Optimized Pre-Copy Live Virtual Machine Migration for Memory-Intensive Workloads

MSc Research Project
Cloud Computing

Prateek Jain
Student ID: X19189851

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Prateek Jain
Student ID:	X19189851
Programme:	Cloud Computing
Year:	2021
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	16/08/2021
Project Title:	Optimized Pre-Copy Live Virtual Machine Migration for Memory-Intensive Workloads
Word Count:	3813
Page Count:	17

Signature:	
Date:	16th August 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimized Pre-Copy Live Virtual Machine Migration for Memory-Intensive Workloads

Prateek Jain
X19189851

Abstract

With the expansion of IT infrastructures and rising energy and power costs, maintaining workload concentration and the high availability of virtual machines (VMs) in a data center is becoming increasingly difficult. To mention a few issues, some physical servers may be overloaded, while others may be idle. If a server fails, all VMs on it are affected; and so on. To guarantee service continuity, these impacted VMs must be moved to other servers. These issues (how to equally allocate jobs among servers, how to safeguard VMs from equipment failures, and so on) are being addressed in tandem with the introduction of a crucial technology—VM migration. Pre-copy migration is a popular method for transferring VMs across physical servers. Before terminating the VM on the source, pre-copy moves the VM's memory state from source to destination in successive iterations. Although the pre-copy approach reduces downtime and total migration time, it does limit the number of copying rounds. Because the writable working set is not guaranteed to converge across consecutive cycles, especially when the VM is running a predominantly write-intensive application, an alternative and more precise method are necessary to manage memory-intensive workloads. The improved migration method is provided in this work to address the limitations of the pre-copy migration approach. The technique works in conjunction with KVM's default migration mechanism. Memcached, a key-value store application is used to assess the performance of the improved VM migration mechanism. Oracle VirtualBox is used as a test environment to carry out the migration procedure.

1 Introduction

Cloud computing has set a benchmark in recent years for the on-demand supply of computer power, storage, and software applications. Cloud computing is based on a technique known as virtualization. By installing a layer (VMM or hypervisor) on top of actual hardware resources, virtualization creates separate execution environments for virtual machines (VM) to perform their services or applications. One of the primary benefits of virtualization is the ability to migrate virtual machines in real-time. The process of migrating a virtual machine (VM) with its applications and operating system to a new location without disrupting or restarting the VM is known as live virtual machine migration Barham et al. (2003).

One of the prominent enabling technologies for fault tolerance and load balancing is live virtual machine migration. With live migration, whole environments, including Virtual Machines (VMs), operating systems, and performing workloads, are transferred across separate physical nodes or Virtual Machine Monitors without interruption to any other underline services.

1.1 Importance of Live Migration

It is critical to comprehend and examine the idea of live migration since it streamlines many cloud management activities performed by cloud providers. With the growing use of cloud computing environments for hosting a range of applications such as Web, Virtual Reality, scientific computing, and big data, the necessity for offering cloud services with Quality of Service (QoS) assurances is becoming important He et al. (2019). To avoid SLA violations and achieve QoS requirements, VM placement must be continually adjusted in data centers. As a dynamic management tool, live VM migration supports various resource scheduling objectives such as load balancing, cloud bursting, resource overbooking, and energy-saving strategy, fault tolerance, scheduled maintenance, and evacuating VMs to other data centers before incidents such as earthquakes, flooding, etc.

In data centers, VM migration is critical for hardware maintenance. Many virtual machines must be switched from one physical server to another due to excessive load, power grid breakdown, or scheduled maintenance activities in a data center. With numerous businesses and customers migrating to cloud data centers, cloud service providers frequently encounter situations of servers overloading while others are inactive; these conditions hinder data center operation. Cloud service providers equally distribute load across physical resources with the aid of VM migration via LAN or through WAN, therefore enhancing the life cycle and productivity of resources. VMs may also be moved to underutilized servers to optimize power usage and save electricity. Microsoft and Amazon utilize the sun and moon idea to migrate VMs to data centers based on their geographical location to deliver low latency to consumers or to decrease cooling costs Zhang et al. (2018).

1.2 Motivation

After extensively reviewing the existing work on live VM migration, it is clear that existing pre-copy migration techniques attempt to impose static stopping limitations such as a maximum number of iterations or a maximum total number of bytes transferred to avoid the iterative transfer phase continuing endlessly while migrating a VM with a high memory write rate. As a result, the research proposal seeks to incorporate adaptive stopping conditions into the pre-copy migration approach, this will bring the iteration phase to a close and convert the migration process from iteration to stop-and-copy, thereby decreasing total migration time and total migrated data.

1.3 Research Question

Can adaptive stopping conditions and hashing algorithms reduce total migration time and total data transferred of VM running memory-intensive workloads?

1.4 Objectives

This project is carried out to enhance the performance of VM migration and increase the working capacity of data centers by achieving the below objectives:

- Setting up a test environment for VM migration experiments using Oracle VirtualBox.
- Configure the necessary infrastructure and security configurations for a successful VM migration.
- Selecting a memory-intensive workload to achieve significant I/O, CPU, and memory overhead during VM migration experiments, just like in a real data center.
- The proposed algorithm's performance evaluation with the KVM's default VM migration.

