core i3	Intel Core i5
RAM	2GB	4GB
Xen hypervisor version	4.4.2	4.4.1

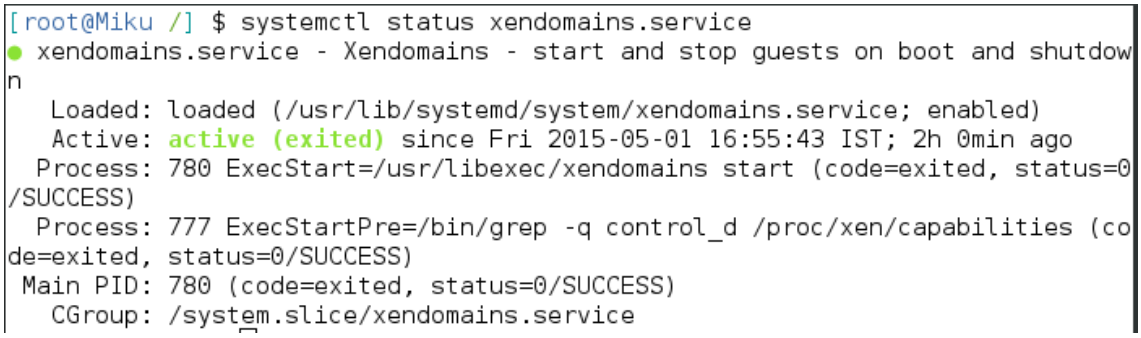


Figure 3.4: Active status of xendomains.service.

Reboot and verify that Xen is running:

Listing 3.3: Verify if Xen is running.

```
# xl info
```

The output shows the system architecture and installed hypervisor versions as shown in fig.3.6.

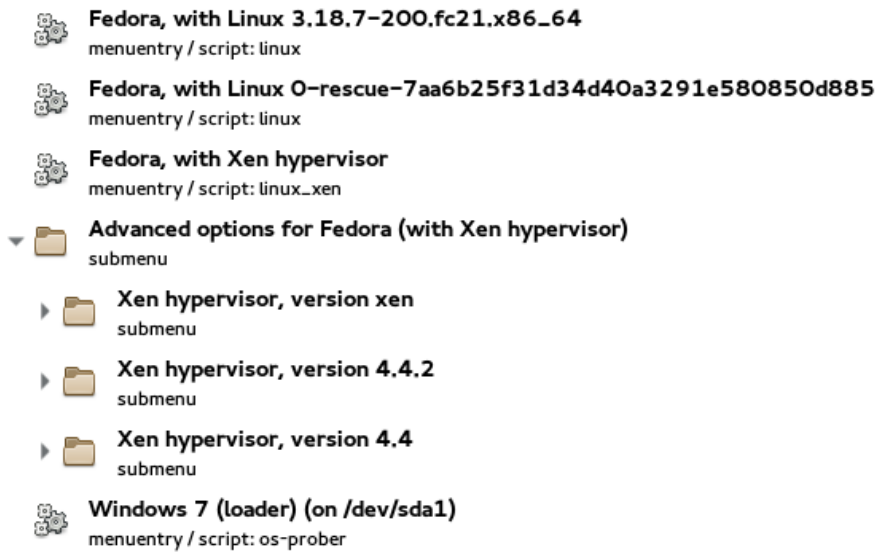


Figure 3.5: Entries of GRUB2.

```
[root@Miku /] $ xl info
host                   : Miku
release                : 3.18.7-200.fc21.x86_64
version                : #1 SMP Wed Feb 11 21:53:17 UTC 2015
machine                : x86_64
nr_cpus                : 4
max_cpu_id             : 3
nr_nodes               : 1
cores_per_socket       : 2
threads_per_core       : 2
cpu_mhz                : 2261
hw_caps                : bfebfbff:28100800:00000000:00003f00:0098e3bd:00000000:00000001:00000000
virt_caps              : hvm
total_memory           : 1908
free_memory            : 117
sharing_freed_memory   : 0
sharing_used_memory    : 0
outstanding_claims     : 0
free_cpus              : 0
xen_major              : 4
xen_minor              : 4
xen_extra              : .2
xen_version            : 4.4.2
xen_caps               : xen-3.0-x86_64 xen-3.0-x86_32p hvm-3.0-x86_32 hvm-3.0-x86_32p hvm-3.0-x86_64
xen_scheduler          : credit
xen_pagesize           : 4096
platform_params        : virt_start=0xffff800000000000
xen_changeset          :
xen_commandline        : placeholder
cc_compiler            : gcc (GCC) 4.9.2 20150212 (Red Hat 4.9.2-6)
cc_compile_by          : mockbuild
cc_compile_domain      : [unknown]
cc_compile_date        : Tue Mar 31 20:21:35 UTC 2015
xend config format     : 4
```

Figure 3.6: Output of *xl info*.

Install virt-manager to manage the VMs.

Listing 3.4: Install virt-manager.

```
# yum -y install virt-manager dejavu* xorg-x11-xauth
# yum -y instal libvirt-daemon-driver-network
  libvirt-daemon-driver-storage libvirt-daemon-xen
# systemctl enable libvirtd.service
# systemctl start libvirtd.service
```

This completes the Xen hypervisor installation. TheVMM should now be installed and can be accessed graphically as shown in fig.3.7.

