International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.2, Issue.4, July-Aug. 2012 pp-1564-1571 ISSN: 2249-6645

Parameter Dependent Performance Optimization In Live Migration Of Virtual Machine

K. Phaneendra[1], Venu Madhav Sunkara[2], R. Vijaya[3], G. Rajendra[4]
*(Department of MCA, LBRCE College/ JNTU Kakinada, India)
** (Department of Computer Science, KL University, Vaddeswaram)
*** (Department of MCA, LBRCE College/ JNTU Kakinada, India)
**** (Department of MCA, LBRCE College/ JNTU Kakinada, India)

ABSTRACT: *Virtualization technology is being used intensively in data centers, cluster systems, enterprises and organizational networks, Hence the capability of Virtual Machine (VM) Migration importance has been increased for maintaining high performance, improved manageability and fault tolerance. Live Migration allows virtual machine monitor to move the running virtual machine from one physical server to another with zero downtime, continuous service availability and complete transaction integrity. In this paper we, present a performance evaluation of parameters that affect live migration and varying in the performance depending on workload.*

Keywords: *Virtual Machine (VM), Migration Time (MT), Down Time (DT), Dirty Page, Virtualization.*

I. INTRODUCTION

System virtualization is the ability to abstract and pool resources on a physical platform. This abstraction decouples software from hardware and enables multiple operating system images to run concurrently on a single physical platform without interfering with each other. As a technique, system virtualization has existed for decades on mainframes. In the past, industry standard x86-based machines, with their limited computing resources. Virtualization can increase utilization of computing resources by consolidating the workloads running on many physical systems into virtual machines running on a single physical system. Virtual machines can be provisioned on-demand, replicated and migrated. Virtual machine (VM) migration, which is the ability to move a VM from one physical server to another under virtual machine monitor (VMM) control, is a capability being increasingly utilized in today's enterprise environments.

Implemented by several existing virtualization products, live migration can aid in aspects such as high-availability services, transparent mobility, consolidated management, and workload balancing [1]. While virtualization and live migration enable important new functionality, the combination introduces novel security challenges. A virtual machine monitor that incorporates a vulnerable implementation of live migration functionality may expose both the guest and host operating system to attack and result in a compromise of integrity. Given the large and increasing market for virtualization technology, a comprehensive understanding of virtual machine migration security is essential. However, the security of virtual machine migration has yet to be analyzed.

After a live migration, guest software continues to maintain an identical view of the pre and post migration hardware. In this paper we discuss Processors provide support for a VMM to hide differences in software-visible processor features during Live.

1.1 SHORT BACKGROUND ON VM TECHNOLOGY

Initially very popular in the 1960's, for instance in the context of shared mainframe computers, this technology was subsequently abandoned in favour of multiprogrammed efficient commodity operating systems, running on increasingly cheaper and more widely available hardware. However, the virtual machine technology, initially based on the principle of exporting (possibly multiple) virtualized software versions of the machine hardware to upper layers (originally the operating system) came back to fashion in the past years. Reasons for this come-back spanned the need for easier management of large scale distributed systems or MPP machines [2], or the need for support for mobility (the easy checkpointing capability mentioned above) and increased security (as many commodity OSs had become quite unmanageable and/or insecure and proved to under-use the same increasingly cheaper hardware resources that caused the "retirement" of VMs 20 years ago).However, while they have indeed come back, VMs are less interesting now for resource multiplexing but more as a way to "circumvent" existing "popular" Operating Systems that have become unmanageable and provide little opportunities for activities like checkpointing or sandboxing. Issues and challenges of VM implementations include minimizing virtualization overhead and exporting a virtualized interface identical or as similar as possible to the virtualized machine to ensure compatibility.

Various design choices exist, such as providing a "classic" VM architecture – such as Xen or VMWare ESX Server [3] (laying underneath the Operating system and thus maximizing performance) or a "hosted" architecture, such as VMWare Workstation (laying "on top" of a host operating system, as an application and improving). Another important design choice is that of slightly modifying the virtualized interface to be exported to replace portions of the instructions set which are not easily virtualizable by different and more efficiently implementable equivalents. This approach is called paravirtualization and it is applied in VM implementations such as Xen or Disco [4]. VMs can provide important benefits that can be useful in many contexts, including migration. Firstly, they have the serious advantage of abstracting away the details of underlying hardware and exporting a uniform view of the virtualized machine, therefore providing an elegant solution to the resource heterogeneity problem. Additionally, they provide a complete encapsulation of the machine software state, therefore the VM can be easily and very conveniently checkpointed, suspended and restarted at will. Consequently, VMs can be dynamically mapped to physical machines or migrated much easier than processes.

