Enhancing the Performance of Live Migration by Mitigating Redundant Memory Page Transfers in Pre-Copy Migration

Interim Report
SCS 4224 : Final Year Project

Samindu Cooray
Student of University of Colombo School of Computing
Email: 2020cs025@stu.ucsc.cmb.ac.lk

November 12, 2024

Supervisor: Dr. Dinuni K Fernando

Declaration

The interim report is my original work and has not been submitted previously for any examination/evaluation at this or any other university/institute. To the best of my knowledge, it does not contain any material published or written by another person, except as acknowledged in the text

Student Name : P.L.S.R. Cooray
Registration Number : 2020/CS/025
Index Number : 20000251

Signature & Date

This is to certify that this interim report is based on the work of Mr. P.L.S.R. Cooray under my supervision. The interim report has been prepared according to the format stipulated and is of an acceptable standard.

Supervisor Name : Dr. D.K.Fernando

Signature & Date

Contents

List of Figures

List of Tables

Acronyms

AWS Amazon Web Services. 1

CC Cloud Computing. 1

CDCs Cloud Data Centers. 1, 2

DT Downtime. 1, 7, 10, 15

FDP Fake Dirty Page. ii, 1, 7–9

FDPS Fake Dirty Pages. iii, 1, 2, 7–11, 13–15, 18, 19

GCP Google Cloud Platform. 1

HBFDP Hash-Based Fake Dirty Prevention. iii, 10, 11, 15–17

HCM Hybrid Migration. iii, 1, 6

JIT Just-In-Time. 18

PosC Post-Copy. 1, 6

PosCM Post-Copy Migration. iii, 1, 5, 6

PreC Pre-Copy. iii, 1, 4, 6, 8, 11, 13–18

PreCM Pre-Copy Migration. iii, 1–11, 13, 14

TMT Total Migration Time. 1, 7, 10, 15–17

VM Virtual Machine. ii, iii, 1–8, 13–15, 18, 22

VMs Virtual Machines. 1, 2, 10–12, 14, 15

XBZRLE XOR Binary Zero Run Length Encoding. iii, 10, 11, 15–17

1 Introduction

Live Virtual Machine (VM) Migration is the process of migrating a VM from a source host machine to a destination host machine without stopping the VM at the start of the migration procedure providing uninterrupted services to end-users throughout the migration. The efficiency and the performance of a Live VM Migration technique are evaluated using attributes such as Total Migration Time, Downtime, Service Degradation, Network Traffic, and Network bandwidth utilization.

When considering Pre-Copy Migration, total migration time is regarded as the primary bottleneck due to its unpredictability (Elsaid et al. 2022). The iteration count required to transfer memory pages cannot be precisely estimated due to varying dirty page rates and network transmission speed. In extreme cases, rapid dirty page generation and low network bandwidth can lead to prolonged total migration time or even migration failure. Therefore rapid dirty page generation is a significant problem in Pre-Copy Migration (Shribman & Hudzia 2013).

However, most of these generated dirty pages are not dirty. They are identical to their existing copies stored in the destination host, where transferring them wastes network bandwidth and migration time. Exploring the root of this inefficiency, Li et al. (2019) identified two reasons for the generation of Fake Dirty Pages. The primary reason is the "write-not-dirty" request issued by silent store instructions. The secondary reason is the defects in the mechanism that tracks dirty pages. Furthermore, Li et al. (2019) conducted experiments on Pre-Copy Migration to observe Fake Dirty Page generation when migrating VMs with memory-intensive workloads. The experiments showed that some workloads generated high amounts of Fake Dirty Pages while others generated significant amounts. Therefore by eliminating these Fake Dirty Page performance matrices such as total migration time, downtime and application performance degradation can be improved.

