Performance Study of Live Virtual Machine Migration using KVM Hypervisor

Deepti Gupta

Computer Science Department,
Guru Gobind Singh Indraprastha University,
Sector 16C, Dwarka, New Delhi, Delhi 19078,India
deeptigupta.1709@gmail.com

Shipra Jain

Computer Science Department,
Guru Gobind Singh Indraprastha University,
Sector 16C, Dwarka, New Delhi, Delhi 19078,India
shiprastays@gmail.com

Sakshi Goel

Computer Science Department,
Guru Gobind Singh Indraprastha University,
Sector 16C, Dwarka, New Delhi, Delhi 19078,India
sakshigoel.2812@gmail.com

Abstract

In a cloud computing environment, migrating operating system instances across distinct physical hosts is a useful tool for administrators of data centers and clusters: It allows a clean separation between hardware and software, and facilitates fault management, load balancing, and low-level system maintenance. Live migration allows moving a continuously running VM from one physical host to another. It provides special benefit for data centers in a variety of scenarios including load balancing, maintenance and power management. Live migration is of two types: (1) Pre-copy live migration and (2) Post-Copy live migration each having its own advantages and disadvantages. In this report performance evaluation of post-copy live migration is presented along with proper mathematical modeling. This model can be used to estimate the Downtime, Resume Time and Total Migration Time of a VM being transferred using post-copy live migration

Keywords: Cloud; Virtual Machine; Hypervisor;KVM

1. Cloud

Cloud as we all know is a large connection of a lot of high powered server, all connected together. These servers are owned by someone and are remotely accessible. Cloud computing provides a “computing-as a-service” model in which compute resources are made available as a utility service — an illusion of availability of as much resources (e.g., CPU, memory, and I/O) as demanded by the user. Moreover, users of cloud services pay only for the amount of resources (a “pay-as-use” model) used by them. This model is quite different from earlier infrastructure models, where enterprises would invest huge amounts of money in building their own computing infrastructure. Typically, traditional data centers are provisioned to meet the peak demand, which results in wastage of resources during non-peak periods. To alleviate the above problem, modern-day data centers are shifting to the cloud. Important characteristics of cloud-based data centers: a) Make resource available on demand: The cloud model enables the users to have a computing environment without investing a huge amount of money to build a computing infrastructure. b) Flexible resource provisioning: Dynamically scale or shrink the provisioned resources as per the dynamic requirements. c) Enable ‘pay-as-use’ model

2. Virtual Machines

What exactly is a Virtual Machine? Suppose user wants a machine. So cloud provider can make a VM on one of the servers and give its remote access to the user. One might think why not give the server directly to user. The reason for this is that the requirement of user is not such that it uses a complete server. His/her requirement is less than that. Cloud provider can make many VMs on one server and can handle many customers together. Why pay for VM? User can directly buy a machine for himself, then, what is the need for cloud and VM? a) Users don’t have to pay for space to store the server. b) Users don’t have to pay for electricity and cooling. c) Users don’t have to pay for person to manage these servers. d) Users don’t have to worry about server technology getting obsolete. All this is managed by the cloud provider. Users just have to pay for VM and that too only for that much that is used by him, thus called ‘pay-as-use’.

3. Hypervisor

Most of the enterprises, today, have applications that have resource centric usage like I/O specific or CPU specific. Above this, the application workload also exhibits dynamic behavior based on time of the day or geographic locality. In such situations, to improve resource utilization, many enterprises are shifting towards cloud computing solutions where elastic resources can be availed on a pay-by-use mode. Virtualization [Anand et al., (2012)] is a key enabler in these cloud based solutions where multiple applications are co-hosted on a single machine. On such virtualized servers, physical hardware is shared among multiple virtual machines by a layer called hypervisor or virtual machine monitor (VMM) [Anand et al., (2012)]. Applications in turn, are hosted in the virtual machines and access the underlying shared hardware through hypervisor. Virtualization helps provide software isolation to applications in such shared environments as well as ensures better utilization of server resources. On cloud systems, I/O intensive applications are good candidates for virtualization because they have sufficient spare CPU cycles which can potentially be used by some other co-hosted application. A VM monitor (VMM) or hypervisor manages and multiplexes access to the physical resources, maintaining isolation between VMs at all times. As the physical resources are virtualized, several VMs, each of which is self-contained with its own operating system, can execute on a physical machine (PM). The hypervisor, which arbitrates access to physical resources, can manipulate the extent of access to a resource (memory allocated or CPU allocated to a VM, etc.). The decoupling between physical and virtual resources provided by the hypervisor enables flexibility of resource provisioning for VMs. Many virtual machine monitors have emerged in such a scenario, varying from VMware ESX to Xen paravirtualized hypervisor.

4. KVM

The Kernel based Virtual Machine (KVM) is a relatively new VMM which utilizes the hardware extensions and has found its way in Linux kernel. It is a full virtualization solution, which requires no changes in guest operating system [Anand et al., (2012)]. KVM is a virtualisation infrastructure for Linux kernel. It plugs itself into the existing OS i.e. Linux and turns it into a standalone hypervisor. It then uses the existing capabilities of Linux to allocate memory, provide security and schedule processes to give right amount of time to each VM. It doesn't need to reinvent the wheel; Linux already provides these functions and does them very well. What is Virtual Machine Migration? Virtual Machine Migration is a technique in which a VM is transferred from one physical server to another along with its memory states.

5. KVM Architecture

Under KVM, virtual machines are created by opening a device node (/dev/KVM.) A guest has its own memory, separate from the userspace process that created it. A virtual CPU is not scheduled on its own, however./dev/KVM Device Node KVM is structured as a fairly typical Linux character device. It exposes a /dev/KVM device node which can be used by userspace to create and run virtual machines through a set of ioctl()s [Kivity et al.,(2007)]. The operations provided by /dev/KVM include: • Creation of a new virtual machine. • Allocation of memory to a virtual machine. • Reading and writing virtual CPU registers. • Injecting an interrupt into a virtual CPU. • Running a virtual CPU.