Therefore, the implementation complexity of migration implementation, which we saw that was a limiting factor in the case of process migration, is greatly reduced. Besides migration-empowered applications like dynamic load balancing, fault tolerance or Internet Suspend-Resume [5] type applications, VM can be used for things like convenient distribution of software packages (for instance Oracle delivers packages of readily installed and configured software under the form of VMs) or for damage containment and “forensics” against worm or hacker attacks.

1.2 VIRTUAL MACHINE MIGRATION

There are many ways to migrate a VM. In *static migration*, the VM is shutdown using OS-supported methods; its static VM image is moved to another VMM and restarted. In *cold migration*, the VM is suspended using OS supported or VMM-supported methods. The suspended VM image is moved to a VMM on a different machine and resumed. In *live migration*, the VMM moves a running VM instance nearly instantaneously from one server to another. Live Migration allows for dynamic load balancing of virtualized resource pools, hardware maintenance without downtime and dynamic failover support. As long as the hardware in the pre and post migration environment is identical, guest software should behave in exactly the same way before and after the migration. It is when guest software runs in a different hardware environment after a migration that certain challenges can arise[6]. Note that even though a VMM presents a virtual platform to guest software, there could be certain interfaces, depending on VMM design, which guest software can directly use to determine underlying hardware’s capabilities.

After the reboot/restart following a static migration, guest software should go through its platform discovery phase and be able to adjust to any differences in underlying (virtual) hardware. Following a cold migration, guest software may continue to maintain an identical view of the pre and post migration hardware. When suspended using OS-supported methods, some operating systems will re-scan the hardware upon resume. Depending on their policies and the hardware differences between current and previous hardware, the OSes may refuse to resume and require a reboot. After a live migration, guest software continues to maintain an identical view of the pre and post migration hardware.

1.3 Issues with VM migration

However, things get a little more complicated. More precisely, to perform a correct migration, besides the checkpointed state of the VM, the memory image of that VM also has to be migrated, for the state to be correctly preserved. All this should be done while programs in the VM are still running; therefore memory pages are still getting dirtied. Therefore, we see that increased simplicity comes at a certain price, since the VM’s memory image and state is undoubtedly much larger than the process’ checkpointed state in the case of process migration.

Additionally, as with all migrations, resources used by processes running within the migrated VM should still be available after the migration attached. Since these resources might be hard to migrate (because of large sizes or consistency constraints for instance), this brings back the problem of residual dependencies. For instance, the problem of migrating the file system present on the virtual disk of a Virtual Machine [7]. In the context of Virtual Machine migration, “residual dependencies” are especially important, considering that the size of a virtual machine can be much larger than that of a process. While migrating the entire VM, as we have seen, has the advantage that support for check pointing is readily provided, unlike in the case of a regular process, the VM’s address space, and especially it’s virtual disk are of considerable size, therefore leaving residual dependencies may be unavoidable to ensure reasonable migration times (at least with current typical network resources)[8]. As with all migration systems, transparency remains an issue also for VM migration.

II. LIVE MIGRATION

Virtual machine live migration is a virtualization process that moves a virtual machine (VM) from one physical host server to another. It moves the memory and state of a VM without shutting down the application, so users will generally not detect any significant interruption in application availability. The process captures the complete memory space occupied by the VM along with the exact state of all the processor registers currently operating on the VM then sends that content across a TCP/IP link to memory space on another server. Processor registers are then loaded, and the newly moved VM can pick up its operation without missing a step.

Most VM live migrations occur between similar hypervisors, so the migrated VM retains its name and other unique identifiers. Even though the VM is on a different server, it's the exact same machine as far as the users are concerned. Live migration is a key benefit of virtualization, allowing workloads to move in real time as server or data center conditions change [9]. Consider the impact on business continuity: A virtual server scheduled for maintenance can migrate its workloads to a spare server or to other servers that have extra computing capacity. Once the maintenance is complete and the server returns to service, these workloads can all migrate back to the original server without disruption.

Live migration helps server consolidation by allowing IT administrators to balance workloads across data center servers, ensuring that each server is used efficiently without being overtaxed. Live migration helps with disaster recovery too because VMs can just as easily be moved from one site to another, relying on spare servers at a remote site to receive and operate the migrated VMs.