On the other hand, if Post-Copy Migration approach is used to migrate a VM running memory-intensive workloads the problem of prolonged total migration time for memory-intensive workload in Pre-Copy Migration is resolved. But Post-Copy Migration inherently contains two weaknesses: Less robust and performance degradation of an application running in the VM which is migrated. When considering Hybrid Migration, it allows memory-intensive workloads to be handled without migration failures by combining Post-Copy and Pre-Copy migration techniques. However, it still inherits the problem of missing the optimal converging point from Pre-Copy Migration (Li et al. 2019).

1.1 Motivation

Cloud Computing (CC) is an essential technology in the modern world due to its efficient and reliable services. At present platforms like Microsoft Azure, Amazon Web Services (AWS), Alibaba Cloud, Google Cloud Platform (GCP) are some of the CC service providers (Rayaprolu 2024). As these services are used by millions of users worldwide, Cloud Data Centers (CDCs) manage these resources, maintaining high-speed network connections for uninterrupted services. However, CDCs consume a significant amount of electricity, with over 30% of the servers using energy while being idle. As a solution, CDCs are embracing virtualization (Jul et al. 2005), allowing OS instances to run concur-

rently on a physical machine, providing individual OS isolation, efficient resource usage, and high performance (Barham et al. 2003).

With the use of virtualization, CDCs maintain servers, which host multiple Virtual Machines (VMs) in each server. These servers can be subjected to various hardware or software failures. This affects the interruptions to the services provided by the VMs in the failing server. When these failures are identified the VMs in the server are migrated to another physical server by using Live VM Migration techniques. However, migrating VMs running memory-intensive workloads is an intractable problem that may increase the total migration time and downtime or lead to migration failure.

The recent work of Nathan et al. (2016) discovered a previously unidentified phenomenon: the presence of **Fake Dirty Pages** during Pre-Copy Migration. While marked as modified and sent to the destination host, these pages are identical to their existing copies, which are already stored in the destination. According to the observations of Li et al. (2019), they have identified that during Pre-Copy Migration of VMs running memory-intensive workloads, some workloads generated high amounts of Fake Dirty Pages while others generate significant amounts. Transferring these Fake Dirty Pages waste the bandwidth and increases the total migration time and downtime.

1.2 Background - Live VM Migration

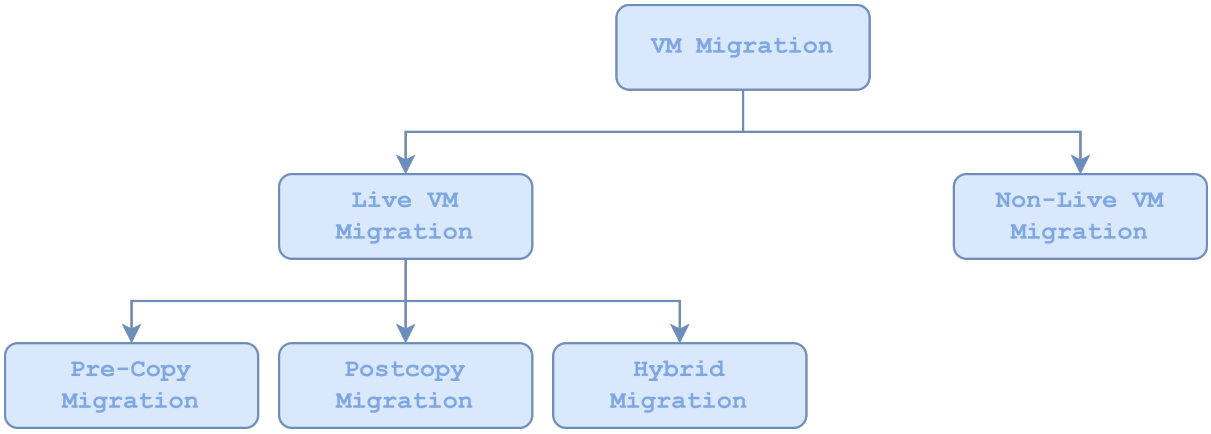

Figure 1: Classification of VM Migration

Migrating a VM to a destination host machine from a source host machine is called VM Migration . VM migration can be categorized mainly into two categories **Non-Live Migration** and **Live Migration** as shown in the Figure 1. When considering Non-Live Migration (Milojičić et al. 2000), the VM is stopped before starting the migration procedure and again resumes it in the destination host machine after transferring the VM completely. Unlike Non-Live Migration, in Live Migration, before starting the migration procedure, the VM is not stopped, providing uninterrupted services to end-users throughout the migration.