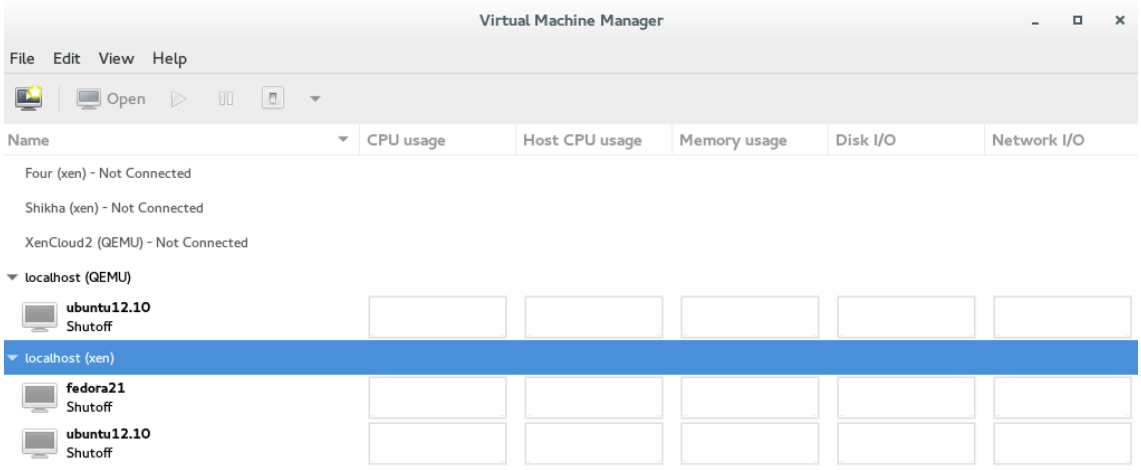


Figure 3.7: Virtual Machine Manager.

3.3.3 Configuring Dom-0 and Dom-Us

To install guest VMs in Xen, we first need to add a connection to the required hypervisor for the localhost - Xen, Quick EMUlator (QEMU)-Kernel Virtual Machine (KVM) and LinuX Containers (LXC) are the hypervisors which can be configured.

Equivalent command in virsh,

Listing 3.5: Connect to Xen.

```
# virsh connect xen:///
```

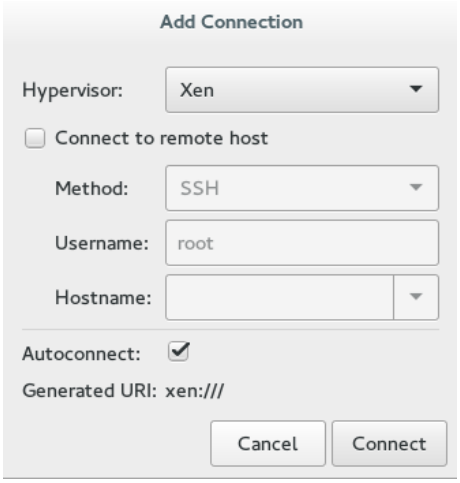
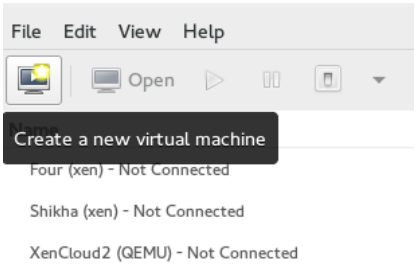
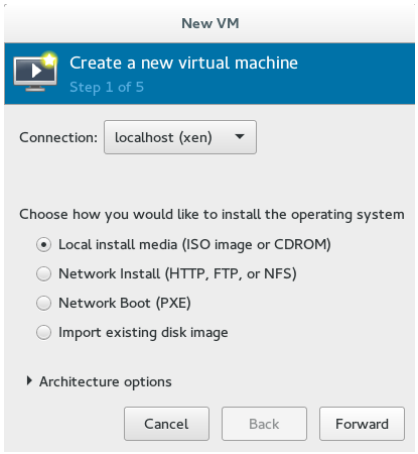


Figure 3.8: Connect to localhost Xen.

Now we can create a new VM as explained in figures (3.9a) - (3.12b),



(a) Click on create a new VM.



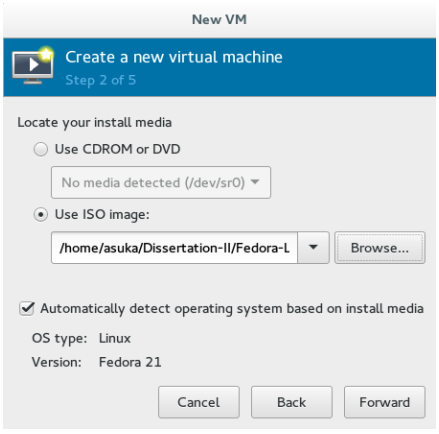
(b) Choose the hypervisor on which VM should be created.

Figure 3.9: VM Creation-I.

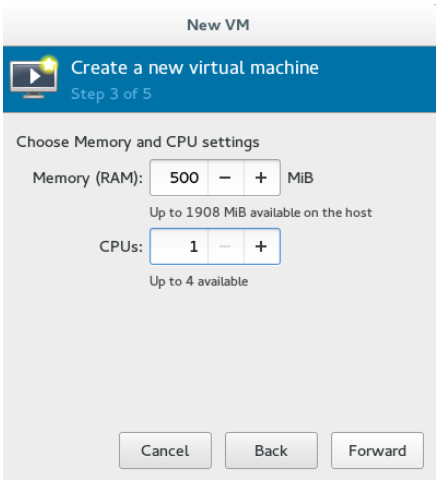
VMs can also be created via command line as follows,

Listing 3.6: Connect to Xen.

```
# virt-install --connect xen:// --virt-type xen
--name testvm --memory 1024 --disk-size=5
--cdrom /dev/cdrom --os-variant Ubuntu12.10
```

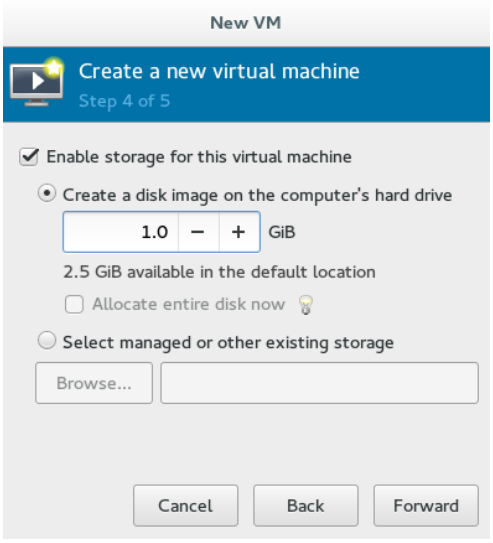


(a) Choose an ISO file of the OS to install on the Dom-U.