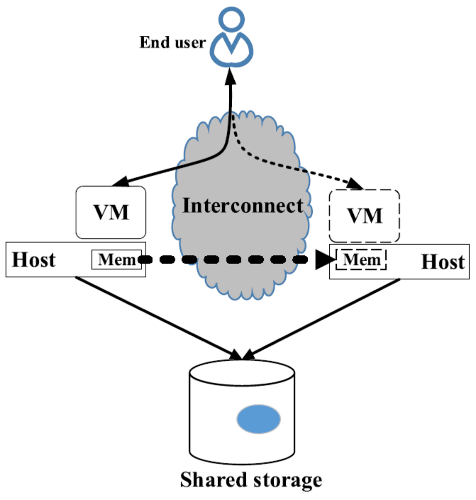


Figure 1: Live VM Migration

2 Related Work

This section describes the high-level overview of the live VM migration and previous research studies in the same area. This section is divided into subsections as following, 2.1 describes the working of the live migration. Further, 2.2 states the several techniques of live migration. 2.3 reviews the previous research work and critically analyzed the methods of live VM migration. In the end, 2.4 mentions the evaluation metrics used to analyze the performance of VM migration.

2.1 The Anatomy of Live Migration

Migrating a VM from one physical host to another necessitates transferring the VM's full volatile state. A virtual machine's volatile state comprises its memory contents, execution context, and device buffers. In most of the cases VM's memory size ranges from four to tens of gigabytes or in some cases up to terabytes, primarily memory transfer dominates the migration overhead Egger et al. (2015).

Live migration can be divided into three distinct phases: prepare, stop-and-copy, and resume. During the prepare phase, a portion of the VM's memory is moved to the destination host while the VM remains on the source. It is conceivable that the

VM's performance will suffer as a result of the migration technique's increased resource usage. The preparation phase is followed by the stop-and-copy phase, in which the VM is entirely stopped and control of the execution is passed from the source to the destination host. Finally, the VM is resumed on the destination during the resume phase, and the remaining volatile state is retrieved from the source host. In the preparation step, certain migration algorithms do not copy the whole volatile state. This might result in significant performance reduction during the restart phase.

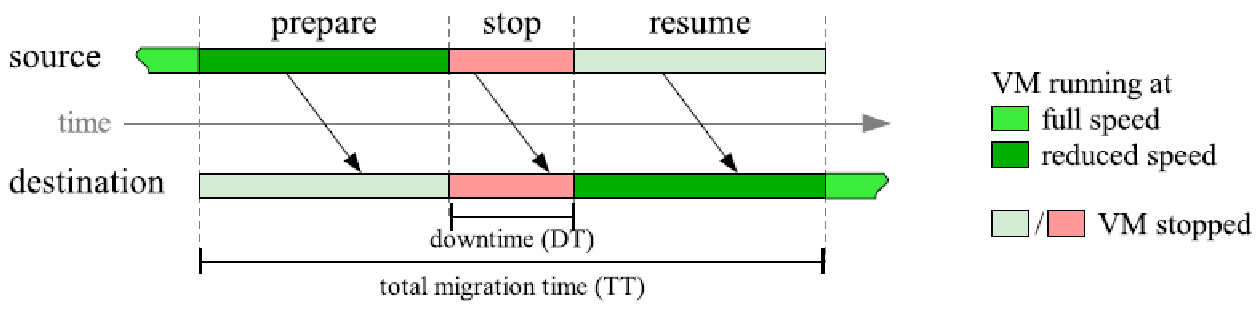


Figure 2: Live VM Migration Phases

2.2 Live Migration Techniques

The moment in time at which a VM's memory is moved distinguishes live migration approaches. 'Pre-Copy' is the most frequently used and default approach for migrating VMs in the most popular hypervisors, including XEN, KVM, and VMware Choudhary et al. (2017). Pre-copy transfers the VM's memory state from source to destination in several rounds before stopping the VM on the source. It first copies all of the memory pages to the destination; however, as the VM is still running on the source, some of the memory pages were written again and became dirty during this process, necessitating a recopy; pre-copy iteratively copies the recently dirtied memory pages from source to destination until some predefined condition is not met. The source VM is then terminated, and all leftover pages, including CPU states, are transferred to the destination. Finally, the destination VM is restarted, and the source copy is deleted. If the memory write rate of the VM's workload exceeds the available network bandwidth for migration and the iterations fail to converge, this iteration phase can go forever Choudhary et al. (2017). As a result, each migration mechanism imposes a distinct set of preset constraints to converge the iteration process. For example, before terminating the VM on the source server, XEN replicates the memory pages until the iteration count reaches 30.

Post-copy suspends the VM on the source host first, then just the processor state is transferred to the destination. Finally, the VM is started on the target host. When the VM on the destination host attempts to read a page that was not previously copied, a page fault occurs. As a result, this faulty page is retrieved from the source. On-demand fetch, pre-fetch, pull fetch, push fetch, and other page fetching techniques are used to retrieve and transmit the remaining pages to the destination. All of the approaches listed above have advantages and disadvantages. Fig 3 represents the migration techniques.

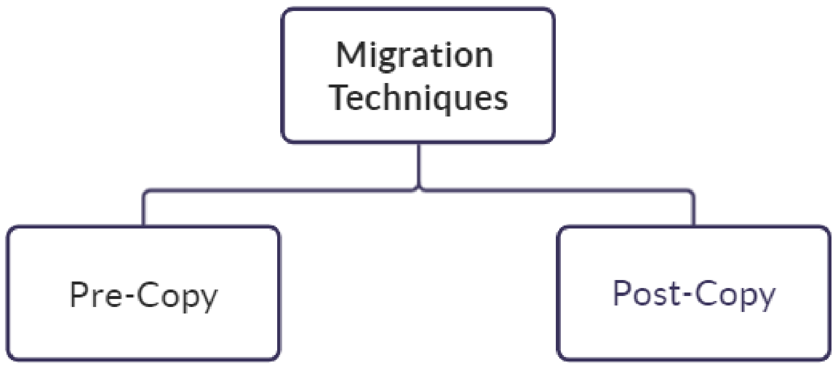


Figure 3: VM Migration Techniques

2.3 Survey

As discussed in section 2.2, the high dirtying rate of memory pages exceeds the available network capacity for migration. Due to this, pre-copy iterations fail to converge and increase the migration time and ultimately fail the migration process causing an application or service downtime. As a result, an adaptive stopping condition for the pre-copy migration approach that can converge the iteration phase depending on the memory access pattern of different write-intensive workloads is required.