All of the major virtualization software platforms include VM live migration tools. These include VMware VMotion (part of vSphere), Microsoft Live Migration (part of Hyper-V R2) and Citrix Systems XenServer live migration. Migration tools typically allow administrators to prioritize the movement of each VM so that failover and failback processes occur in a predictable and repeatable manner. Mission-critical VMs usually take priority and are often moved to spare servers with ample computing resources. Secondary VMs can be addressed next, although the migration software may be left to move noncritical VMs automatically based on the computing resources on each available server. Migration audits allow administrators to locate VMs and track their movements to refine and optimize ongoing migration behaviours. Live migration works between almost all virtual host servers, but it's important to test migration behaviours between servers with various processor manufacturers. Processors from Intel and AMD both include extensions that provide hardware assistance for virtualization tasks, including migration. However, Intel VT and AMD-V processors use different architectures to facilitate migration, and moving VMs between Intel and AMD-based servers may result in unexpectedly poor migration performance.

2.1 Live Migration options for storage configurations

In addition to network settings, there are some storage connection types that must also be carefully configured on Hyper-V hosts for Live Migration to run properly. A Virtual Hard Disk (VHD) attachment, for instance, is arguably the simplest for Live Migration purposes. When VHDs are attached to a highly available VM, they must also exist on shared storage. This setup ensures that every cluster node can automatically access the disk when a VM migrates. For pass-through disks, another storage configuration, additional care is necessary. These disks have a direct relationship with both VMs and their hosts, which must be considered before performing Live Migration. A pass-through disk must be exposed to the host and then passed through to the VM. Pass-through disks are supported in a clustered configuration; but the cluster must be informed of any new pass-through disks by refreshing the VM configuration after it has been attached. Pass-through disks must be managed like other cluster resources. The storage area network connections to the cluster must be exposed to every potential cluster host.

III. Design

At a high level we can consider a virtual machine to encapsulate access to a set of physical resources. Providing live migration of these VMs in a clustered server environment leads us to focus on the physical resources used in such environments: specifically on memory, network and disk. This section summarizes the design decisions that we have made in our approach to live VM migration. We start by describing how memory and then device access is moved across a set of physical hosts and then go on to a high-level description of how a migration progresses.

3.1 Migrating Memory

Moving the contents of a VM's memory from one physical host to another can be approached in any number of ways. However, when a VM is running a live service it is important that this transfer occurs in a manner that balances the requirements of minimizing both *downtime* and *total migration time*. The former is the period during which the service is unavailable due to there being no currently executing instance of the VM; this period will be directly visible to clients of the VM as service interruption. The latter is the duration between when migration is initiated and when the original VM may be finally discarded and, hence, the source host may potentially be taken down for maintenance, upgrade or repair. It is easiest to consider the trade-offs between these requirements by generalizing memory transfer into three phases:

Push phase

The source VM continues running while certain pages are pushed across the network to the new destination. To ensure consistency, pages modified during this process must be re-sent.

Stop-and-copy phase

The source VM is stopped, pages are copied across to the destination VM, then the new VM is started.

Pull phase

The new VM executes and, if it accesses a page that has not yet been copied, this page is faulted in ("pulled") across the network from the source VM.

Although one can imagine a scheme incorporating all three phases, most practical solutions select one or two of the three. For example, *pure stop-and-copy* [10] involves halting the original VM, copying all pages to the destination, and then starting the new VM. This has advantages in terms of simplicity but means that both downtime and total migration time are proportional to the amount of physical memory allocated to the VM. This can lead to an unacceptable outage if the VM is running a live service.

Another option is *pure demand-migration* [11] in which a short stop-and-copy phase transfers essential kernel data structures to the destination. The destination VM is then started, and other pages are transferred across the network on first use. This results in a much shorter downtime, but produces a much longer total migration time; and in practice, performance after migration is likely to be unacceptably degraded until a considerable set of pages have been faulted across. Until this time the VM will fault on a high proportion of its memory accesses, each of which initiates a synchronous transfer across the network.

The approach taken in this paper, *pre-copy* [12] migration, balances these concerns by combining a bounded iterative push phase with a typically very short stop-and-copy phase. By `iterative' we mean that pre-copying occurs in *rounds*, in which the pages to be transferred during round n are those that are modified during round n-1 (all pages are transferred in the first round). Every VM will have some (hopefully small) set of pages that it updates very frequently and which are therefore poor candidates for pre-copy migration. Hence we bound the number of rounds of pre-copying, based on our analysis of the *writable working set* (WWS) behaviour of typical server workloads. Finally, a crucial additional concern for live migration is the impact on active services. For instance, iteratively scanning and sending a VM's memory image between two hosts in a cluster could easily consume the entire bandwidth available between them and hence starve the active services of resources. This *service degradation* will occur to some extent during any live migration scheme. We address this issue by carefully controlling the network and CPU resources used by the migration process; thereby ensuring that it does not interfere excessively with active traffic or processing.