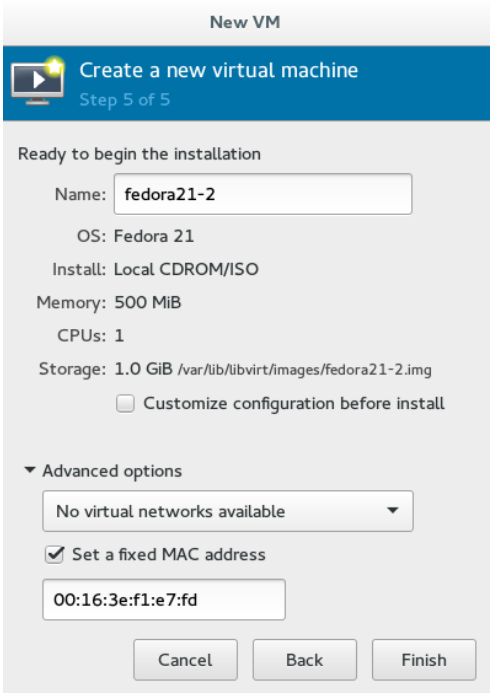


(b) Choose the size of RAM and number of vCPUs required.

Figure 3.10: VM Creation-II.

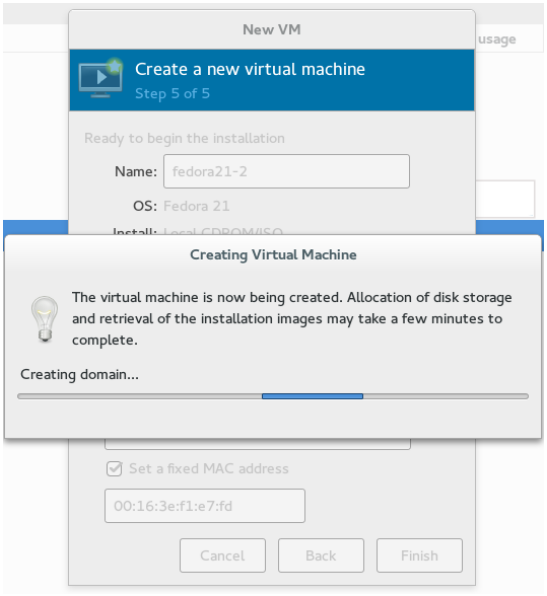


(a) Select space to allocate for VM.



(b) Verify all the settings.

Figure 3.11: VM Creation-III.



(a) VM in the process of creation.

(b) VM starts running after creation and shows CPU & memory usage in VMM.

Figure 3.12: VM Creation-IV.

The list of currently running VMs can be checked as,

Listing 3.7: Vm list in Xen.

```
# xl list
```

```
[root@Miku asuka] $ xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0  1194     4      r-----    6130.2
fedora21-2                                   1   500     1      -b----      31.0
```

Figure 3.13: List of running VMs.

3.3.4 Migrating VM Guests

In order to migrate VMs from one host to another, the following prerequistes should be fulfilled.

1. There should be an **Secure Shell (SSH)** connection between the two hosts, for remote login.

2. The two hosts need a **shared storage**: Network File Storage (NFS), Storage Area Network (SAN) or Internet Small Computer System Interface (iSCSI), for the sharing of the VM image file.

3.3.4.1 OpenSSH Configuration

Listing 3.8: OpenSSH Installation.

```
# yum -y install openssh-server
# /sbin/service sshd status
# /sbin/service sshd start
```

By default, SSH listens to incoming connections on port 22, unless another port is explicity specified. Port 22 is blocked by the firewall and needs to be opened or we need to accept connections on port numbers between 1025-65535.

Listing 3.9: Command to check status of OpenSSH.

```
# /sbin/service sshd status
```

Checking the status of SSH shows the port number it is listening on in fig.3.15.