Following are some of the benefits as stated in Jul et al. (2005),

- After migration completion, the original host machine does not have to be available for access.
- The clients are not required to reconnect again after migration.

- The operators of the data centers can perform live migration without being concerned about the operations running in the VM.

Further, Live VM Migration has adapted over time to minimize the service downtime (Zhu et al. 2013), efficiently utilize the network bandwidth (Svärd et al. 2011), and decreasing the performance degradation (Ibrahim et al. 2011).

1.2.1 Live VM Migration

In Live VM Migration, the main concern is to minimize both *total migration time* (the duration from initiating the migration to completing the migration process) and *downtime* (the period where the VM is suspended).

Generally, There are three phases when transferring the memory of a VM:

1. **Push Phase**

   Memory pages assigned to the VM are migrated over the network to the destination from the source. Since the VM is continuously working in the source host, some memory content may get modified; therefore, these modified copies should be again sent to the destination.

2. **Stop-and-Copy Phase**

   The source stops the execution of the VM and transfers every memory content to the destination. Upon the migration completion, the destination starts the VM

3. **Pull Phase**

   After starting the VM in the destination, memory content is fetched from the source upon page faults.

Out of the above-mentioned phases, one or two are selected in practical solutions. Sapuntzakis et al. (2002) and Kozuch & Satyanarayanan (2002) state a *pure stop-and-copy* method that only uses the stop-and-copy phase and Zayas (1987) states a *pure demand-migration* method which uses both stop-and-copy and pull phases. However, applying them in Live Migration may lead to unacceptable results.

Based on the method of memory content transferring, there are three Live Migration techniques. They are *Pre-Copy Migration* (Jul et al. 2005), *Post-Copy Migration* (Hines et al. 2009) and *Hybrid Migration* (Sahni & Varma 2012)

1.2.1.1 Pre-Copy Migration

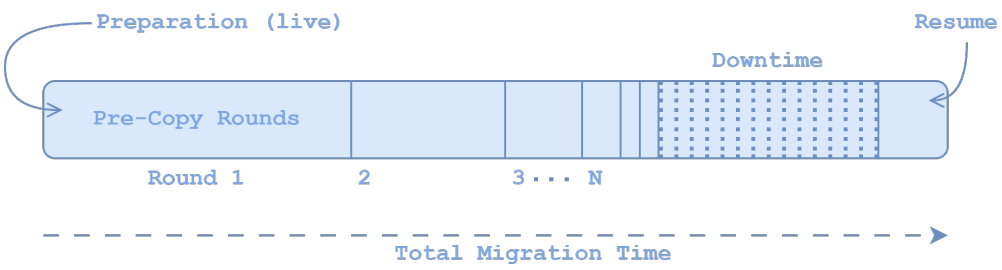

Figure 2: Pre-Copy Migration Timeline (Hines et al. 2009).

At the initial round of *Pre-Copy Migration*, every page assigned to the VM are migrated into the destination host. Then, in successive rounds, the memory content modified (dirtied) during the earlier round by the VM is sent to the destination. These rounds are stopped when the page dirtying rate converges to a specific point. Then the remaining dirty pages (Writable Working Set) and the CPU state are transferred to the destination after the source stops the VM. Figure 2 graphically demonstrates the Pre-Copy Migration procedure. Then the destination starts the VM, making it the primary host and the source host discards the VM from it (Jul et al. 2005).

Therefore the proposed *Pre-Copy Migration* by Jul et al. (2005) combines iterative push phase with short stop-and-copy phase.