Many efforts have been presented to improve the iterative transfer phase, which we shall examine.

Transferring memory pages in many rounds is not a smart idea if the pages are regularly updated. It not only lengthens the entire transfer time, but it also consumes network bandwidth. Clark et al. (2005) discovers that certain of the memory pages are regularly affected, i.e., a writable working set (WWS), and can be avoided in the following iterations, for this Clark suggested a method for identifying dirty pages that have been changed after being moved in two consecutive cycles and retransfer them. Cho-Chin Lin et al. (2012) would only retransfer the pages if they were confirmed clean for two consecutive cycles after being updated to address this issue. Both approaches increase the chances of reaching a maximum iteration count of 30.

Hu et al. (2011) presented a method for storing the page changed rate over several iterations in a bitmap and only transfer the modified page when the bit corresponding to the particular page reaches 5. TPO, a three-phase approach for optimizing migration, was suggested by Sharma and Chawla (2016). In the first iteration, the initial phase transfers the un-updated memory pages, reducing the transmitted memory pages. The second step predicts frequently modified memory pages and avoids transferring them based on prior iterations. The RLE method is used in the third phase to compress and transmit big memory pages. suggested a strategy for categorizing memory pages into five groups: anonymous, inode, kernel, cache, and free memory pages, and only transfer the anonymous, inode, and kernel pages as they are required for kernel execution.

Ref.	Method	Workload	Reduction	Limitation
Clark et al. (2005)	Bound the number of pre-copying rounds, based on analysis of the writable working set	N/A	N/A	capped number of iterations (30)
Hu et al. (2011)	Time-series prediction	Kernel compile	70% TMT , 25% TDT	N/A
Cho-Chin Lin et al. (2012)	Second chance strategy	Synthetic	30% TDT	does not provide evaluation on real workloads
Sharma and Chawla (2016)	Reduce transferring of memory pages based on three phases.	N/A	70% TMT, 71% TDT (XEN)	0.05% space overhead and CPU processing overhead to predict frequently updated pages.
Wang et al. (2017)	Avoid transferring both free and cache memory pages.	N/A	72% TMT (KVM)	Time overhead due to introspection of pages and handling missing cache pages
THIS ONE	Adaptive stopping condition based on hashing algorithm	Memcached	reduction in TMT and TDT (to be determine)	small CPU computation overhead

Table 1: Summary of Literature Review and Research Niche. TMT: Total Migration Time; TDT: Total Data Transferred; N/A: Not Available.

2.4 Live Migration Metrics

The following table displays the seven VM live migration metrics that were measured and forecasted in this research. The downtime DT is a metric of relevance to the end-user, whereas the total migration time TT, the quantity of transferred data TD, the amount of retransferred data RD, and the CPU and memory usage caused by the live migration are metrics of interest to the data center operator (CPU and MEM). Lower is better for all measures.

Total migration time (TT)	The total amount of time it takes to complete a migration from start to finish.
Downtime (DT)	The length of the stop-and-copy phase, i.e. the time the VM is suspended.
Transferred data (TD)	Total amount transferred from source to destination host
Retransferred Data (RD)	Total amount of retransferred from the source to the destination host during multiple pre-copy iterations
CPU utilization (CPU)	Additional CPU burden on the source host as a result of migration
Memory utilization (MEM)	The amount of RAM used by the improved live migration algorithm

3 Methodology

This section discusses the methodology, dataset, and tools utilized to carry out the VM migration project.

3.1 Oracle Virtual Box

Oracle VM VirtualBox [1], originally known as Sun VirtualBox or Sun xVM VirtualBox, is an x86-based hypervisor developed by Oracle Corporation. It is cross-platform virtualization software that allows users to simultaneously run multiple operating systems on Windows, Mac OS, or on Linux operation system. VirtualBox is an example of the type-2 hypervisor and one of the best applications to run multiple VMs on the local system. It is one of the highly recommended software in the computing industry which allows professionals to study new and developing technologies by testing, deploying their applications or datasets on their local system or laptop Kuhn et al. (2015).

In this experiment, VirtualBox is installed on a personal laptop running Windows, and it is used to build Linux-based virtual machines (VMs). The setup details are provided in the table below.

Machine	**Operating System**	**vCPU**	**Memory**
Personal Laptop	Windows 10	8	8 GB
VirtualBox VM	Ubuntu LTS Version v20.04	8	4.8 GB