Listing 3.10: Remote login command.

```
# ssh -p 22 169.254.11.115
```

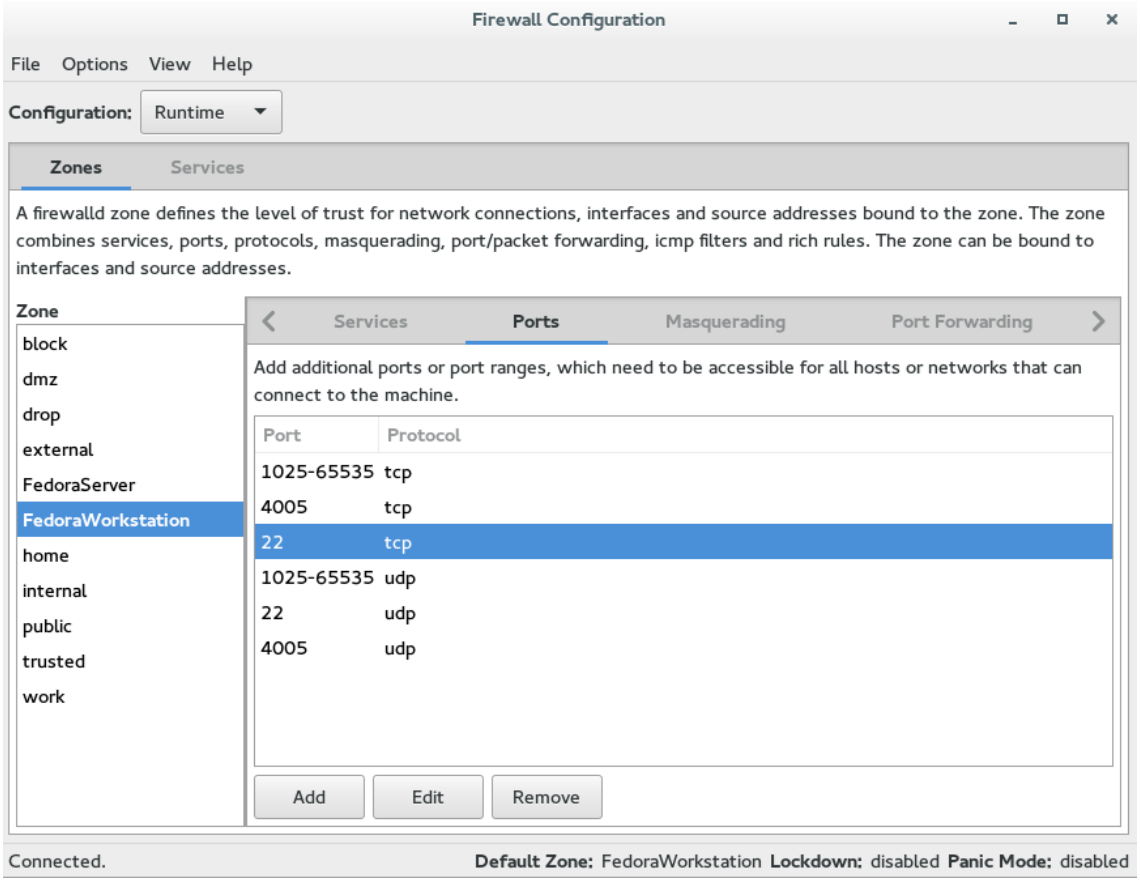


Figure 3.14: Opening port 22 for TCP & UDP connections.

```
[root@Miku asuka] $ /sbin/service sshd status
Redirecting to /bin/systemctl status  sshd.service
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled)
   Active: active (running) since Fri 2015-05-01 20:30:52 IST; 3h 28min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 1287 (sshd)
   CGroup: /system.slice/sshd.service
           └─1287 /usr/sbin/sshd -D

May 01 20:30:52 Miku sshd[1287]: Server listening on 0.0.0.0 port 22.
May 01 20:30:52 Miku sshd[1287]: Server listening on :: port 22.
```

Figure 3.15: Status of SSH service.

3.3.4.2 NFS Configuration

One of the two hosts needs to act as an NFS server, and the other one as an NFS client. For VM migration to succeed, the directory used for storing disk images has to be mounted from the shared storage from both hosts .

Listing 3.11: NFS configuration.

```
# yum -y install sshfs
# # rpcinfo -p | grep nfs
100003    3   tcp   2049  nfs
100003    4   tcp   2049  nfs
100227    3   tcp   2049  nfs_acl
100003    3   udp   2049  nfs
100003    4   udp   2049  nfs
100227    3   udp   2049  nfs_acl
# cat /proc/filesystems | grep nfs
nodev   xenfs
nodev   nfsd
nodev   nfs
nodev   nfs4
# rpcinfo -p | grep portmapper
100000    4   tcp    111  portmapper
100000    3   tcp    111  portmapper
100000    2   tcp    111  portmapper
100000    4   udp    111  portmapper
100000    3   udp    111  portmapper
100000    2   udp    111  portmapper
```

Listing 3.12: Edit exports file.

```
# vim /etc/exports
# add line /var/lib/libvirt/images *(rw,sync,no_root_squash)
# systemctl restart nfs
# reboot system
# mount -v 169.254.11.115:/var/lib/libvirt/images
 /var/lib/libvirt/images
```

NFS service needs to be started on every system boot as it it inactive by default.

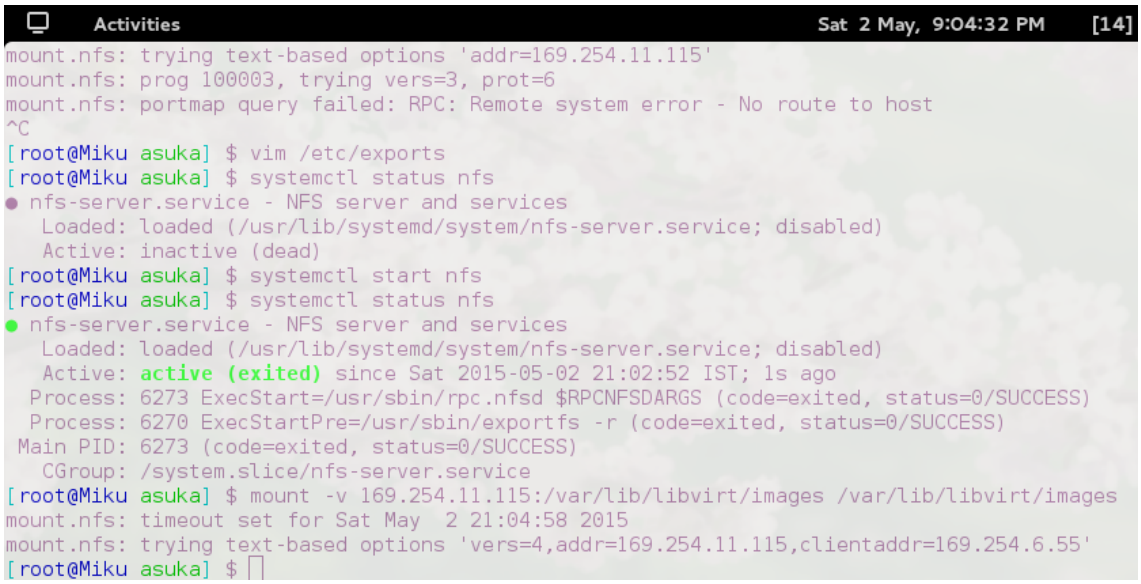


Figure 3.16: Status of NFS service.

3.3.4.3 Start Migration

The guest VM or Dom-U needs to be running before migration can be initiated.

Listing 3.13: Initiate migration.

```
# virsh migrate --live centos6.5
        xen+ssh://192.168.43.55/system
```

The system prompts for a password and migrates the VM.

```
[root@Four four4]# virsh migrate --live 2 xen+ssh://192.168.43.55/system
root@192.168.43.55's password:
```

Figure 3.17: Migration command.

Our technique involved serializing and compressing (lossless) the memory pages of the VM, which involved being able to peek at the contents of the primary memory. This can be done in a number of ways. Linux has support for hexdump commands which print the binary, hexadcimal , canonical forms of data of a file with the ASCII representation. We can dump the contents of the memory using the given commands,

Listing 3.14: hexdump command(i)

```
# dd if = /dev/mem | hexdump -d
```

Input offset is in hexadecimal; 16 space-seperated two-column hex bytes; same 16 bytes in %_p format enclosed in ‘ | ’.