Figure 1 shows how guest memory is arranged. Like user memory in Linux, the kernel allocates discontiguous pages to form the guest address space. In addition, userspace can mmap() guest memory to obtain direct access. This is useful for emulating dma-capable devices. Running a virtual CPU deserves some further elaboration. In effect, a new execution mode, guest mode is added to Linux, joining the existing kernel mode and user mode.

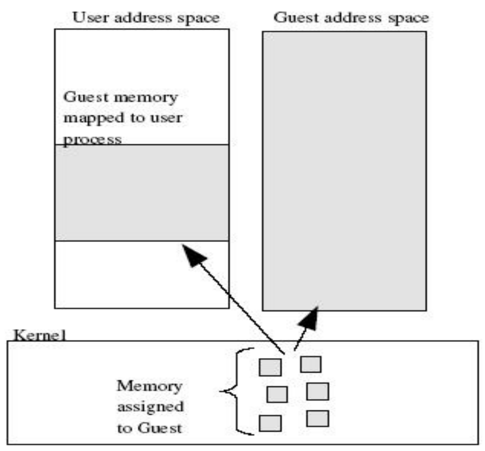


Figure 1 : KVM Memory Map[Kivity et al., (2007)]

Guest execution is performed in a triply-nested loop: At the outermost level, userspace calls the kernel to execute guest code until it encounters an I/O instruction, or until an external event such as arrival of a network packet or a timeout occurs. External events are represented by signals. • At the kernel level, the kernel causes the hardware to enter guest mode. If the processor exits guest mode due to an event such as an external interrupt or a shadow page table fault, the kernel performs the necessary handling and resumes guest execution. If the exit reason is due to an I/O instruction or a signal queued to the process, then the kernel exits to userspace. • At the hardware level, the processor executes guest code until it encounters an instruction that needs assistance, a fault, or an external interrupt. Refer to Figure 2 for a flowchart-like representation of the guest execution loop. Linux Integration Being tightly integrated into Linux confers some important benefits to KVM: • On the developer level, there are many opportunities for reusing existing functionality within the kernel, for example, the scheduler, NUMAsupport, and high-resolution timers. • On the user level, one can reuse the existing Linux process management infrastructure, e.g., top(1) to look at CPU usage and taskset(1) to pin virtual machines to specific CPUs. Users can use kill(1) to pause or terminate their virtual machines.

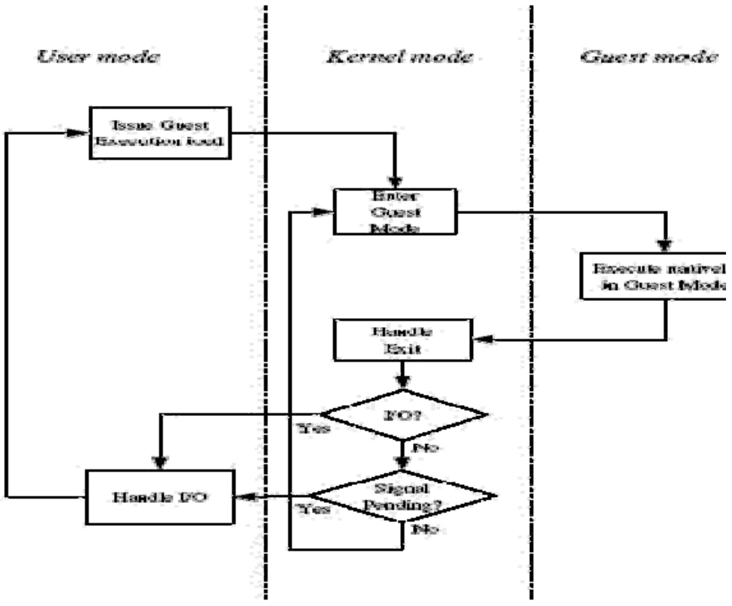


Figure 2: Guest Execution Loop[Kivity et al., (2007)]

6. When to Migrate ?

[Mishra et al.,(2012)] There are many situations when migration of VMs becomes necessary to maintain the overall efficiency of the data center. These situations or triggers are shown in figure 3 and discussed next.

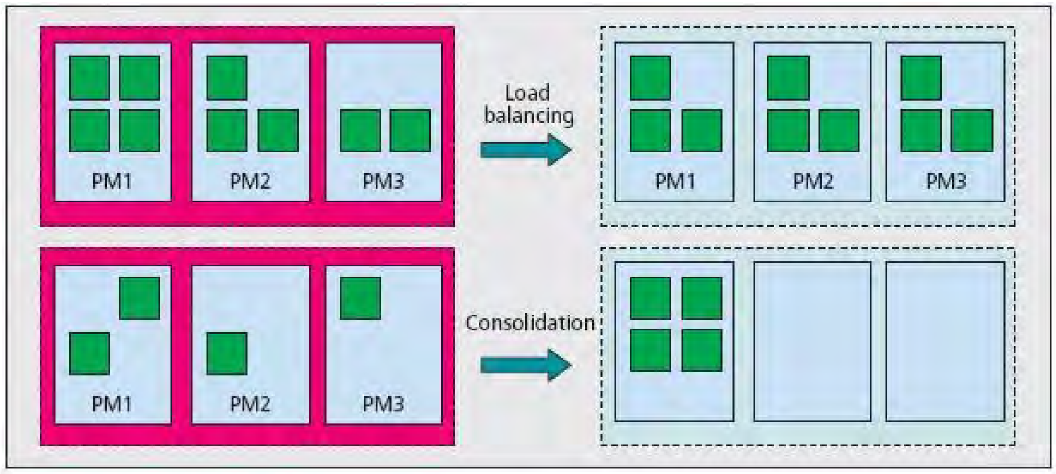


Figure 3: Load balancing and consolidation scenarios [Mishra et al., (2012)]

6.1. *Periodic*

The migrations in a data center can be triggered periodically. For example, data centers in one part of world may be heavily used in daytime (9 a.m. to 9 p.m.), whereas they may be underloaded during the night. Such “time of day” based migration of VMs ensures that VMs are “near” clients, and the communication delays and overheads are minimized. Migrations can also be done periodically to consolidate the reduced loads.