3.2 Resources for Migration

A key challenge in managing the migration of OS instances is what to do about resources that are associated with the physical machine that they are migrating away from. While memory can be copied directly to the new host, connections to local devices such as disks and network interfaces demand additional consideration. The two key problems that we have encountered in this space concern what to do with network resources and local storage.

For network resources, we want a migrated OS to maintain all open network connections without relying on forwarding mechanisms on the original host (which may be shut down following migration), or on support from mobility or redirection mechanisms that are not already present (as in [13]). A migrating VM will include all protocol state (e.g. TCP PCBs), and will carry its IP address with it. To address these requirements we observed that in a cluster environment, the network interfaces of the source and destination machines typically exist on a single switched LAN. Our solution for managing migration with respect to network in this environment is to generate an unsolicited ARP reply from the migrated host, advertising that the IP has moved to a new location. This will reconfigure peers to send packets to the new physical address, and while a very small number of in-flight packets may be lost, the migrated domain will be able to continue using open connections with almost no observable interference.

Some routers are configured not to accept broadcast ARP replies (in order to prevent IP spoofing), so an unsolicited ARP may not work in all scenarios. If the operating system is aware of the migration, it can opt to send directed replies only to interfaces listed in its own ARP cache, to remove the need for a broadcast. Alternatively, on a switched network, the migrating OS can keep its original Ethernet MAC address, relying on the network switch to detect its move to a new port. In the cluster, the migration of storage may be similarly addressed: Most modern data centers consolidate their storage requirements using a network-attached storage (NAS) device, in preference to using local disks in individual servers. NAS has many advantages in this environment, including simple centralised administration, widespread vendor support, and reliance on fewer spindles leading to a reduced failure rate. A further advantage for migration is that it obviates the need to migrate disk storage, as the NAS is uniformly accessible from all host machines in the cluster

3.3 Pre-copy Migration

Pre-copy migration tries to tackle problems associated with earlier designs by combining a bounded iterative push step with a final and typically very short stop-and-copy[14] phase. The core idea of this design is that of iterative convergence. The design involves iterating through multiple rounds of copying in which the VM memory pages that have been modified since the previous copy are resent to the destination on the assumption that at some point the number of modified pages will be small enough to halt the VM temporarily, copy the (small number of) remaining pages across, and restart it on the destination host. Such a design minimises both total migration time and downtime.

3.3.1 Stages in Pre-copy Migration

Pre-copy migration involves 6 stages, namely:

1) **Initialisation**: a target is pre-selected for future migration.
2) **Reservation**: resources at the destination host are reserved.
3) **Iterative pre-copy**: pages modified during the previous iteration are transferred to the destination. The entire RAM is sent in the first iteration.
4) **Stop-and-copy**: the VM is halted for a final transfer round.
5) **Commitment:** the destination host indicates that it has received successfully a consistent copy of the VM.
6) **Activation:** resources are re-attached to the VM on the destination host.

Unless there are stop conditions, the iterative pre-copy stage may continue indefinitely. Thus, the definition of stop conditions is critical in terminating this stage in a timely manner. These conditions are usually highly dependent on the design of both the hypervisor and the live migration sub-system but are generally defined to minimise link usage and the amount of data copied between physical hosts while minimising VM downtime. However, the existence of these stop conditions has a significant effect on migration performance and may cause non-linear trends in the total migration time and downtime experienced by VMs.

3.3.2. Defining Migration Performance

Migration performance may be evaluated by measuring total migration time and total downtime. The former is the period when state on both machines is synchronised, which may affect reliability while the latter is the duration in which the VM is suspended thus seen by clients as service outage. Using the pre-copy migration model, total migration time may be defined as the sum of the time spent on all 6 migration stages (Equation 1) from initialisation at the source host through to activation at the destination. Total downtime, however, is the time required for the final 3 stages to complete (Equation 2). While it is expected that the iterative pre-copy stage will dominate total migration time, our measurements found that for certain classes of applications specifically those that do not have a high memory page modification rate the initialisation, reservation, commitment and activation stages may add a significant overhead to total migration time and downtime. We classify the initialisation and reservation stages together as pre-migration overhead while the commitment and activation stages compose post-migration overhead.