Listing 3.15: hexdump command(ii)

```
# xxd -b \dev\mem
```

In virsh we can use the dump command to achieve the same effect for Dom-Us. The output appears as in fig.3.18.

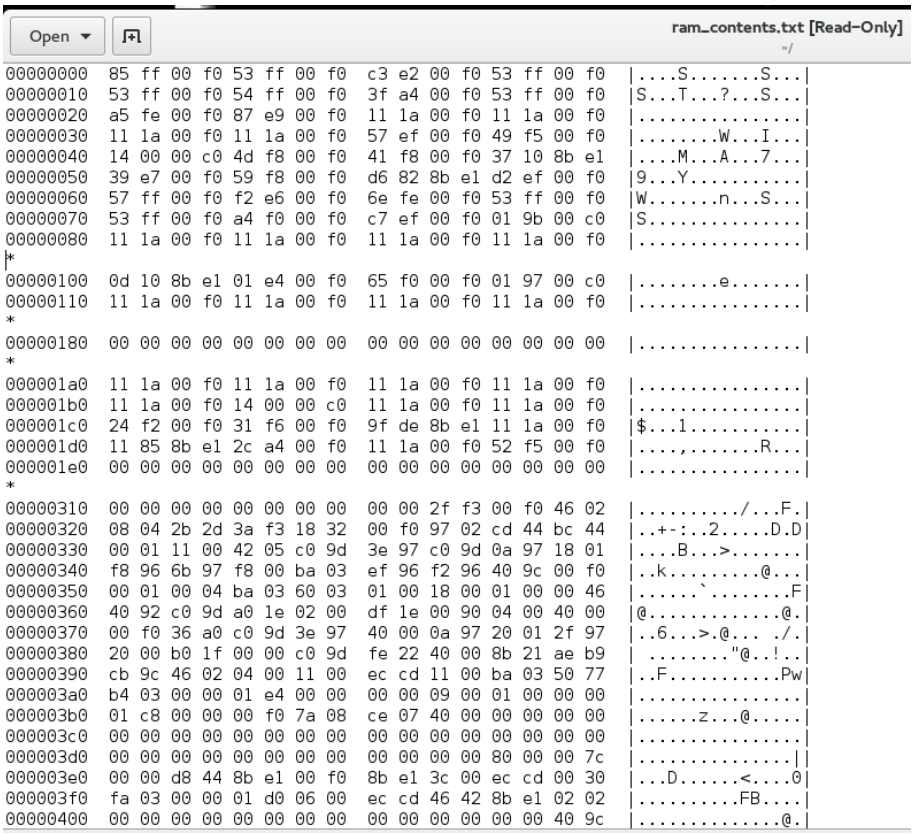


Figure 3.18: Output of hexdump command, showing contents of primary memory.

The library used for serialization is an open-source library named tpl by [Troy D. Hanson] and the lossless compression algorithm is also open-source by [Rich Geldreich].

In oreder to automate the migration process, we can write a BASH script to create and migrate VMs every minute or so.

Listing 3.16: Automating the migration process

```
# !/bin/sh

while true
do
        virsh migrate --live <domain-id>
                xen+ssh://192.168.43.55/system
        sh MyScript.sh
        sleep 10
done
```

Chapter 4
Results and Discussions

4.1 Results

Table 4.1: Comparison of our proposed approach with the original downtime.

	0.5GB VM	1GB VM	1.5GB VM	2GB VM
Downtime (ms) proposed approach	20	25	29	33
Downtime (ms) non-adaptive approach	23	28	28	33

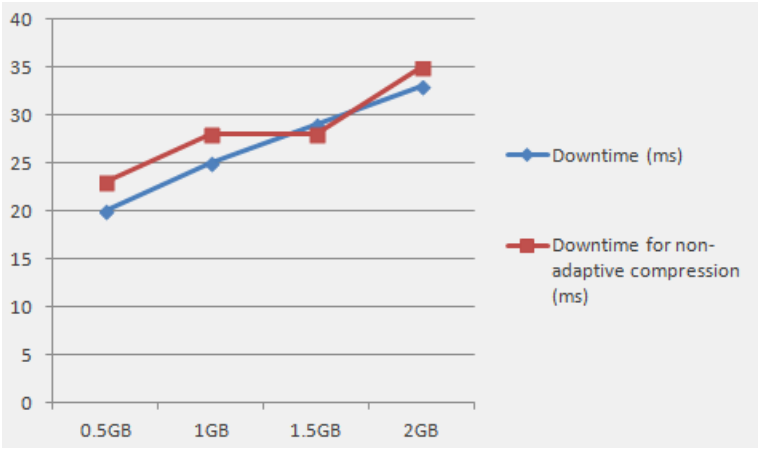


Figure 4.1: Comparision of downtime of adative memory compression with serializing & compressing approach.

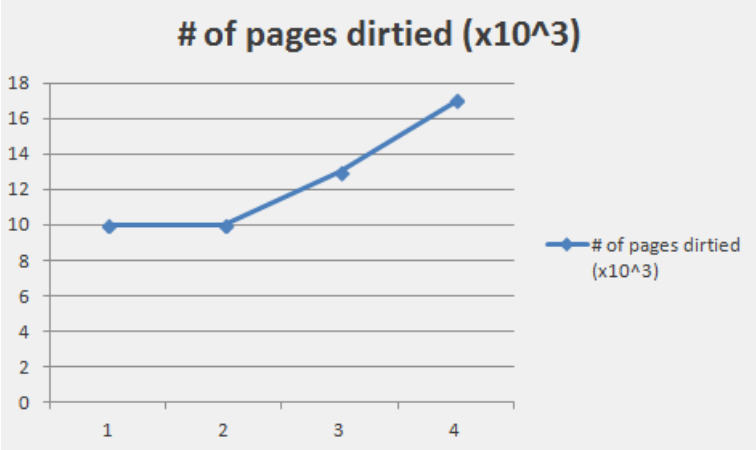


Figure 4.2: Page dirty rate of serializing & compression approach.

The CPU utilization of the Dom-0 can be seen using the *xentop* command in Linux.