6.2. Due to hot spot

A hot spot is the overloaded condition of a PM. It can also be defined as the state when performance of a system falls below the minimum acceptance level. Detection of a hot spot can be done both proactively and reactively. Proactive hot spot detection techniques predict the occurrence of a hot spot by analyzing the trends in resource utilizations of the VM. If the resource utilization shows an increase for some time window, it is likely that it may result in a hot spot in the future. Such timeseries analysis-based techniques help avoid hot spots even before they occur. One such technique to predict CPU utilization is discussed in. More sophisticated proactive techniques analyze the request arrival rates. Increase in request arrivals suggests that the VM will require more resources to fulfill them, thus causing a potential hot spot. Reactive hot spot detection techniques use more direct techniques like observing the page thrashing rate, CPU and memory utilization levels, and so on. Hot spots can be locally mitigated if enough capacity is available at the [Ibrahim et al., (2010)] host PM. Extra resources can be allocated to the VM showing signs of overload. When extra capacity is not available locally, migration is the only option available.

6.3. Excess spare capacity

Low utilization of PMs results in resource wastage. An optimum level of utilization is required to be maintained for the efficient working of a data center. Physical machines that have excess spare capacity (i.e., low resource utilization) cause overall inefficiency in the data center. At the level of a PM,the hypervisors have monitoring tools, similar to normal operating systems, which can provide the utilization information of different resources for that machine. Resource utilization levels of PMs across a data center are continuously monitored, and whenever the utilization levels fall below a certain threshold, migrations can be triggered. When a number of PMs are underutilized, VMs are migrated from such machines to make them completely free. Such “freed” PMs can then be shut down to save power, which results in consolidation.

6.4. Load Imbalance

Virtual machines change their resource requirements dynamically. This dynamism leads to imbalances in the resource utilization levels of different PMs. Some PMs can get heavily loaded while others may be lightly loaded. In a data center, resource utilization levels of PMs are monitored continuously. If there is large discrepancy in the utilization levels of different PMs, load balancing is triggered. Load balancing involves migration of VMs from highly loaded PMs to low loaded ones. An overloaded PM is undesirable as it causes delays in service of user requests. Similarly, the PMs that are lightly loaded cause inefficient resource utilization.

7. Types of Migration

There are 2 types of migration:

2.1) Live: Live migration [Clark et al., (2005)] describes the process of copying a VM from one physical machine to another physical machine, while the VM is still powered on [Anand et al., (2012)]. It provides special benefit to server virtualization and has become a significant tool for a variety of scenarios.

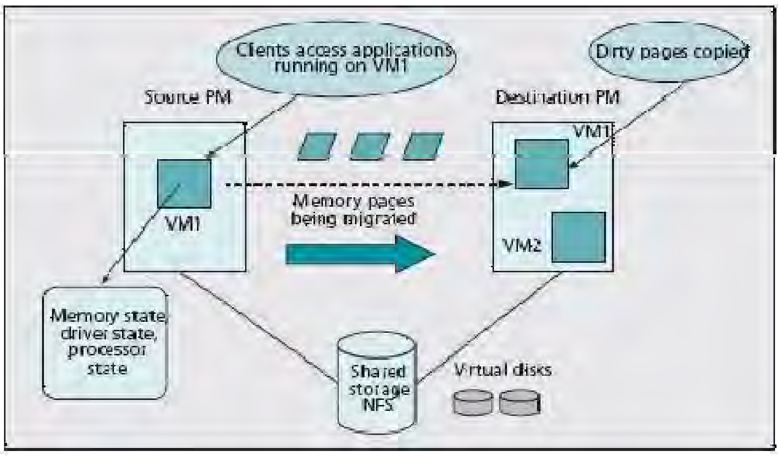

Clients access applications running on VM1
Dirty pages copied
Source PM
Destination PM
VM1
Memory pages being migrated
VM1
VM2
Memory state driver state, processor state
Shared storage NFS
Virtual disks

Figure 4: Live VM Migration process[Mishra et al., (2012)]

2.2) Non-live: VM being transferred stops its processes during migration and resume them at destination only after the completion of migration.

8. Types of Live Migration

Live Migration can further be of two types:

8.1. Pre-copy live migration

The bulk of the VM's memory state is migrated to a target node even as the VM continues to execute at a source node. If a transmitted page is dirtied, it is re-sent to the target in the next round. This iterative copying of dirtied pages continues until either a small, writable working set (WWS) has been identified, or a preset number of iterations is reached, whichever comes first. This constitutes the end of the memory transfer phase and the beginning of service downtime. The VM is then suspended and its processor state plus any remaining dirty pages are sent to a target node. Finally, the VM is restarted and the copy at source is destroyed [Ibrahim et al., (2010)].