$$\text{TotalMigrationTime} = \underbrace{\text{Initialisation} + \text{Reservation}}_{\text{Pre-migrationOverhead}}$$

$$+\sum \text{pre-copy} + \text{stop-and-copy}$$

$$\underbrace{+ \text{Commitment} + \text{Activation}}_{\text{Post-migrationOverhead}} \longrightarrow \text{Equation (1)}$$

$$\text{TotalDowntime} = \text{Stop-and-copy}$$

$$\underbrace{+ \text{Commitment} + \text{Activation}}_{\text{Post-migrationOverhead}}$$

$$\longrightarrow \text{Equation (2)}$$

3.3.3. Migration Bounds

Given the stop conditions, it is possible to work out the upper and lower migration performance bounds for a specific migration algorithm. We will use a real-world case to characterise these boundaries. While there exist a range of live migration platforms, for the remainder of this paper we will base our analysis on the Xen migration platform. Xen is already being used as the basis for large scale cloud deployments [15] and thus this work would immediately benefit these deployments. Moreover, Xen is open-source allowing us to quickly and efficiently determine the migration sub-system design and implementation. Note however that our measurement techniques, methodology, and prediction models design basis are applicable to any virtualisation platform that employs the pre-copy migration mechanism. The stop conditions that are used in Xen migration algorithm are defined 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 guarantees a short downtime as few pages are to be transferred. On the other hand, the other 2 conditions just force migration into the stop-and-copy stage which might still have many modified pages to be copied across resulting in large downtime.

1) Bounding Total Migration Time (Equation 3): Consider the case of an idle VM running no applications. In this case the iterative pre-copy stage will terminate after the first iteration as there is no memory difference. Consequently, the migration sub-system needs only to send the entire RAM in the first round. The total migration lower bound is thus the time required to send the entire RAM coupled with pre- and post-migration overheads. On the other hand, consider the case where the entire memory pages are being modified as fast as link speed. In this scenario, the iterative pre-copy stage will be forced to terminate after copying more than 3 times the total amount of RAM allocated to the VM. Migration then re-sends the entire modified RAM during the stop-and-copy stage. The total migration upper bound is thus defined as the time required sending 5 times the VM size less 1 page1 plus pre- and postmigration overheads.

$$\text{Overheads} + (\text{VMSize} / \text{LinkSpeed}) <= \text{TotalMigrationTime} <= (\text{Overheads} + ((5 * \text{VMSize} - 1) * \text{page}) / \text{LinkSpeed})$$

$$\longrightarrow \text{Equation (3)}$$

2) Bounding Total Downtime (Equation 4): Similarly, the total downtime lower bound is defined as the time required for the post-migration overhead, assuming that the final stop and copy stage does not transfer any pages. This occurs either when the VM is idle or the link speed is fast enough to copy all dirtied pages in the pre-copy stage. On the other hand, the total downtime upper bound is defined as the time required to copy the entire RAM in the stop-and-copy stage coupled with the post-migration overhead.

$$\text{Post-migrationOverhead} <= \text{TotalDowntime} <= (\text{Post-migrationOverhead} + (\text{VMSize} / \text{LinkSpeed}))$$

$$\longrightarrow \text{Equation (4)}$$

3.3.4 Difference in Bounds

Modelling bounds is useful as it enables us to reason about migration times provided that we know the link speed and VM memory size. These bounds are the limits in which the total migration time and total downtime are guaranteed to lie. Given a 1,024 MB VM and 1 Gbps migration link, for example, the total migration time has a lower bound of 13 and upper bound of 50 seconds respectively. Similarly, the downtime has a lower bound of .314 and upper bound of 9.497 seconds respectively. Table I illustrates the migration bounds for some common link speeds. While the downtime lower limit is fixed (as it is dependent purely on post-migration overhead) all other bounds vary in accordance to link speed due to their correlation with the VM memory size. As the table indicates, the bounds vary significantly. For bigger VM memory sizes (which is common in current installations [16]) we have even larger differences. Thus, using bounds is at best an imprecise exercise and does not allow for accurate prediction of migration times. Building better predictions requires understanding the relationship between factors that impact migration performance.

Table I: Migrationbounds.Mt:Total Migration Time (Seconds). Dt: Total Downtime (Milliseconds). Lb: Lower Bound. Ub: Upper Bound. Vm Size= 1,024 Mb.

Speed	MTLB	MTUB	DTLB	DTUB
100 Mbps	92.2 s	437.1 s	311 ms	90,466.5 ms
1 Gbps	11.7 s	43.95 s	311 ms	9,347.3 ms