Listing 4.1: xentop command

```
# xentop
```

```
xentop - 07:12:22   Xen 4.4.1
3 domains: 1 running, 2 blocked, 0 paused, 0 crashed, 0 dying, 0 shutdown
Mem: 3920180k total, 3735844k used, 184336k free    CPUs: 4 @ 2527MHz
      NAME  STATE   CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS NETS NETTX(k) NETRX(k) VBDS   VBD_OO   VBD_RD   VBD_WR  VBD_RSECT  VB
D_WSECT SSID
 centos6.5 --b---         27    0.0    1048584   26.7    1049600      26.8     1    0        0        0    0        0        0        0          0
      0    0
VCPUs(sec):   0:        27s
      NAME  STATE   CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS NETS NETTX(k) NETRX(k) VBDS   VBD_OO   VBD_RD   VBD_WR  VBD_RSECT  VB
D_WSECT SSID
  Domain-0 -----r       4352   10.4    1566784   40.0   no limit       n/a     4    0        0        0    0        0        0        0          0
      0    2
VCPUs(sec):   0:       748s  1:      1403s  2:      1153s  3:      1046s
      NAME  STATE   CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS NETS NETTX(k) NETRX(k) VBDS   VBD_OO   VBD_RD   VBD_WR  VBD_RSECT  VB
D_WSECT SSID
  fedora21 --b---         105    0.1    1048584   26.7    1049600      26.8     2    0        0        0    0        0        0        0          0
      0    0
VCPUs(sec):   0:        60s  1:        45s
```

Figure 4.3: Output of xentop command.

The network utilization - number of packets sent and received, is captured in the /proc/net/dev file.

Listing 4.2: Command to display network utilization

```
# cat /proc/net/dev
```

```
[root@Four four4]# cat /proc/net/dev
Inter-|   Receive                                                |  Transmit
 face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets errs drop fifo colls carrier compressed
wlp4s0: 96714374   91900    0    0    0     0          0         0  9485175   91236    0    0    0     0       0          0
enp9s0:       0       0    0    0    0     0          0         0        0       0    0    0    0     0       0          0
    lo:   20807     160    0    0    0     0          0         0    20807     160    0    0    0     0       0          0
[root@Four four4]#
```

Figure 4.4: Screenshot showing network utilizaton.

4.2 Discussions

Xen uses the pseudo-physical memory model in which applications on an OS with protected memory have their own address space and have access to the entire memory space from their perspective. For x86 platforms, their exists a concept of segmented virtual address space. In this, addresses are resolved to a segment and offset which are further mapped to a linear address i.e., virtual address. A page is typically of 4KB in size, although superpages can be as big as 2MB. Page table size varies with the machine. For our Dell Inspiron system with 2GB primary memory, page table size was calculated as follows,

page size $= 4KB$
address apce $= 48 - bit$
physical memory $= 2GB = 2,097,152KB$
number of pages $= \frac{2048M}{4K} = 512K$ pages

19 bits to represent each page + 4 reserved bits (valid bits, protection bits, dirty bits, use bits) = 23 bits

number of page table entries $= \frac{virtual\_address\_space}{page\_size} = \frac{2^{48}}{2^{40}} = 2^8$ entries $= 256$ entries

number of frames $= \frac{512*1024*1024}{4*1024} = 2^{17}$

17 bits to represent each page + 4 reserved bits (valid bits, protection bits, dirty bits, use bits) = 21 bits = 2.625 bytes

size of page table $= 2^8 * 2.675 = 672$ bytes $= 0.65KB$

Table 4.1 shows the downtime of VMs of various sizes and compares it with the results of a non-adaptive compression technique. The results show that as the size of VM increases, the downtime also increases. This happens because the size of primary memory for each VM is different and it has more number of pages to transfer. Compared to the

non-adaptive compression approach, the proposed approach performs better. However, the dip in the curve at the 1.5GB VM has occured because of uneven network bandwidth, during which the number of pages being migrated were very less, increasing the downtime.

Fig.4.2 shows the page dirty rate of the proposed approach. The total number of pages are $512x10^3$. The amount of pages dirtied are: 10K for 0.5GB, 10K for 1GB, 13K for 1.5GB, 17K for 2GB VM. The page dirty rate is directly proportional to the network bandwidth. If the link speed of the network is low or unsteady, the page dirty rate increases.

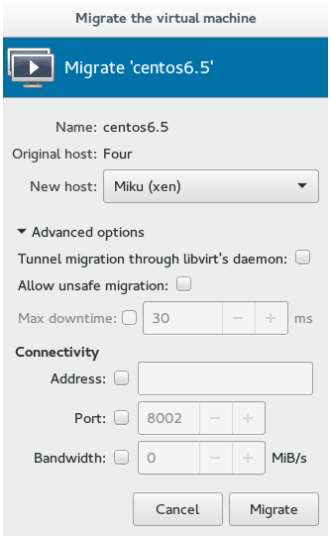
4.3 Limitations

The limitations arise while trying to migrate VM using GUI tools such as Virtual Machine Monitor of libvirt. The tool has bugs and even though it shows a *migrate* option on right-click, it fails to migrate any and all Dom-Us with the following error:

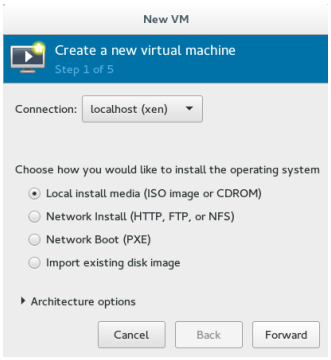
Listing 4.3: Error on migration.

```
Unable to migrate guest: this function is not supported by
the connection driver: virDomainMigrate
```

This error arises because of a bug in the coding of the tool, in the function named *virDomainMigrate* and is currently under consideration for a patch. We have to manually migrate VMs using command-line tools only. Also, migration cannot be invoked without a shared storage, else it results in an error.



(a) Starting migration to a remote host.



(b) Error pops up during migration.

Figure 4.5: VM Creation-I.

Chapter 5
Conclusion and Future Scope

The critical tasks in virtualized systems today are the virtual machine provisioning and migration services. They help us service the requests of end users with ease, help in maintenance of servers by migrating virtual machines to a completely different server in a short amount of time, help in workload balancing and other needs.