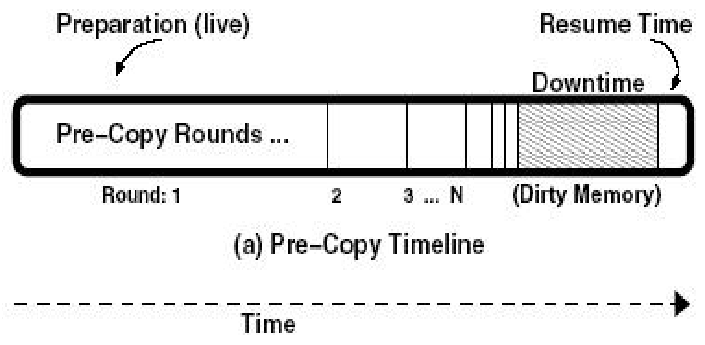


Figure 5: Timeline for pre-copy[Ibrahim et al., (2010)]

Pre copy consists of the following three phases (see figure 6): 1) Pre-Copy Phase: At this stage, the VM continues to run, while its memory is iteratively copied page wise from the source to the target host. Iteratively means, the algorithm works in several rounds. It starts with transferring all active memory pages. As each round takes some time and in the mean time the VM is still running on the source host, some pages may be dirtied and have to be re-sent in an additional round to ensure memory consistency. 2) Pre-Copy Termination Phase: Without any stop condition, the iteratively pre-copy phase may carry on indefinitely. Stop conditions depend highly on the design of the used hypervisor, but typically take one of the following thresholds into account: (1) the number of performed iterations exceeds a pre-defined threshold, (2) the total amount of memory that has already been transmitted, exceeds a pre-defined threshold or (3) the number of pages dirtied in the previous round falls below a pre-defined threshold. 3) Stop-and-Copy Phase: At this stage the hypervisor suspends the VM to stop page dirtying and copies the remaining dirty pages as well as the state of the CPU registers to the destination host. After the migration process is completed, the hypervisor on the target host resumes the VM.[Strunk, (2012)] Figure 6: Iterative rounds in pre-copy live migration[Ibrahim et al., (2010)]

8.2. Post-copy live migration

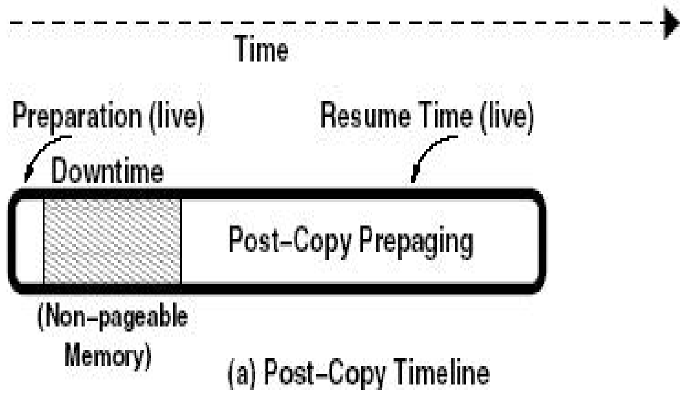


Figure 7: Timeline for Post-copy

In the basic approach, post-copy first suspends the migrating VM at the source node, copies minimal processor state to the target node, resumes the virtual machine, and begins fetching memory pages over the network from the source. The manner in which pages are fetched gives rise to different variants of post-copy, each of which provides incremental improvements. Post-copy ensures that each page is sent over the network only once, unlike in pre-copy where repeatedly dirtied pages could be resent multiple times [Ibrahim et al., (2010)]. Figure 7: Timeline for Post-copy[Ibrahim et al., (2010)] Variants of Post Copy Post-Copy via Demand Paging: The demand paging variant of post-copy is the simplest and slowest option. Once the VM resumes at the target, its memory accesses result in page faults that can be serviced by requesting the referenced page over the network from [Kivity et al., (2007)] the source node. However, servicing each fault will signifi-cantly slow down the VM due to the network’s round trip latency. Consequently, even though each page is transferred only once, this approach considerably lengthens the resume time and leaves long term residual dependencies in the form of unfetched pages, possibly for an indeterminate duration. Thus, post-copy performance for this variant by itself would be unacceptable both from the viewpoint of total migration time and application degradation. Post-Copy via Active Pushing: One way to reduce the duration of residual dependencies on the source node is to proactively “push” the VM’s 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. Thus, each page is transferred only once, either by demand paging or by an active push. Post-Copy via Pre-paging: The goal 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, our approach works by using the faulting addresses as hints to estimate the spatial locality of the VM’s memory access pattern. The pre-paging component then shifts the transmission window of the pages to be pushed such that the current page fault location falls within the window. This increases the probability that the pushed pages would be the ones accessed by the VM in the near future, reducing the number of major faults. Hybrid Pre and Post Copy: The hybrid approach, works by doing a single pre-copy round in the preparation phase of the migration. During this time, the guest VM continues running at the source while all its memory pages are copied to the target host. After one iteration only, the VM is suspended and its processor state and dirty pages are copied to the target. Subsequently, the VM is resumed and the post-copy mechanism described above kicks in immediately, bringing in the remaining dirty pages from the source. As with pre-copy, this scheme can perform well for read-intensive workloads. Yet it also provides deterministic total migration time for write-intensive workloads, as with postcopy.[Ibrahim et al., (2010)] 1.9 Parameters that we intend to study 1. Downtime: This is time during which the migrating VM’s execution is stopped. 2. Resume Time: This is the time between resuming the VM’s execution at the target and the end of migration altogether, at which point all dependencies on the source must be eliminated 3. Total Migration Time: This is the sum of all the above times from start to finish. Total time is important because it affects the release of resources on both participating nodes as well as within the VMs on both nodes. Until the completion of migration, we cannot free the source VM’s memory

9. Benefits of KVM Hypervisor

KVM or Kernel Based Virtual Machine is a virtualization infrastructure for the Linux kernel which turns it into a hypervisor. It utilizes the hardware extensions and has found its way in Linux kernel. It is a full virtualization solution, which requires no changes in guest operating system. KVM is the future. It is built into the Linux kernel and also has the same performance and speed as XEN. People who have deployed XEN since earlier days and it takes a lot of pain to compile and install everything in XEN. Also, KVM is newer code: when it started being designed four years ago Intel and AMD were already extending their processor instruction sets to add virtualization-supporting instructions. Early hypervisors like ESX had to cope with the previous generation of CPU chips, which had no virtualization support, and so were larger and made less efficient use of server resources. Comparing it with VMware, the rapid maturation of KVM, or Kernel-based Virtual Machine, over the course of the last couple of years constituted the first open-source challenge to it. Integrated into the Linux kernel, KVM provides feature-rich and highly efficient virtualization. Windows is the only platform on which VMware's graphical management infrastructure is supported. KVM, in contrast, can be managed via the command line, via graphical interfaces (running either on the local machine or forwarded over SSH to a remote workstation).