Table 2: Setup Details

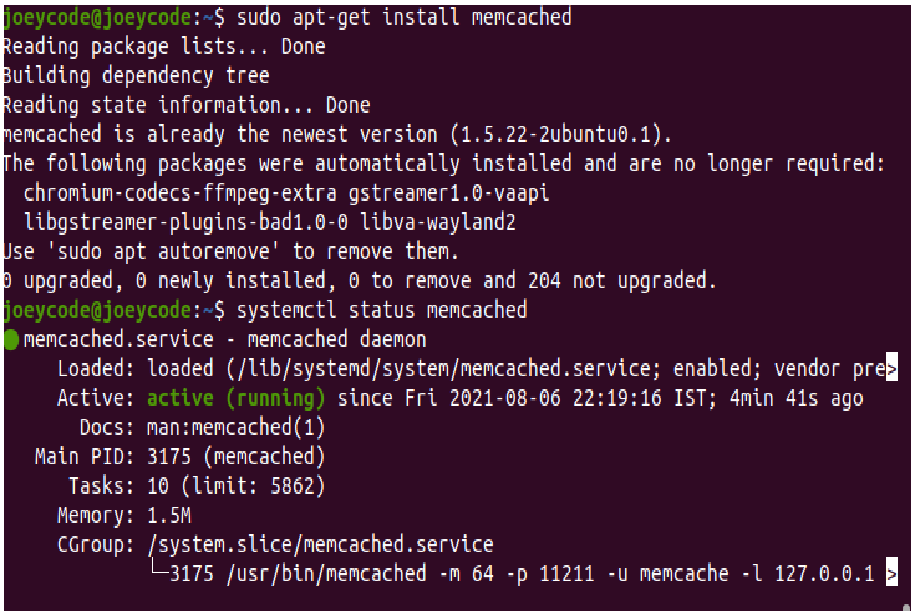
3.2 Dataset: Memcached

The Standard Performance Evaluation Corporation (SPEC) Dixit (1991) is the primary benchmarking system assessment standard. SPEC provides a variety of testing com-

[1] `https://www.virtualbox.org/`

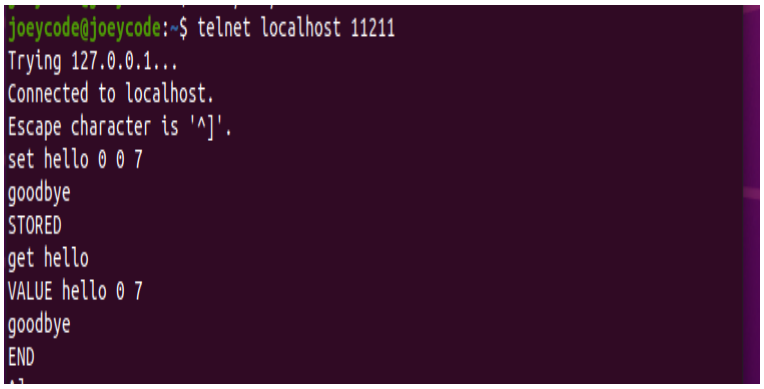
ponents that may be used to benchmark a system. In this experiment, Memcached2[2] workload is used which is a key-value distributed memory object caching system. The workload is based on SPEC CPU2006, to deliver system performance to genuine scientific and engineering activity.

Memcached is installed on the host VM running inside the virtual machine created on the VirtualBox.



```
joeycode@joeycode:~$ sudo apt-get install memcached
Reading package lists... Done
Building dependency tree
Reading state information... Done
memcached is already the newest version (1.5.22-2ubuntu0.1).
The following packages were automatically installed and are no longer required:
  chromium-codecs-ffmpeg-extra gstreamer1.0-vaapi
  libgstreamer-plugins-bad1.0-0 libva-wayland2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 204 not upgraded.
joeycode@joeycode:~$ systemctl status memcached
● memcached.service - memcached daemon
     Loaded: loaded (/lib/systemd/system/memcached.service; enabled; vendor pre>
     Active: active (running) since Fri 2021-08-06 22:19:16 IST; 4min 41s ago
       Docs: man:memcached(1)
   Main PID: 3175 (memcached)
      Tasks: 10 (limit: 5862)
     Memory: 1.5M
     CGroup: /system.slice/memcached.service
             └─3175 /usr/bin/memcached -m 64 -p 11211 -u memcache -l 127.0.0.1 >
```

Figure 4: Installation Step



```
joeycode@joeycode:~$ telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
set hello 0 0 7
goodbye
STORED
get hello
VALUE hello 0 7
goodbye
END
```

Figure 5: Working with Memcached

3.3 Migration Model

Following the start of the VM migration, all memory pages are transferred to the destination, as in the first iteration of the default pre-copy procedure. This transferred data is tracked by the algorithm and saved in a variable (TD). Then, before the second iteration, to track page retransmission, before transmitting each page, the method turns it into a safe hash using the SHA-256 algorithm [3] and save the resulting hash value in a bitmap

[2]https://memcached.org/

[3]https://csrc.nist.gov/csrc/media/publications/fips/180/2/archive/2002-08-01/documents/fips180-2withchangenotice.pdf

array. The bitmap is kept by the algorithm during the migration operation. In the following iteration, the secure hash is converted first, and then the converted hash is matched to the hashes from prior iterations; If a comparable page is identified, the related page is rejected; else, the page is retransmitted. For subsequent rounds, the secure hash is appended to the bitmap. Following each retransmission, the algorithm keeps track of the retransmitted data and saves the retransferred bytes in another variable (RD) during the iteration phase. The migration process reaches a tipping point when the retransferred data (RD) equals or exceeds 75% of the migrated data (TD).