Pre-Copy Migration Stages

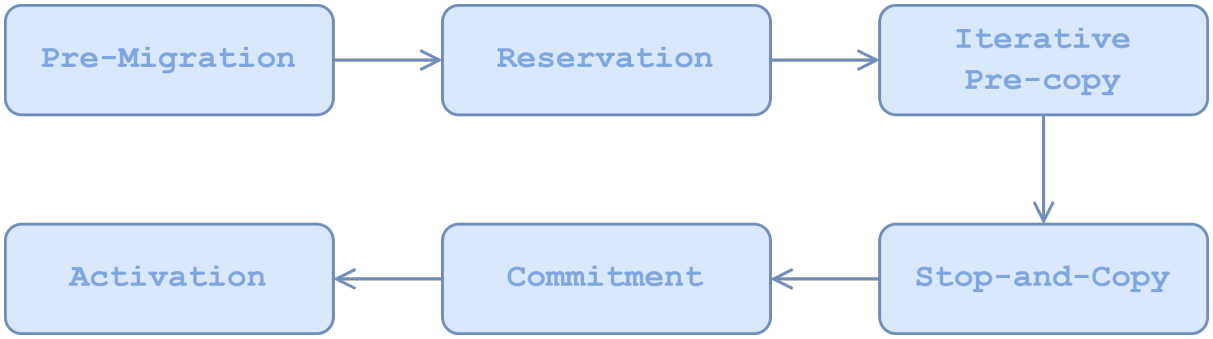

Figure 3: Stages in Pre-Copy VM Migration (Jul et al. 2005).

1. **Pre-Migration:** The VM is active in the source host.
2. **Reservation:** Reserves a container on the destination host after confirming the resource availability.
3. **Iterative Pre-Copy:** All the memory pages are transferred to the destination host in the initial iteration. Then, only the modified (dirtied) pages are transferred in the next iterations.
4. **Stop-and-Copy:** The CPU state and the remaining memory pages are transferred to the destination after the source stops the Virtual Machine.
5. **Commitment:** The destination informs the source that the migration of the VM is completed the destination received it, and the source sends an acknowledgment to this message. The source discards the VM.
6. **Activation:** The destination host becomes the primary host and runs a post-migration code.

Pre-Copy Migration Techniques

There are various number of Pre-Copy Migration techniques, given below are some of those techniques:

- **Dynamic Rate-Limiting :** Handle the Bandwidth Utilization in the iterative push phase and stop-and-copy phase (Jul et al. 2005).

- **Data Compression :** Exploits word-level duplication in data to reduce data transfer (Jin et al. 2009).
- **Migration Control Method :** Detects memory modification patterns and terminated migrations with high downtime (Ibrahim et al. 2011).
- **Delta Compression :** The modifications done to the data are stored without storing the full data sets (Svärd et al. 2011).
- **Memory Compaction Technique :** A technique that is a combination of memory snapshot and disk-memory deduplication cache (Piao et al. 2014).
- **Adaptive VM downtime control technique** (Piao et al. 2014)
- **Deduplication :** Transfer only one copy of duplicate pages (Wood et al. 2015).
- **Three Phase Optimization :** Minimizes the memory page transfer in transferring rounds (Sharma & Chawla 2016).
- **Page Content Reduction Technique :** Transferring encoded form of XOR differential page without transferring the whole page. (Bhardwaj & Krishna 2019)

1.2.1.2 Post-Copy Migration

Pre-Copy Migration effectively minimizes the application performance degradation and downtime when executing CPU-Intensive workloads in the VM. However, its performance for memory-intensive workloads is less effective because of the high page dirty rates. To solve this problem, Hines et al. (2009) introduces Post-Copy Migration as another Live VM Migration technique.

Post-Copy Migration initially transfers the CPU states to the receiving host after stopping the VM in the source. Then, the destination host starts the VM without memory content. After that, the migration thread starts the active push of memory pages to the destination from the source. Concurrently, fetching of the page-faulted memory pages in the destination is done on-demand from the source (Hines et al. 2009). Figure 4 graphically demonstrates the Post-Copy Migration procedure.