We have proposed an algorithmic approach to provisioning virtual machines on a private cloud and efficiently migrating them to a public server for scaling purpose by reducing the time it takes for the virtual machines to migrate on the Xen hypervisor. The resulting migration times is reduced as comapred to the original scheme of simply clustering and sending the pages to the destination.

The work can further be expanded on other hypervisors like QEMU, KVM to improve their migration times. A new resource allocation scheme named *Dockers* or Linux Containers has recently been proposed in October 2014 and can prove to be a fruitful alternative for virtualization as the size of allocation in dockers is very less as compared to VMs whose sizes are in GBs.

Chapter 6
References

[Buyya, Broberg (2011)] Buyya, R., Broberg, J., Goscinski, A.,(2011) *Cloud Computing: Principles and Paradigms*, Wiley Series on Parallel and Distributed Computing.

[Chaganti (2007)] Chaganti, P.,(2007) *Xen Virtualization: A Practical handbook*, Packt Publishing.

[Chisnall (2008)] Chisnall, D.,(2008) *The Definitive Guide to the Xen Hypervisor*, Prentice Hall.

[Hagen (2008)] Hagen, W.,(2008) *Professional Xen Virtualization*, Wiley Publishing, Inc.

[Tanenbaum, Steen (2007)] Tanenbaum, A.S., Steen, M.V.,(2007) *Distributed Systems: Principles and Paradigms*, Pearson Prentice Hall.

[Akoush et al. (2010)] Akoush, S., Sohan, R., Rice, A., Moore, A.W., Hopper, A., "Predicting the Performance of Virtual Machine Migration," *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pp. 37 - 46, August 2010.

[Barham et al. (2003)] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A., "Xen and the Art of Virtualization," *SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 164 - 177, October 2003.

[Calcavecchia et al. (2012)] Calcavecchia, N.M., Caprarescu, B.A., Nitto, E.D., Dubois, D.J., Petcu, D., "DEPAS: A Decentralized Probabilistic Algorithm for Auto-Scaling," *Journal of Computing, Springer Vienna*,Volume 94, Issue 8 - 10, pp. 701 - 730, September 2012.

[Clark et al. (2005)] Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A., "Live Migration of Virtual Machines," *Proceeding NSDI'05 Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation - Volume 2*, pp. 273-286, 2005.

[Cuomo et al. (2012)] Cuomo, A., Modica, G.D., Distefano, S., Puliafito, A., Rak, M., Tomarchio, O., Venticinque, S., Villano, U., "An SLA-based Broker for Cloud Infrastructure," *Journal of Grid Computing*, Volume: 11, Issue: 1, pp. 1 - 25, October 2012.

[Hu et al. (2013)] Hu, L., Zhao, J., Xu, G., Ding, Y., Chu, J., "HMDC: Live Virtual Machine Migration Based on Hybrid Memory Copy and Delta Compression," *Applied Mathematics & Information Sciences*, Volume: 7, Issue: 1, pp. 639 - 646, January 2013.

[Imteyaz (2010)] Imteyaz, Y., "On-demand Resource Provisioning in Hybrid Cloud Environment," *International Conference on Advances in Communication, Network, and Computing – CNC*, 2010.

[Jiang et al. (2013)] Jiang, Y., Perng, C.S., Li, T., Chang, R.N., "Cloud Analytics for Capacity Planning and Instant VM Provisioning," *Network and Service Management, IEEE Transactions on*, Volume: 10, Issue: 3, pp. 312 - 325, September 2013.

[Jin et al. (2009)] Jin, H., Deng, L., Wu, S., Shi, X., Chen, H., Pan, X., "Live Migration of Virtual Machines with Adaptive Memory Compression," *IEEE Int. Conf. Cluster Computing and Workshops*, 2009, pp. 1 - 10.

[Liu et al. (2011)] Liu, H., Jin,H., Xu, C.X., Liao, X., "Performance and Energy Modeling for Live Migration of Virtual Machines," *HPDC'11 Proceedings of the 20th International Symposium on High Performance Distributed Computing*, December 2011.

[Mishra et al. (2012)] Mishra, M., Das, A., Kulkarni, P., Sahoo, A., "Dynamic Resource Management Using Virtual Machine Migration," *Communications Magazine, IEEE*, Volume:50, Issue: 9, pp. 34 - 40, September 2012.

[Ramachandran et al. (2012)] Ramachandran, L., Narendra, N.C., Ponnalagu, K., "Dynamic Provisioning in Multi-tenant Service Clouds," *Service Oriented Computing and Applications*, Volume: 6, Issue: 4, pp. 283 - 302, July 2012.

[Rodriguez, Buyya (2014)] Rodriguez, M.A., Buyya, R., "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflow on Clouds," *Cloud Computing, IEEE Transactions on*, Volume: 2, Issue: 2, pp. 222 - 235, April 2014.

[Son, Jun (2013)] Son, S., Jun, S.C., "An SLA-based Cloud Computing That Facilitates Resource Allocation in the Distributed Data Centers of a Cloud Provider," *The Journal of Supercomputing*, Volume: 64, Issue: 2, pp. 606 - 637, May 2013.

[Wood et al. (2007)] Wood, T., Shenoy, P., Venkataramani, A., Yousif, M., "Black-box and Gray-box Strategies for Virtual Machine Migration," *NSDI'07 Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, April 2007.

[Wu at al. (2013)] Wu, H., Zhang, W., Zhang, J., Wei, J., Huang T., "A Benefit-aware On-demand Provisioning Approach for Multi-tier Applications in Cloud Computing," *Frontiers of Computer Science*, Volume: 7, Issue: 4, pp. 459 - 474, July 2013.

[Wuhib et al. (2013)] Wuhib, F., Stadler, R., Spreitzer, M., "A Gossip Protocol for Dynamic Resource Management in Large Cloud Environments," *Network and Service Management, IEEE Transactions on*, Volume: 9, Issue: 2, pp. 213 - 225, June 2012.