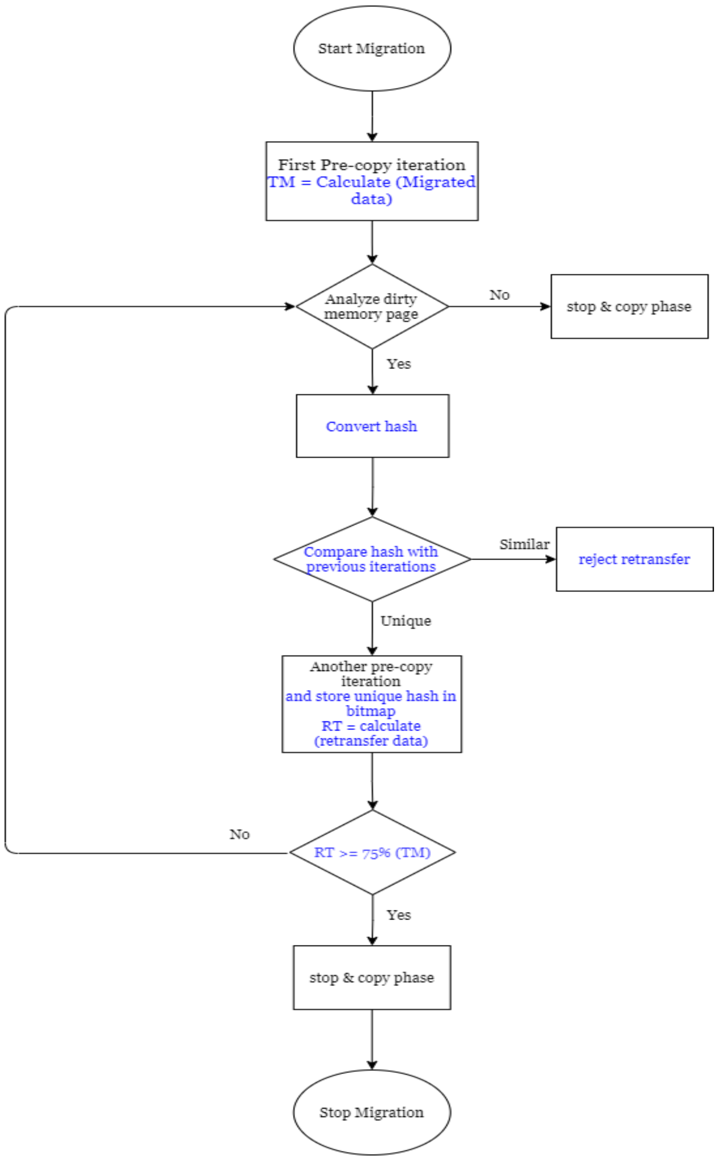

Start Migration
First Pre-copy iteration
TM = Calculate (Migrated data)
Analyze dirty memory page
No
stop & copy phase
Yes
Convert hash
Compare hash with previous iterations
Similar
reject retransfer
Unique
Another pre-copy iteration
and store unique hash in bitmap
RT = calculate (retransfer data)
No
RT >= 75% (TM)
Yes
stop & copy phase
Stop Migration

Figure 6: Workflow of Algorithm

4 Design Specification

The code for the enhanced VM migration algorithm is shown in this section. The technique is intended to decrease the overall VM migration time as well as the total transferred data across the network. The algorithm steps are as follows:

- In the first iteration, the algorithm collects the *starttime* after beginning the migration process and migrates all of the memory pages accessible on the RAM.
- The migrated data is computed and saved in a variable *TD* for subsequent usage and thread for tracking dirty page is initiated.
- Before the next iteration, the algorithm examines the state of the tracking thread to see if the pages were dirty after the first iteration.
- If dirty pages are discovered, the algorithm resets the dirty tracking thread and transforms the dirty pages into the secure hash. These secure hashes are preserved for future iterations. If no dirty pages are detected, the algorithm simply enters the stop-and-copy phase.
- The secure hash is preserved in a bitmap array, and the algorithm compares the hashes among themselves before transferring pages throughout the iterations. The algorithm rejects the retransfer of the page if the hash corresponding to that page is already present in the bitmap and only transfers the unique hash.
- The retransferred data is calculated and saved in a variable *RD*.
- Algorithm compares the two variables after every iteration and goes into stop-and-copy phase as soon as, retransfer 75% of transferred data and captures the time in *endtime* variable.
- Total Migration Time is computed by subtracting the *endtime* from *starttime* and Total Migration Data is computed by adding the *TD* and *RD*.

Algorithm 1 Pseudo Code of Enhanced Migration algorithm

```
1: procedure MIGRATION($TM$, $RT$)
2:     $starttime \leftarrow$ gettime()
3:     for all block in ram do
4:         for all page in block do
5:             migrate page to destination                    ▷ First pre-copy iteration
6:             $TM \leftarrow$bytes_sent (migrate page)
7:             set dirty_tracking ()
8:         end for
9:     end for
10: if dirty_tracking () is true do
11:     reset dirty_tracking ()
12:     $a \leftarrow$convert dirty page into hash value (SHA-256)
13:     $bitmap[] \leftarrow$ all converted hash
14:     if $a$ is present in $bitmap[]$ do                    ▷ Compare with all previous iterations
15:         do not transfer (page)
16:     else retransfer page                                   ▷ Another iteration
17:         $RT \leftarrow$ bytes_sent (retransfer page)
18:         $bitmap[] \leftarrow$ a
19:     end if
20: else switch to stop and copy phase
21: end if
22: if $RT/TM \geqslant 75\%$ do                          ▷ Stopping condition, 75% may vary based on testing
23:         switch to stop and copy phase
24:         $endtime \leftarrow$ gettime()
25:         $TotalMigrationTime \leftarrow endtime - starttime$
26:         $TotalMigratedData \leftarrow TM + RT$
27:     else goto line 11
28:     end if
29: end procedure
```

Figure 7: Algorithm for Enhanced VM Migration

5 Implementation

In this section, the implementation scenario of the experiment is discussed. The experiment is executed on the Oracle VirtualBox, a virtual machine running Linux operating system is deployed on the VirtualBox.

The experiment is carried out with KVM, an open-source virtualization tool, version 5.4. In the experiment, the virtual machine is configured with 2GB memory and 5 processors. The system used to install VirtualBox is running a windows 10 operating system with 8 CPUs and 8GB memory.

Host machine running ubuntu v20.04.02 operating system is configured with 8 processors, 4.8GM of memory, and has 545 GB of disk capacity. On the other hand, the client machine has 8 processors, 8GB of memory, 512 GB of disk capacity, and is running ubuntu v20.04.02. The VM is transferred from the host system to the client system to fulfill the live migration. The VM is transferred from host to client using the NFS (Network File System) protocol. It is a software application that allows users to view, store and access files on the remote server. NFS protocol is the Network Attached Storage standard which provides data access to different clients connected in a network.