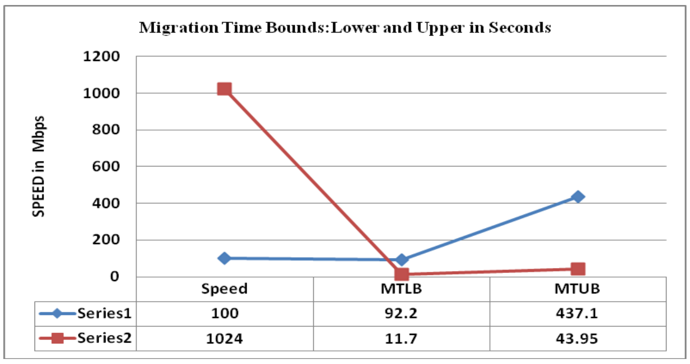
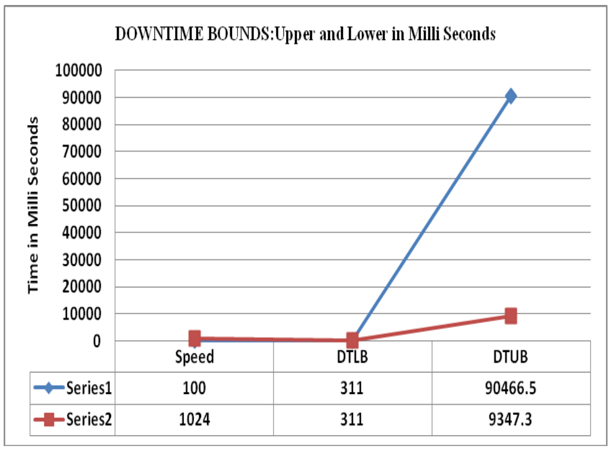