10. Why Post-Copy over Pre-Copy?

1. In post copy live migration, downtime is less as compared to pre-copy live migration.
2. Using post-copy live migration ensures that each page is transferred only once whereas in pre-copy, pages are migrated iteratively till the number of dirtied pages remains very less. So, the network traffic is reduced to a large extent in case of post-copy as compared to pre-copy.

3. Post copy is applicable for:
    - ● Planned maintenance
    - ● Predictable total migration time is important
    - ● Dynamic consolidation
    - ● In cloud use case, usually resources are over-committed
    - ● If machine load becomes high, evacuate the VM to other machine promptly (Precopy optimization (= CPU cycle) may make things worse)
    - ● Wide area migration
    - ● Not all network bandwidth can be used for migration
    - ● Network bandwidth might be reserved by QoS

11. Proposed Algorithm

```
1. Input: VMem, VMin, Tfault, R, P, Y Output: Tmig, Tdown, Tres
2. VMig ⇓ VMin, Tmig ⇓ 0, Tdown ⇓0, Tres ⇓0
3. Tdown ⇓ VMig/R
4. VMig ⇓ VMem
Without Pre-Paging
5. X ⇓(VMem – VMin)/P
6. Tres ⇓ X*Tfault + ( VMem- VMin) /R
7. Tmig ⇓ VMem( 1 + α + α(Tfault/P))/R
With Pre-Paging
5. X’ ⇓(VMem – VMin)/(1+Y)P
6. Tres ⇓ X’*Tfault + ( VMem- VMin) /R
7. Tmig ⇓ VMem( 1 + α + α(Tfault/(1+Y)P))/R Where α= (VMem/VMin)-1
```

Table 1: List of parameters

PARAMETER	DESCRIPTION
VMEM	Current size of Virtual Machine memory
VMIG	Total network traffic
VMIN	Minimum memory required to transfer Virtual Machine to destination
TMIG	Total migration time
TDOWN	Total Downtime
TRES	Time to handle a fault
R	Memory transmission rate
P	Size of memory page
Y	Number of extra pages transferred at the time of each fault (in pre-paging)
X	Number of network faults(without pre-paging)
X’	Number of network faults(with pre-paging)

12. Mathematical Modelling

As we know in post-copy live migration, downtime is the phase where a minimum amount of memory is transferred from source to target in the beginning so that after the transfer, VM can be resumed at the destination.

Hence, downtime

Tdown = VMin /R (1) Where, R= memory transmission rate.

After this,

$$VMig = VMem \qquad (2)$$

Because demand paging is used.

Without Pre-Paging

Remaining memory now is (VMem-VMin).

Now, if X is the number of Network Faults,

Then Resume Time i.e Time after resuming the VM on destination till end of migration can be calculated as

Tres = X*Tfault + (VMem-VMin)/R …(3) Where, Tfault = time to handle a page fault.

If P is the size of page on both the machines,

Then,

$$X = \text{Total page faults} = (VMem - VMin)/P \qquad (4)$$

So, Total Migration Time can be calculated as

Tmig = Tdown + Tres

= VMin/R + (VMem-VMin)/R + X*Tfault Putting

Value of X from (4),

Tmig = VMin/R + (VMem-VMin)/R + ((VMem-VMin)/P)*Tfault

=VMin(1 + ((VMem/VMin) -1) + ((VMem/VMin) -1)*(Tfault*P))/R (5)

Let ((VMem/VMin) -1)= α

Hence,

$$Tmig = VMin(1 + \alpha + \alpha(Tfault/P))/R \qquad (6)$$

With Pre-Paging

Now, as we know that in case of pre-pagin , whenever a fault occurs , nearby pages of the faulting address are also transferred along with the faulted page

Let Y be the number of pages transferred at the time of a fault, for each fault. Let X' be the total page faults

Total pages to be transferred = (VMem - VMin) / P = X So,

X = X' + X' * Y

= X' (1 + Y) (7)

So,

(VMem - VMin) / P = X' (1+Y)

Therefore,

$$X' = (VMem - VMin)/ (P(1 + Y)) \quad (8)$$

As we know,

$$Tres = X' * Tfault + (VMem - VMin) / R$$

Therefore, from (8)

$$Tres = ((VMem - VMin)/ (P(1 + Y))) * Tfault + (VMem - VMin) /R \quad (9)$$

Tmig = Total migration time = Tdown + Tres

Therefore ,

$$Tmig = VMin /R + ((VMem - VMin)/ (P * (1 + Y))) * Tfault + (VMem - VMin) /R \ldots( )$$

So,

$$Tmig =VMem(1 + \alpha + \alpha(Tfault/(1+Y)P))/R \quad (10)$$

13. Implementation Details

13.1 Performance analysis

We measured migration time of live migration. In this experiment, virtual machines have no running process on them. Also Memory size of a virtual machine and migration time are proportional. The experimental result for this is shown in Figure 9.

13.1.1 Analysis of total migration time of virtual machine running different processes

We migrated virtual machines on which different processes were running in different cases. We migrated virtual machine on which a word document was running. We varied the size of the document to analyze the effect on total migration time, as shown in figure. The vertical axis depicts the total migration time in milliseconds (ms) and the horizontal axis depicts the size of the process running on the virtual machine.