As a workload, in-memory key-value store Memcached, is used to evaluate the performance of the migration algorithm.

5.1 Integration with KVM

The enhanced migration code is integrated with the 'migration.c' file in the migration module of the QEMU-KVM hypervisor. Once the hypervisor triggers the migration process, code captures the start time of the process and during the first iteration calculates and saves the migrated data in variable *TD*. Over the next iterations, as the dirty pages get migrated over to the client, code converts each migrated page into a secure hash using SHA-256 and saves in an array. Code compares hashes and rejects the migration of the corresponding page if the hash is already present in the array. Only the unique pages are transferred again. Code calculates and stores the retransferred data in *RD* variable. These variables are used to calculate the total data being migrated. After the successful migration code checks and remembers the end time to calculate the total migration time.

5.2 Migration requirements

This section provides the basic requirements before starting the VM migration:

- On shared networked storage install guest using one of the following protocols:
  - Fibre Channel
  - iSCSI
  - NFS
  - GFS2
- Two systems with the same Linux version, updates, and with appropriate ports open are required.
- Network configuration and all bridging configuration of both the system should be identical.
- On both the source and destination computers shared storage must be mounted in the same place. The names of the mounted directories must be similar.

5.3 Migration Steps

The below steps provide details about the execution of VM migration in QEMU-KVM:

- A secure SSH link between both host and client-server is established.
- Start the VM on the host machine and configure the QEMU-KVM hypervisor to migrate VM to the client machine.
- Start the Memcached service on the VM.

- Migrate all the memory pages from host to client in the first iteration, convert hash of the migrated pages, and store in a bitmap array.
- Calculate and store the amount of data migrated from host to client in the first iteration.
- Retransfer the dirty pages in the subsequent iteration, calculate and store the retransfer data.
- Hypervisor triggers and switch the process from iteration phase to stop-and-copy phase once retransfer data is equal or greater than 75% of the data transferred in the first iteration.
- Next the VM is terminated on the host machine.
- Finally, all the activities of VM resume on the client machine, and migration is completed.

6 Evaluation and Discussion

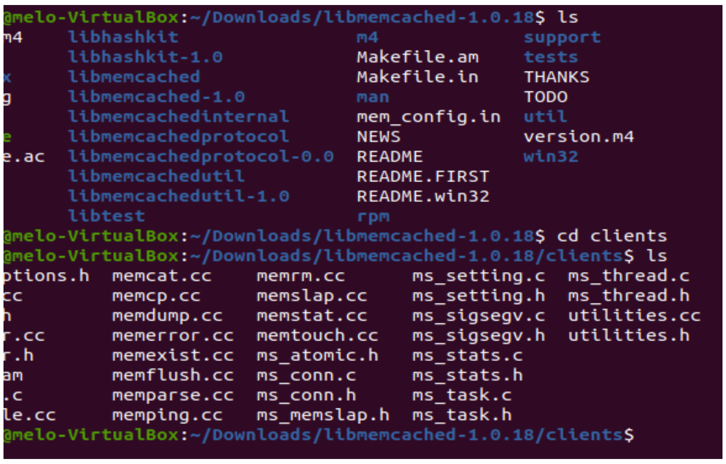
The goal of this part is to offer a detailed analysis of the experiment's outcomes and major findings. This section is divided into subsections as following, section 6.1 describe the tool used for critically analyze the performance of memcached application during VM migration. Further, section 6.2 states the steps involved during the installation of memaslap client application. Section 6.3 provides the insights of memcached server performance during migration with KVM's default migration method. In the end, section 6.4 discussed about the obtained results

6.1 Testing Tool: Memaslap

An open-source program called memaslap[4] is used to test the performance of memcached during the VM migration. It is a component of the libmemcached toolkit[5] that may produce customizable loads such as threads, concurrencies, connections, run duration, overwrite, miss rate, key size, value size, get/set percentage, and so on to benchmark the memcached server.

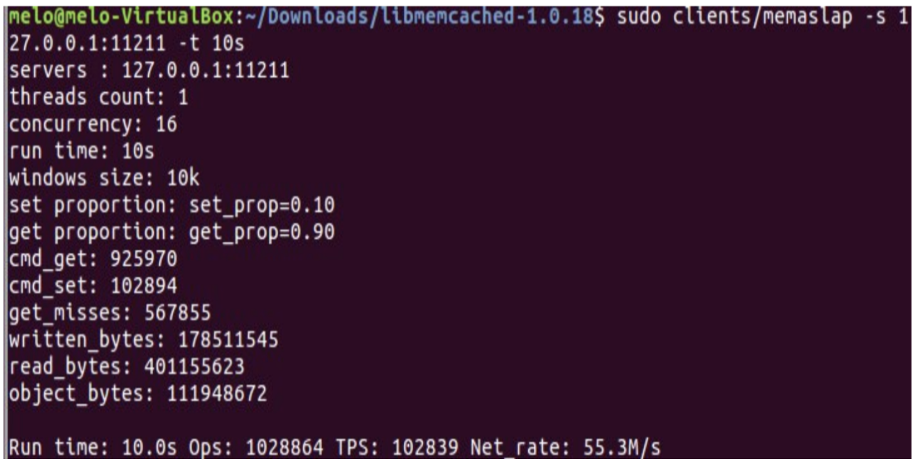
[4]`http://docs.libmemcached.org/bin/memaslap.html`
[5]`http://libmemcached.org/lib/Memcached.html.`