Fig.1 Migration Time Lower Bound and Upper Bound in Sec



	Speed	DTLB	DTUB
Series1	100	311	90466.5
Series2	1024	311	9347.3

Fig.2 Down Time Lower Bound and Upper Bound in Sec

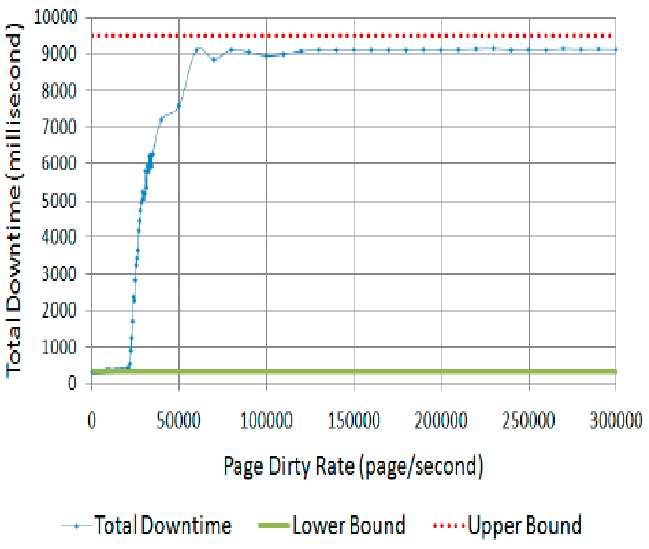


Fig.5 1 Gbps Total Down Time

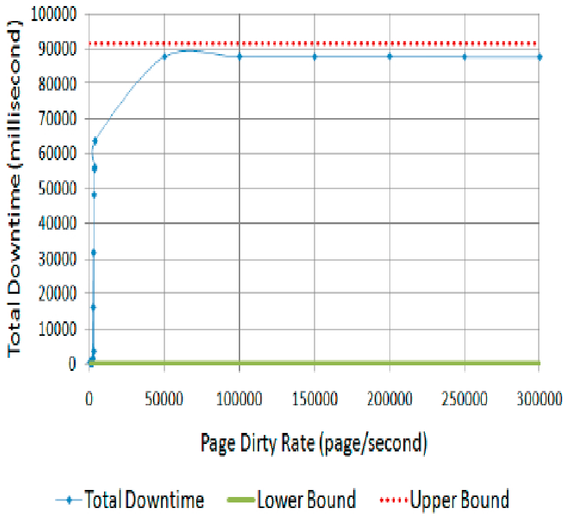


Fig.3 100 Mbps Total Down Time

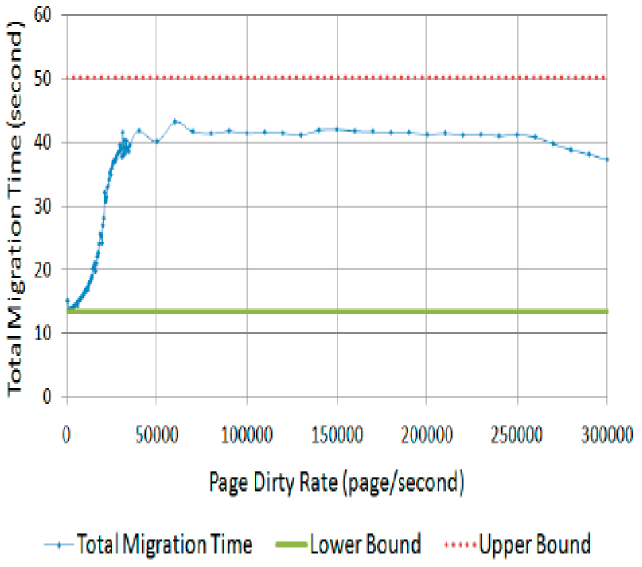


Fig.6 1 Gbps Total Migration Time

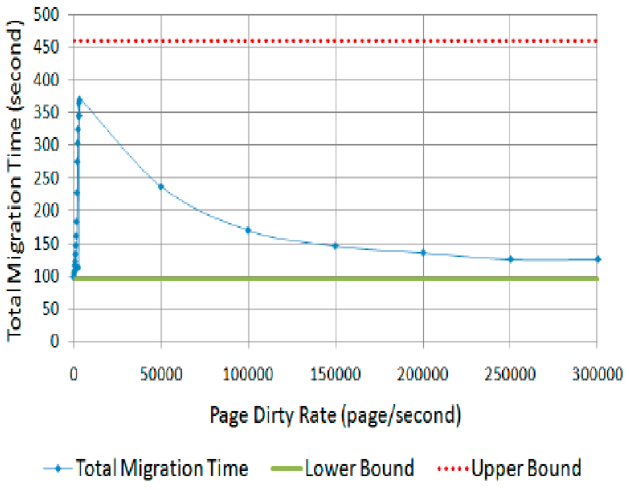


Fig. 4 100 Mbps Total Migration Time

IV. PARAMETERS AFFECTING MIGRATION

There are several factors that we need to study as a prerequisite for accurate migration modelling. In this section, we explore these factors and their impact on total migration time and downtime. Moreover, stop conditions that may force migration to reach its final stages are generally what governs migration performance. Obviously, this is implementation specific which is exemplified by but not limited to Xen support for live migration. Migration link bandwidth is perhaps the most influential parameter on migration performance. Link capacity is inversely proportional to total migration time and downtime. Higher speed links allow faster transfers and thus require less time to complete. Figure 1 illustrates migration performance for a 1,024 MB VM running a micro-benchmark that writes to memory pages with rates up to 300,000 pages/second on100 Mbps, 1 Gbps links. It represents the impact of each link speed on total migration time and downtime.

As link bandwidth increases, the point in the curve when migration performance starts to degrade rapidly shifts to the right roughly with the same ratio. Page dirty rate is the rate at which memory pages in the VM are modified which, in turn, directly affects the number of pages that are transferred in each pre-copy iteration. Higher page dirty rates result in more data being sent per iteration which leads to longer total migration time. Furthermore, higher page dirty rates results in longer VM downtime as more pages need to be sent in the final transfer round in which the VM is suspended.

Figure 1 shows the effect of varying the page dirty rate on total migration time and downtime for each link speed. The relationship between page the dirty rate and migration performance is not linear because of the stop conditions. If the page dirty rate is below link capacity, the migration sub-system is able to transfer all modified pages in a timely fashion, resulting in a low total migration time and downtime. On the other hand, if the page dirty rate starts approaching link capacity, migration performance degrades significantly. Total downtime at low page dirty rates is virtually constant and approximately equal to the lower bound (Equation 4). This is because the link has enough capacity to transfer dirty pages in successive iterations leading to a very short stop-and-copy stage. When the page dirty rate increases to the point that 29 iterations are not sufficient to ensure a short final copy round or when more than 3x the VM size have been transferred, migration is forced to enter its final stage with a large number of dirty pages yet to be sent.

Consequently, total downtime starts to increase in proportion to the increase in the number of modified pages that need to be transferred in the stop-and copy stage. Total downtime further increases until the defined upper bound in which it has to send the entire VM memory. Total migration time also increases with an increasing page dirty rate. This is attributable to the fact that more modified pages have to be sent in each pre-copy round. Moreover, the migration sub-system has to go through more iteration with the hope to have a short final stop-and-copy round. For page dirty rates near link speed, total migration time approaches its upper bound (Equation 3) as migration stops when 3x VM size has been transferred. Then, it starts to fall back towards its lower bound.

For extremely high page dirty rates (compared to link speed), migration is forced to reach its final transfer stage after 29 iterations having sent virtually no pages.2 It then has to transfer the entire RAM in the final iteration. This is exemplified clearly for the 100 Mbps link in Figure 4, in which the total migration time drops back to its lower bound (almost all dirty pages are skipped in every iteration except the final one) while having a total downtime (Figure 3) at its upper bound (as the entire RAM has to be transferred in the stop-and-copy stage).