[Yu et al. (2014)] Yu, J.L., Choi, C.H., Jin, D.S., Lee, J.R., Byun, H.J., "A Dynamic Virtual Machine Allocation Technique Using Network Resource Contention for a High-performance Virtualized Computing Cloud," *International Journal of Software Engineering and Its Applications*, Volume: 8, Issue: 9, pp. 17 - 28, 2014.

[Zaman, Grosu (2011)] Zaman, S., Grosu, D., "Combinatorial Auction-Based Mechanisms for VM Provisioning and Allocation in Clouds," *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on* , pp. 107 - 114, 2011.

[Rich Geldreich] Rich Geldreich, *miniz: Single C source file Deflate/Inflate compression library with zlib-compatible API, ZIP archive reading/writing, PNG writing* , `http://code.google.com/p/miniz/`, Last accessed on 04/05/15.

[Troy D. Hanson] Troy D. Hanson, *Easily store and retrieve binary data in C*, `http://tpl.sourceforge.net/`, Last accessed on 04/05/15.

[Xen] *Xen 4.5.0*, `http://fossies.org/dox/xen-4.5.0/index.html`, Last accessed on 04/05/15.

Chapter 7
Publications

Paper communicated in Thomson Reuters indexed journal:
Abbi, D., Pateriya, P.K., "Minimizing Migration Time of Virtual Machines by Serializing & Compressing Memory Pages in Xen Hypervisor," *Sustainable Computing: Informatics and Systems*, Volume: 5, Issue: 2, 2015 (communicated).

Chapter 8

Appendix

Algorithm 1 Modified Rule-Based Resource Manager

1: Rule_Based_RM($New\_VM\_Request_{c+s}$)
2: this
3: **loop**
4: **if** ($New\_VM\_Req\_(c+s) \leq Available_{(c+s)}$) **then**
5: Allocate on Private Cloud
6: **end if**
7: **if** ($New\_VM\_Req_{(c+s)}) > Available_{(c+s)} AND High\_Priority\_Request$ **then**
8: **for** δ_T **do**
9: Check Low_Priority_Request_Count on Private Cloud
10: **end for**
11: **if** (Low_Priority_Request_Count = 0) **then**
12: Choose y such that ($Priority_y < Priority\_New\_VM\_Request_{(c+s)}$)
13: AND ($y_{(c+s)} \geq New\_VM\_Request_{(c+s)}$)
14: AND ($Deadline_y \geq 2Deadline_{New\_VM\_Request}$)
15: Put $y_{(c+s)}$ at the front of queue
16: Allocate $New\_VM\_Request\_(c+s)$ on Private Cloud
17: **end if**
18: **if** $IsComplete(New\_VM\_Request_{(c+s)} = true)$ **then**
19: Re-allocate $y_{(c+s)}$ on Private Cloud
20: **end if**
21: Choose r from waiting queue such that ▷ To prevent starvation of low priority requests
22: $Priority_r = 4$ AND $Threshold < Waiting\_time\_in\_queue < Deadline_r$
23: **while** $Priority \geq 1$ **do**
24: $Priority_r = Priority - 1$ ▷ Increase priority
25: **end while**

```
26:     else if Low_Priority_Request_Count > 0 then
27:         Choose x such that x_(c+s) ≥ New_VM_request_(c+s)
28:         To Minimum (σ_x Relocation_Cost_on_Public_Cloud)
29:         if (x = φ) then
30:             count = 0
31:             Choose x such that (x_(c+s) ≥ 1/2 New_VM_Request_(c+s))
32:             count = count + 1
33:             if Elapsed Time of x_(c+s) < 50 then
34:                 Migrate x to Public Cloud
35:                 if (Available_(c+s) ≥ Need_(c+s) by New_VM_Request) then
36:                     Allocate New_VM_Request_(c+s) on Private Cloud
37:                 else
38:                     Repeat
39:                 end if
40:             end if
41:         end if
42:     else if Elapsed Time of x_(c+s) > 50 then      ▷ Let it complete on private cloud
43:         Additional_waiting_time_x_(c+s) = (Estimated_completion_time −
    Elapsed_time )
44:         if Additional_waiting_time ≥ ΔT then
45:             Migrate New_VM_Request_(c+s) to Public Cloud
46:         end if
47:     end if
48: end loop
```

Glossary

Cloud computing

a parallel and distributed computing system that consists of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified resources based on service level agreements. 1

Hypervisor

a piece of computer software-hardware or firmware that creates and runs virtual machines. 7

Live Migration

the process of transferring memory pages before the virtual machine which involves very less downtime. 7

SLA

a legal agreement an enterprise enters into with the infrastructure service providers to guarantee a minimum quality of service. 14

Virtualization

abstraction of the four computing resources: computation (CPU)- storage (HDD)- memory (RAM) and network or I/O. 6

Xen

a bare-metal hypervisor which runs directly on hardware to control the resources and manage guest operating system. 17

Acronyms

CRM

Customer-Relationship-Management. 3

Dom-0

Domain-0. 18, 22, 23, 41

Dom-U

Domain-U. 18, 37, 39, 44

EC2

Elastic Compute Cloud. 3

GAE

Google App Engine. 3

IaaS

Infrastructure-as-a-Service. 3

iSCSI

Internet Small Computer System Interface. 35

KVM

Kernel Virtual Machine. 31, 46

LXC

LinuX Containers. 31

NFS

Network File Storage. 35, 36, 37

OS

Operating System. 18, 42

PaaS

Platform-as-a-Service. 3

PM

Physical Machine. 7, 14, 21

QEMU

Quick EMUlator. 31, 46

QoS

Quality of Service. 2

SaaS

Software-as-a-Service. 3

SAN

Storage Area Network. 35

SLA

Service Level Agreements. 6, 14

SOA

Service Oriented Architecture. 1

SSH

Secure Shell. 35

VM

Virtual Machine. 5, 6, 7, 8, 11, 13, 14, 17, 18, 19, 21, 22, 23, 25, 26, 27, 28, 29, 31, 32, 34, 35, 36, 37, 38, 39, 43, 44, 46

VMM

Virtual Machine Monitor. 5, 7, 25, 29

WWS

Writable Working Set. 22