Similarly, virtual machines with a word document, spreadsheet process, PowerPoint presentation, image files, video files, Mozilla Firefox browser, running on it, one process per case , was performed , the size of the documents was varied and we analyzed the total migration time, as shown in figures 10(a), 10(b), 10(c), 10(d), 10(e), 10(f) respectively.

13.1.2 Analysis of total migration time of virtual machine running several processes simultaneously

We migrated virtual machine running several processes at the same time. In one case the virtual machine has a word document and an excel sheet running on it and in another case the virtual machine has a word document, excel file, Mozilla Firefox browser and the Linux terminal running on it. We noted the total migration time for both the cases. The total migration time increases exponentially as the number of processes are increased. The result is shown in Figure 12.

13.1.3 Analysis of total migration time of virtual machine running cpu intensive process

We migrated virtual machines on which a CPU intensive process is running. The virtual machine has a process which is infinitely increasing the value of a count variable. The Total Migration time has been compared with that of a virtual machine running a I/O intensive process, as shown in figure .The I/O intensive process reads the contents of a file and writes it to other file.

The vertical axis depicts the total migration time of the machines. In the case of CPU-intensive process, the migration takes sec, and in case of I/O intensive process, migration takes sec. These results demonstrated that migration of a virtual machine with an I/O intensive process running on it takes more time than migration of a virtual machine with CPU intensive process running.

13.1.4 Comparison of total migration time over wired network and wireless network

We migrated virtual machine with no process running on it, first over a wired network with speed 90 Mbps and then over wireless network of the same speed and compared the total migration time for both the cases. In case of wired network , the total migration time is sec and in wireless network it is sec. The total migration time increases over wireless network.

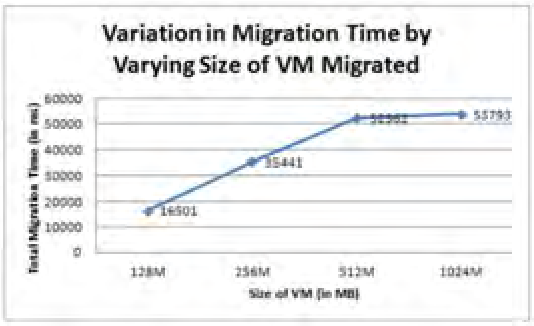
13.1.5 Comparison of total migration time over wired network varying the network bandwidths.

We migrated virtual machine over wired networks of different speeds,924 Mbps, 512 Mbps, 256 Mbps and 128 Mbps. The total migration time in case of 924 Mbps network bandwidth was sec, 512 Mbps was sec, 256 Mbps was sec and 128 Mbps was sec. Figure shows the graphical analysis of the above speeds.

13.1.6 Comparison of total migration time over wireless network of different bandwidths.

We migrated virtual machine over wireless networks of different speeds, 924 Mbps, 512 Mbps, 256 Mbps and 128 Mbps. The total migration time in case of 924 Mbps network bandwidth was sec, 512 Mbps was sec, 256 Mbps was sec and 128 Mbps was sec. Figure shows the graphical analysis of the above speeds.

Figure shows the graphical comparison of the total migration time over wired and wireless networks of different speeds.



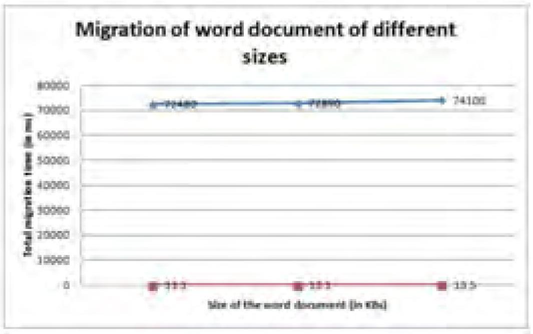
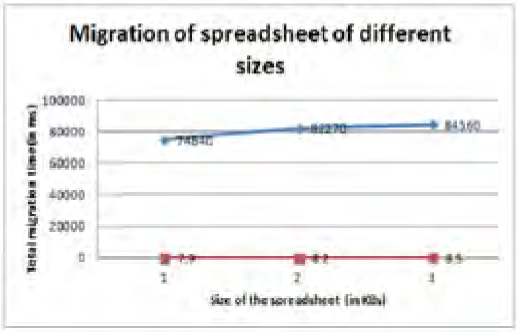


Figure 9: VM Memory v/s Total Migration Time.



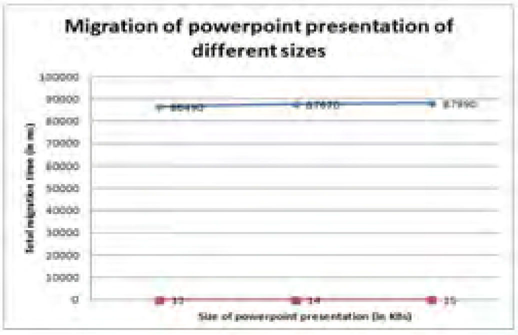
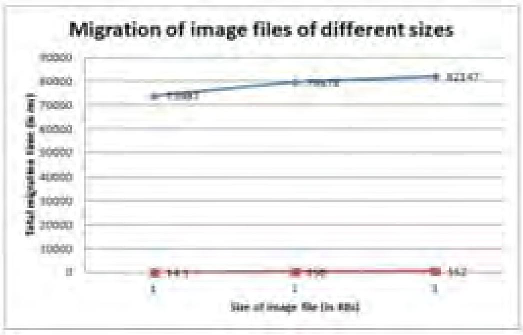


Figure 10(a): Migration Time v/s size of spadsheet

Figure 10(b): Migration Time v/s size of Presentation.