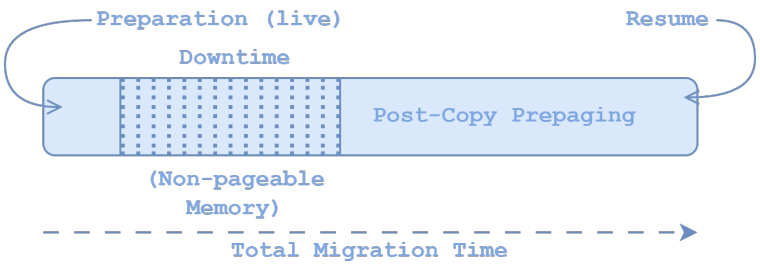

Figure 4: Post-Copy Migration Timeline (Hines et al. 2009).

Post-Copy Migration Variants

There are different variants of Post-Copy Migration, pivoting on different techniques of page retrieval from source to destination. These techniques are:

- **Active Push :** Proactively pushes pages from source to destination (Hines et al. 2009).

- **On-Demand Paging :** Only page-faulted pages are retrieved from the source (Hines et al. 2009).
- **Pre-Paging :** Guess future page faults and adjust retrieval patterns to avoid page faults (Hines et al. 2009).
- **Self Ballooning :** Reduces the transfer of unused pages (Hines et al. 2009).

Weaknesses in Post-Copy Migration

Although the Post-Copy Migration resolves Pre-Copy Migration's problem with dirty pages not converging to a minimum value to complete the migration (Li et al. 2019), it has two inherent weaknesses:

1. **Less Robust:** When considering Pre-Copy Migration, the VM in the source host will still be working although the migration fails due to a problem in the migration network or destination host. But in Post-Copy Migration, since both destination and source hosts contain a piece of the latest state of memory, it will cause the failure of the VM.
2. **Performance Degradation:** In Post-Copy Migration, access latency for memory is increased due to continuous page faults resulting in performance degradation of applications in the VM.

1.2.1.3 Hybrid Migration

By combining migration techniques Post-Copy and Pre-Copy, Hybrid Migration is performed. The migration procedure starts with a few Pre-Copy rounds, as shown in Figure 5. Next, the CPU state is migrated to the destination host after the VM is stopped in the source, and upon the completion of the transfer the destination host starts the VM. Then, the destination host retrieves the pages using one or more techniques mentioned in the above section. Therefore, as Hybrid Migration contains combined advantages of both Post-Copy and Pre-Copy Migrations, memory-intensive workloads can be handled without any migration failures (Sahni & Varma 2012).

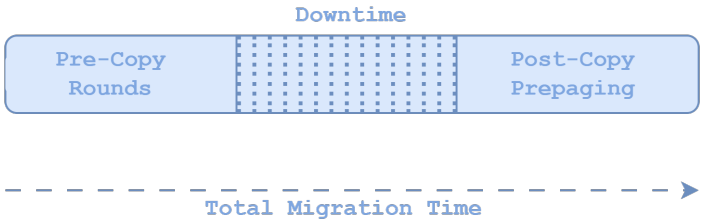

Figure 5: Hybrid Migration Timeline

When looking into the optimizations in Hybrid Migration, Li et al. (2019) introduces an Intelligent Hybrid Migration approach. In traditional Hybrid Migration, the data-center admin might have to switch manually to Post-Copy from Pre-Copy or set to shift after executing fixed two or three Pre-Copy rounds. But the proposed intelligent Hybrid Migration will minimize the number of page faults in the Post-Copy phase by shifting to Post-Copy from Pre-Copy at a nearly ideal point.