6.2 Installation: Memaslap



```
@melo-VirtualBox:~/Downloads/libmemcached-1.0.18$ ls
m4    libhashkit                   m4              support
      libhashkit-1.0               Makefile.am     tests
x     libmemcached                 Makefile.in     THANKS
g     libmemcached-1.0             man             TODO
      libmemcachedinternal         mem_config.in   util
e     libmemcachedprotocol         NEWS            version.m4
e.ac  libmemcachedprotocol-0.0     README          win32
      libmemcachedutil             README.FIRST
      libmemcachedutil-1.0         README.win32
      libtest                      rpm
@melo-VirtualBox:~/Downloads/libmemcached-1.0.18$ cd clients
@melo-VirtualBox:~/Downloads/libmemcached-1.0.18/clients$ ls
ptions.h  memcat.cc    memrm.cc      ms_setting.c  ms_thread.c
cc        memcp.cc     memslap.cc    ms_setting.h  ms_thread.h
h         memdump.cc   memstat.cc    ms_sigsegv.c  utilities.cc
r.cc      memerror.cc  memtouch.cc   ms_sigsegv.h  utilities.h
r.h       memexist.cc  ms_atomic.h   ms_stats.c
am        memflush.cc  ms_conn.c     ms_stats.h
.c        memparse.cc  ms_conn.h     ms_task.c
le.cc     memping.cc   ms_memslap.h  ms_task.h
@melo-VirtualBox:~/Downloads/libmemcached-1.0.18/clients$
```

Figure 8: Memaslap file directory



```
melo@melo-VirtualBox:~/Downloads/libmemcached-1.0.18$ sudo clients/memaslap -s 1
27.0.0.1:11211 -t 10s
servers : 127.0.0.1:11211
threads count: 1
concurrency: 16
run time: 10s
windows size: 10k
set proportion: set_prop=0.10
get proportion: get_prop=0.90
cmd_get: 925970
cmd_set: 102894
get_misses: 567855
written_bytes: 178511545
read_bytes: 401155623
object_bytes: 111948672

Run time: 10.0s Ops: 1028864 TPS: 102839 Net_rate: 55.3M/s
```

Figure 9: Operations on Memcached

6.3 Performance Analysis: KVM's Default Migration

To evaluate and represent the performance of memcached server during the VM migration average latency and throughput is measured by the memaslap application.

The graphs fig.10 and fig.11 show the throughput and average latency of the memcached server during the VM migration using KVM's default migration. The graphs show that the evaluation lasts 180 seconds and that there is no degradation in performance observed during or after the VM transfer. The blue box represents the time span during which the VM is migrating from host to client.