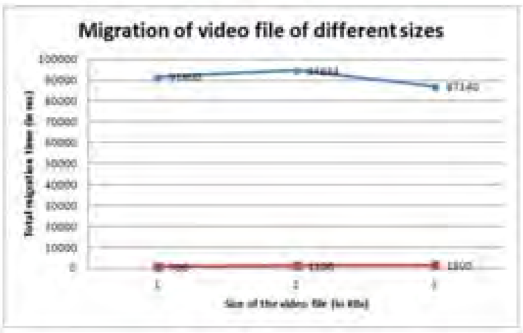


Figure 10(c): Migration Time v/s size of Image File.

Figure 10(d): Migration Time v/s size of Video File.

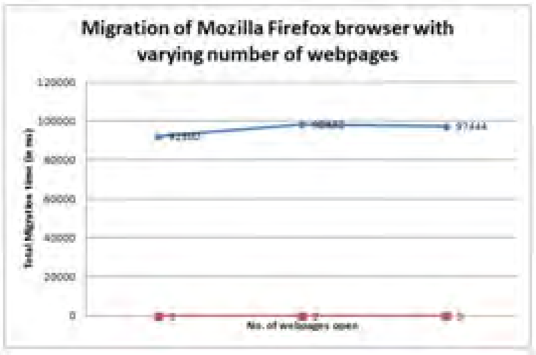


Figure 10(e): Migration Time v/s size of Web Browser.

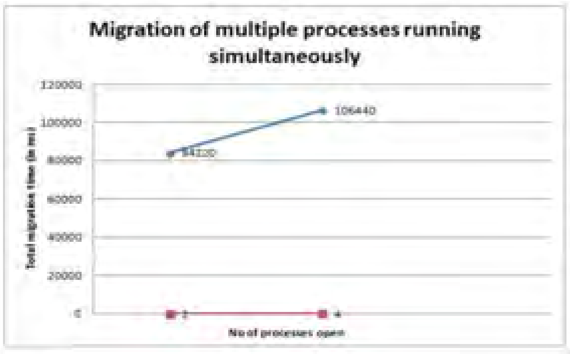


Figure 12: Migration of multiple processes simultaneously

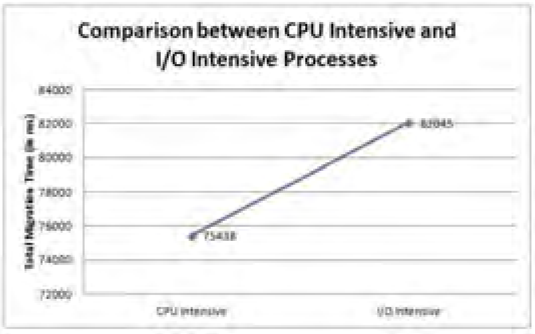


Figure 13: Comparison between CPU Intensive and I/O Intensive Process.

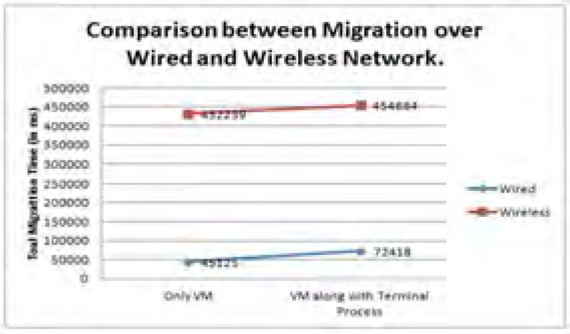


Figure 14: Comparison between Migration over Wired and Wireless Network.

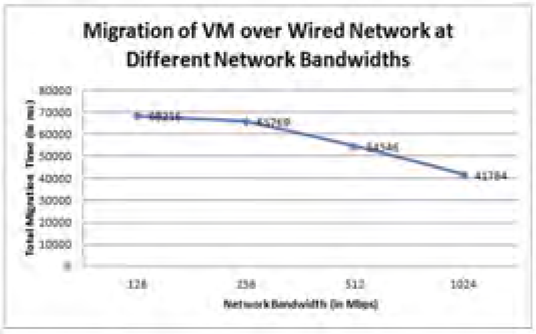


Figure 15: Migration of VM over Wired Network at Different Network Bandwidths.

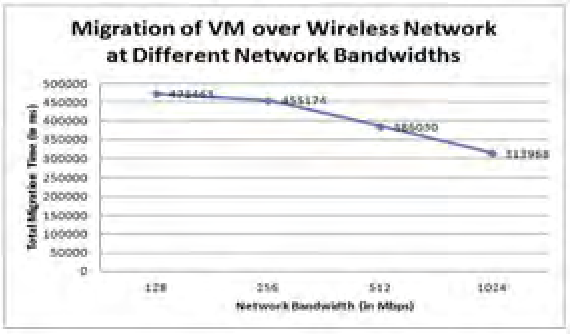


Figure 16: Migration of VM over Wireless Network at. Different Network Bandwidths

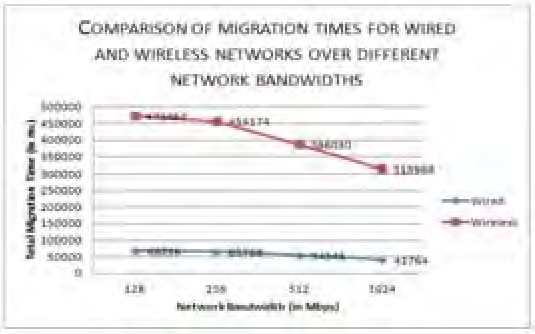


Figure 17: Comparison of Migration Times for Wired and Wireless Networks over Different Network Bandwidths.