The first pre-copy iteration tries to copy across the entire VM allocated memory. The duration of this first iteration is thus directly proportional to the VM memory size and subsequently impacts total migration time. On average, total migration time increases linearly with VM size. On the other hand, the total downtime for low page dirty rates is almost the same regardless of the VM size as the migration sub-system succeeds in copying all dirtied pages between successive iterations resulting in a short stop-and-copy stage. When the link is unable to keep up with the page dirty rate, larger VMs suffer longer downtime (linearly proportional to the VM size) as there are more distinct physical pages that require copying in the stop-and-copy stage.

Pre- and post-migration overheads refer to operations that are not part of the actual transfer process. These are operations related to initialising a container on the destination host, mirroring block devices, maintaining free resources, reattaching device drivers to the new VM, and advertising moved IP addresses. As these overheads are static, they are significant especially with higher link speeds. For instance, pre-migration setup constitutes around 77% of total migration time on a 1Gbps link for a 512 MB idle VM. More importantly, post-migration overhead is an order of magnitude larger than the time required for the stop-and-copy stage. To conclude this section, there are several parameters affecting migration performance. These parameters may be classified as having either a static or dynamic effect on migration performance. Parameters having static effects are considered as unavoidable migration overheads. On the other hand, parameters having dynamic effects on migration affect only the transfer process. Dynamic parameters are typically related to the VM specification and applications hosted inside it.

We show that the page dirty rate and link speed are the major factors influencing migration times. We also show how particular combinations of these factors can extend expected total migration time and downtime. Finally, we observe that the pre- and post-migration overheads become significant compared to the iterative pre-copy and stop-and-copy stages, especially for VMs that have low page dirty rates and are being migrated over high speed links.

V. CONCLUSION

In this paper, we studied live migration behaviour in precopy migration architectures, specifically using the Xen virtualisation platform. We show that the link speed and page dirty rate are the major factors impacting migration behaviour. These factors have a non-linear effect on migration performance largely because of the hard stop conditions that force migration to its final stop-and-copy stage. In a virtualised environment, administrators can dynamically change VM placements in order to plan maintenance, balance loads, or save energy. Live migration is the tool used.

Future Scope

The experiments that we have carried out prove that the migration link speed is the most influential parameter on performance. We have been working on local area networks assuming live migration inside one datacentre. However, moving workloads between different data centres, especially for cloud providers, is also useful. We plan to further utilise the models to study migration behaviour on wide area networks.

REFERENCES

1. Casas, J., Clark, D. L., Conuru, R., Otto, S. W., Prouty, R. M., and Walpole, J. (Spring 1995). MPVM: A Migration Transparent Version of PVM. *Computing Systems*, 8(2):171–216.
2. Rosenblum M., Garfinkel T. Virtual Machine Monitors: Current Technology and Future Trends, Computer, vol. 38, no. 5, pp. 39-47, May, 2005.
3. P. Barham et al., Xen and the Art of Virtualization.
4. E. Bugnion et al., “Disco: Running Commodity Operating Systems on Scalable Multiprocessors,”*ACM Trans. Computer Systems*, vol. 15, no. 4, 1997, pp. 412-447.
5. M. Kozuch et al., Internet Suspend Resume, WMCSA, 2002.
6. POPEK, G. J., GOLDBERG, R. P Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM* 17 (7): 412 –421.
7. D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. *ACM Computing Surveys*, 32(3):241.299, 2000.
8. C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M.Rosenblum. Optimizing the migration of virtual computers. In *Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI-02)*, December 2002.
9. Live Migration with AMD-V™ Extended Migration Technology,AMD white paper,2008,www.amd.com.
10. Andrew Whitaker, Richard S. Cox, Marianne Shaw, and Steven D. Gribble. Constructing services with interposable virtual hardware. In *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI '04)*, 2004.
11. E. Zayas. Attacking the process migration bottleneck. In *Proceedings of the eleventh ACM Symposium on Operating systems principles*, pages 13.24. ACM Press, 1987.
12. Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable remote execution facilities for the V-system. In *Proceedings of the tenth ACM Symposium on Operating System Principles*, pages 2.12. ACM Press, 1985.
13. S. Osman, D. Subhraveti, G. Su, and J. Nieh. The design and implementation of zap: A system for migrating computing environments. In *Proc. 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI-02)*, pages 361.376, December 2002.
14. C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, “Live migration of virtual machines,” in Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI’05), Berkeley, CA, USA, 2005, pp. 273–286.
15. Amazon Elastic Compute Cloud (Amazon EC2). Amazon Web Services LLC. [Online]. Available: http://aws.amazon.com/ec2/
16. S. Hacking and B. Hudzia, “Improving the live migration process of large enterprise applications,” in Proc. ACM International Workshop on Virtualization Technologies in Distributed Computing (VTDC’09), New York, NY, USA, 2009, pp. 51–58.