2 Literature Review

The efficiency and the performance of a Live VM Migration technique is evaluated using attributes such as Network Traffic, Downtime, Total Migration Time, Service Degradation, and Network bandwidth utilization. When considering Pre-Copy Migration, total migration time is regarded as the primary bottleneck due to its unpredictability. Two significant factors directly impacting the total migration time and downtime are the dirty page generation rate and VM size. The total migration time of Pre-Copy Migration relies on two distinct phases: Initial iteration that transfers whole memory and subsequent iterations that transfer dirty pages. The time allocation between these phases depends heavily on the VM's memory behavior.

Problems that directly affect the time consumption in the push phase, resulting in higher total migration time in Pre-Copy Migration, are Rapid Dirty Page Generation, Page Level Content Redundancy, Lower Network Transferring Rates, and Fake Dirty Pages.

Additionally, Performance and energy consumption concerns also arise, including service degradation due to resource competition and data transfer overhead and increased CPU activity, VM size and network bandwidth, resource consumption on hosts, and state transfer and I/O bandwidth, respectively.

2.1 Fake Dirty Problem

As mentioned in Section 2, *Rapid Dirty Page Generation* is one of the significant problems in Pre-Copy Migration. However, most of these generated dirty pages are actually **not dirty**. These pages, marked as modified and transferred, are identical to their existing copies stored in the destination host. This unnecessary duplication wastes network bandwidth and migration time (Li et al. 2019, Nathan et al. 2016). Li et al. (2019) observed the Fake Dirty Page generation in Pre-Copy Migration when migrating VMs running memory-intensive workloads.

2.2 Reasons for Fake Dirty Page Generation

There are two reasons identified by Li et al. (2019) for Fake Dirty Page generation. The primary reason is **"write-not-dirty"** request issued by *silent store instructions*. The secondary reason is the defects in the mechanism that tracks dirty pages.

2.2.1 "Write-not-dirty" Pages

The first reason for generating Fake Dirty Pages are "Write-not-dirty" Pages. These **Write-not-dirty** pages are memory pages marked as dirty but do not have actual content change(Li et al. 2019).

Silent store instructions are the main cause for the write-not-dirty requests (Molina et al. 1999, Lepak & Lipasti 2000*a*,*b*). These instructions write a value to a memory address exactly like the existing value, resulting in no system state change. According to the evaluations done by Lepak & Lipasti (2000*a*), an average amount of 20% to 68% are silent store instructions among all the store instructions executed. Also, the analysis done by Lepak & Lipasti (2000*b*) identified that silent store instructions execute in compiler optimizations and all levels of program executions and have an algorithmic nature.

Therefore, in workloads wide amounts Fake Dirty Pages are generated, due to write-not-dirty requests resulting in prolonged total migration time (Li et al. 2019).

2.2.2 Defective Dirty Page Tracking

The second reason for generating Fake Dirty Pages is the method used by the migration thread for dirty page tracking during migration. Both hypervisors, KVM(Bellard 2005) and XEN(Barham et al. 2003) use bitmaps to keep track of these dirty pages.

2.2.2.1 KVM

The virtualization platform KVM/QEMU (QEMU 2024, KVM 2024) can be divided into KVM (kernel module) and QEMU (userspace tool). Both these parts contain their dirty bitmap. The dirty bitmap of the kernel traces the write request to a page and marks the page as dirty. This process is continued throughout an iteration, marking all the dirty pages. When the iteration ends, the kernel's dirty bitmap is synced with userspace's dirty bitmap using API: *ioctl*, and the dirty bitmap of the kernel is reset to track the next iteration. In the next iteration, the synced dirty bitmap of the userspace is used to transfer modified pages. Therefore, the tracking of dirty pages is limited to one Pre-Copy round, resulting in Fake Dirty Page generation.