14. Simulation Framework

14.1. Experimental setup

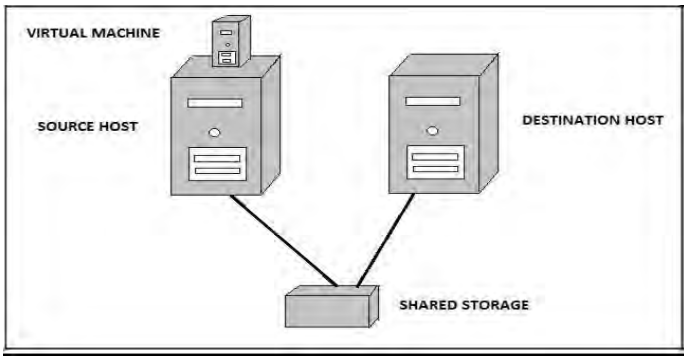


Figure 18: Experimental Setup

14.2. Systm specification

Table 2: Specification of the Computers

	OS	CPU	Memory (MB)	HDD (GB)	Network (Mbps)
Source Host	Ubuntu 12.04.3, 64-bit	Core i5 480m, 2.67 GHz	4092	500	90
Destination Host	Ubuntu 12.04.3, 64-bit	Core i3 330m, 2.13 GHz	4092	250	90
VM	Ubuntu 12.04.3, 64-bit	QEMU Virtual CPU version 1.7.0, 2.67 GHz	924	9	90

.

14.3. Softwares used

14.3.1 Softwares used

Ubuntu is a Debian-based operating system based on free software. It provides Linux server, desktop, phone, tablet and TV operating systems.Ubuntu releases updated versions predictably - every six months and that each release would receive free support for nine months with security fixes, other high-impact bug fixes and very conservative, substantially beneficial low-risk bug fixes. The first release was on October 2004.

14.3.2. QEMU(for post-copy migration)

QEMU (short for "Quick EMUlator") is a free and open-source hosted hypervisor that performs hardware virtualization. QEMU is a hosted virtual machine monitor: It emulates central processing units through dynamic binary translation and provides a set of device models, enabling it to run a variety of unmodified guest operating systems. It also provides an accelerated mode for supporting a mixture of binary translation (for kernel code) and native execution (for user code), in the same fashion asVMware Workstation and VirtualBox do. QEMU can also be used purely for CPU emulation for user-level processes, allowing applications compiled for one architecture to be run on another.

15. Result

Post-copy live migration of virtual machine using KVM hypervisor has been successfully studied and implemented. The impact of varying process size, VM size, network speed and type of network has been studied and analyzed with appropriate comparisons. The experimental values have been verified with the proposed cost evaluation model.

16. Conclusion and Future Work

We have studied the concept of live migration, the issues that need to be considered while migrating, i.e. when to migrate, which VM to migrate and where to migrate. We have theoretically studied pre-copy and post-copy live migration and the various parameters like total migration time, downtime, resume time for pre-copy live migration using Xen hypervisor and KVM hypervisor. We have studied the KVM hypervisor and it's architecture. Based on our study, we have proposed an algorithm and a cost evaluation model to evaluate total migration time, downtime and resume time for post copy live migration using KVM hypervisor , which has not been studied yet as per the best of our knowledge . We have proved our proposed model with proper mathematical modeling. We calculated the Total Migration Time for post-copy live virtual machine migration using KVM hypervisor for different scenarios: tion(1) varying the size of processes running on virtual machine, (2) migrating the VM over wired network, (3) migrating the VM over wireless network, (4) Migrating the VM Over networks of different speed. We verified experimentally the obtained values with the proposed algorithm with an acceptable percentage error of 4%. We also calculated the Total Downtime for all the cases from the proposed cost evaluation model. As future work, the comparison of the total migration time can be done for post-copy live migration using KVM, Xen, VMware and other hypervisors. Comparison of pre-copy and postcopy live migration using KVM hypervisor can be performed.

References

[1] Mayank Mishra, Anwesha Das, Purushottam Kulkarni, and Anirudha Sahoo, “Dynamic Resource Management Using Virtual Machine Migrations” - IEEE Communications Magazine, Volume 50, 2012.

[2] Yosuke Kino, Kenichi Nii, Saneyasu Yamaguchi, “A Study on Performance of Processes in Migrating Virtual Machines” The Fourth International Workshop on Ad Hoc, Sensor and P2P Networks (AHSP2011) (2011/06).

[3] Anja Strunk, “ Costs of Virtual Machine Live Migration: A Survey”, Services (SERVICES), 2012 IEEE Eighth World Congress on, june 2012, pp. 323 –329.

[4] Xiujie Feng*, Jianxiong Tang, Xuan Luo, Yaohui Jin, “A performance study of live VM migration technologies: VMotion vs XenMotion”, IEEE, 2011.

[5] Khaled Z. Ibrahim, Steven Hofmeyr, Costin Iancu, Eric Roman, “Optimized pre-copy live migration for memory intensive applications”, SC '11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis.

[6] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan, “Post-Copy Live Migration of Virtual Machines”, ACM, July 2009.

[7] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, Xiaofei Liao, “Performance and Energy Modeling for Live Migration of Virtual Machines” ACM 2011.

[8] Avi Kivity, Yaniv Kamay and Dor Laor, “KVM: the Linux Virtual Machine Monitor”, Ottawa Linux Symposium.

[9] Ankit Anand, Mohit Dhingra, J. Lakshmi, S. K. Nandy, “Resource usage monitoring for KVM based virtual machines”, 2012 18th International Conference on Advanced Computing and Communications (ADCOM)

[10] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen†, Eric Jul†, Christian Limpach, Ian Pratt, Andrew Warfield, “Live Migration of Virtual Machines”, ACM 2005.

[11] KVM, http://www.linux-KVM.org/page/Main_Page

[12] Qemu download, http://wiki.qemu-project.org/Download