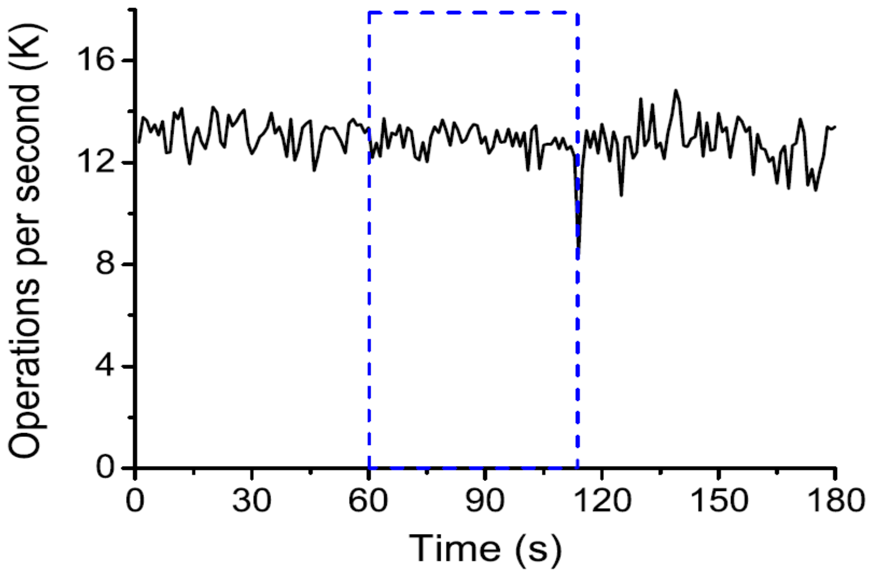


Figure 10: Throughput

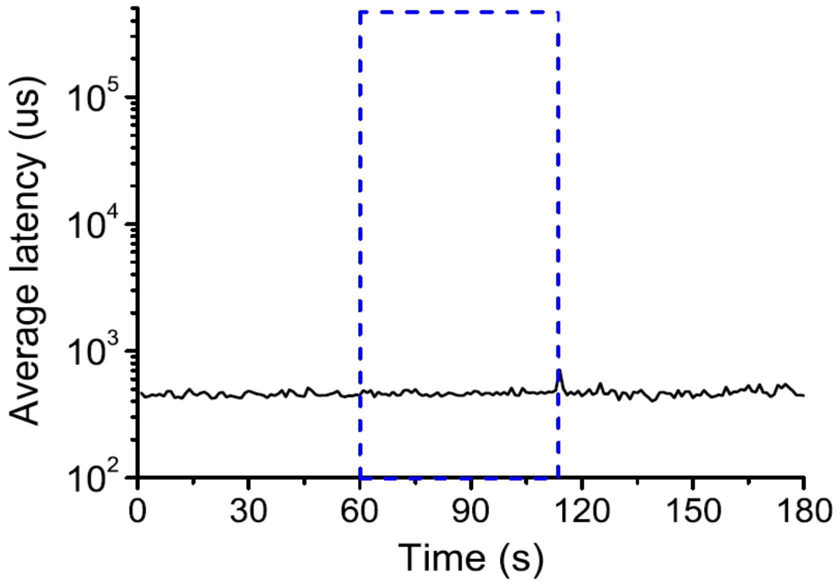


Figure 11: Average Latency

6.4 Performance Analysis: Enhanced Migration

This research work is carried out to improve the functioning of the pre-copy Virtual Machine migration mechanism. The algorithm is defined to achieve an impactful performance during migration by reducing the overall migration time and data transmission between hosts and clients. The work leverage open-source software like Oracle VirtualBox to create a test environment to successfully execute the migration process, Memcached along with memaslap applications is utilized to evaluate the impact of migration on an up and running application. The suggested approach is combined with the default migration mechanism of KVM. However, due to limitations imposed by the Linux operating system and the QEMU-KVM hypervisor, the suggested method was unable to integrate. When the SHA-256 method was used with migration files to transform memory pages into safe hashes, the process became stopped, resulting in migration failures and the virtual machine failing in the majority of the experiments. To avoid retransfer of pages in subsequent rounds, the suggested method can construct an array and retain the transferred memory pages required for comparison. The proposed solution falls short in terms of dealing with the non-compatible hashing method problem, and the project's implementation on VirtualBox was inconclusive since the VM guest was created on VirtualBox on top of the Windows operating system did not operate properly. Experiments using virtual machine migration simulators like HTC-Sim Forshaw et al. (2016) with some changes to let the hashing method access the memory pages during migration are required to enhance the suggested technique and obtain satisfying results.

7 Conclusion and Future Work

The research is motivated to find out the performance improvement of VM migration in terms of migration time and data transmission in the pre-copy technique. The successful implementation of this research work would aid cloud data center efficient management. The work is in line with 3 out 4 objectives mentioned in section 1.4. Nonetheless, fails to evaluate the performance of the proposed algorithm with KVM's default migration algorithm. The proposed algorithm design and use of hashing algorithm did not prove to be as effective as intended. To prevent retransferring pages in subsequent rounds, the proposed approach can build an array and save the transferred memory pages for comparison, however, algorithm design and architecture need modification. Also, the use of VirtualBox to execute experiments imposed many hurdles. In the future, simulators such as HTC-Sim, as well as some adjustments to the recommended methodology to allow the hashing method to access memory pages during migration, will be necessary to get satisfactory results.

References

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A. (2003). Xen and the art of virtualization, Vol. 37, Association for Computing Machinery, New York, NY, USA, p. 164–177. CORE2020 Ranking: A*.
URL: *https://doi.org/10.1145/1165389.945462*

Cho-Chin Lin, Yu-Chi Huang and Zong-De Jian (2012). A two-phase iterative pre-copy strategy for live migration of virtual machines, *2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT)*, Vol. 1, Seoul, Korea (South), pp. 29–34. CORE2020 Ranking: A.

Choudhary, A., Govil, M. C., Singh, G., Awasthi, L. K., Pilli, E. S. and Kapil, D. (2017). A critical survey of live virtual machine migration techniques, *Journal of Cloud Computing* **6**(1): 23. JCR Impact Factor: 2.788.
URL: *https://doi.org/10.1186/s13677-017-0092-1*

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I. and Warfield, A. (2005). Live migration of virtual machines, *Proceedings of the 2nd Conference on*

Symposium on Networked Systems Design and Implementation - Volume 2, NSDI'05, USENIX Association, Anaheim, CA, USA, p. 273–286. CORE2020 Ranking: B.

Dixit, K. M. (1991). The spec benchmarks, *Parallel Computing* **17**(10): 1195–1209. Benchmarking of high performance supercomputers.
URL: *https://www.sciencedirect.com/science/article/pii/S016781910580033X*

Egger, B., Gustafsson, E., Jo, C. and Son, J. (2015). Efficiently restoring virtual machines, *International Journal of Parallel Programming* **43**(3): 421–439.
URL: *https://doi.org/10.1007/s10766-013-0295-0*

Forshaw, M., McGough, A. and Thomas, N. (2016). Htc-sim: a trace-driven simulation framework for energy consumption in high-throughput computing systems, *Concurrency and Computation: Practice and Experience* **28**(12): 3260–3290.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3804*

He, T., N. Toosi, A. and Buyya, R. (2019). Performance evaluation of live virtual machine migration in sdn-enabled cloud data centers, *Journal of Parallel and Distributed Computing* **131**: 55–68.
URL: *https://www.sciencedirect.com/science/article/pii/S074373151830474X*

Hu, B., Lei, Z., Lei, Y., Xu, D. and Li, J. (2011). A time-series based precopy approach for live migration of virtual machines, *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, Tainan, Taiwan, pp. 947–952. CORE2020 Ranking: B.

Kuhn, D., Kim, C. and Lopuz, B. (2015). *Chapter 12: VirtualBox for Oracle*, Apress, Berkeley, CA, pp. 325–344.
URL: *https://doi.org/10.1007/978-1-4842-1254-7*$_1$2

Sharma, S. and Chawla, M. (2016). A three phase optimization method for precopy based vm live migration, *SpringerPlus* **5**(1): 1022. JCR Impact Factor:2.015.
URL: *https://doi.org/10.1186/s40064-016-2642-2*

Wang, C., Hao, Z., Cui, L., Zhang, X. and Yun, X. (2017). Introspection-based memory pruning for live vm migration, *International Journal of Parallel Programming* **45**(6): 1298–1309. JCR Impact Factor:1.125.
URL: *https://doi.org/10.1007/s10766-016-0471-0*

Zhang, F., Liu, G., Zhao, B., Kasprzak, P., Fu, X. and Yahyapour, R. (2018). Cbase: Fast virtual machine storage data migration with a new data center structure, *Journal of Parallel and Distributed Computing* **124**. JCR Impact Factor:2.296.