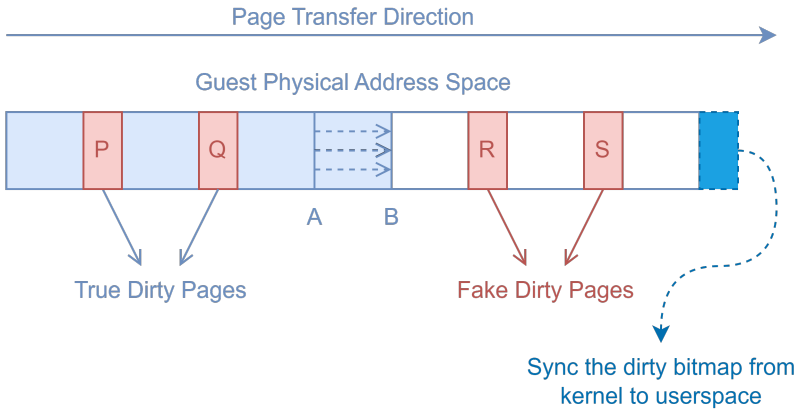

Figure 6: Tracking Mechanism of Dirty Pages in KVM (Li et al. 2019).

Li et al. (2019) used Figure 6 to explain how Fake Dirty Page are generated due the tracking mechanism. During the first iteration of Pre-Copy Migration, the pages in the memory region allocated to the VM are transferred by the migration thread to the destination starting from a low address in a sequential order. Let's assume that during the transfer of pages to the destinations from address A to address B, four pages, P, Q, R, and S, are modified and marked as dirty in the kernel dirty bitmap. At that moment, since P and Q are already transferred in the ongoing iteration, the modifications done to them are not sent to the destination, making them dirty pages in the second iteration. However, pages R and S are dirtied before they are transferred, and as a result, the modified versions of R and S are transferred in this current iteration, although pages R and S are marked as dirty in the kernel's dirty bitmap. When the kernel's dirty bitmap syncs with the userspace's dirty bitmap, pages R and S are selected to resent it in the second iteration. Therefore, in the second iteration, when pages R and S are resent, they already exist in the destination, making pages R and S fake dirty pages.

2.2.2.2 XEN

XEN (XEN 2024) maintains three bitmaps in the migration thread to mark the pages that must be transferred.

Below are the functionalities of the three bitmaps maintained by XEN:

1. **to_send:** Pages dirtied during an iteration are marked in this bitmap. Therefore, this bitmap is referred to by the migration thread to transfer pages in the next iteration. The to_send bitmap is updated at the end of every iteration by referring to the kernel bitmap and then resetting it.
2. **to_skip:** Pages that should not transfer in the current iteration but are transferred in the next iteration are marked in this bitmap. This bitmap is updated by the migration thread from the kernel bitmap in a random iteration without resetting the kernel bitmap.
3. **to_fix:** Pages that are transferred in the stop-and-copy phase are marked in this bitmap.

Compared with KVM, XEN's tracking mechanism works well to minimize Fake Dirty Page generation with the use of *to_skip* bitmap as it avoids written dirty pages to be transferred in the current iteration. But if a page is dirtied before being transferred and after the *to_skip bitmap* is updated, it is the same problem as the KVM's tracking mechanism, making the page fake dirty (Li et al. 2019).

In this next section 2.3, let's explore the proposed solutions to avoid the transmission of Fake Dirty Pages (FDPS) during migration.

2.3 Solution proposed in Previous Studies

According to Nathan et al. (2016), delta compression proposed by Zhang et al. (2010), Svärd et al. (2011) and deduplication proposed by Deshpande et al. (2011), Wood et al. (2015), have the inherent nature to prevent the transferring of fake dirty pages.

Li et al. (2019) proposed a solution to avoid Fake Dirty Pages transfer using secure hashes. Their solution stores the secure hashes of all the transferred pages in the initial iteration. Then in the next iterations, the pages marked as dirty are compared with the previously stored respective secure hashes before transferring. If the new and the old hashes of a page differ, the page is transferred to the destination as a dirty page, and the new version hash replaces the stored former hash. Otherwise, if the hashes are similar, the page is a Fake Dirty Page where it is not transferred to the destination. In this solution, Li et al. (2019) uses SHA1 (SHA1 2024) secure hashes to compute the hash of a page. They consider two similar hashes to be just hashes of an unchanged page as the hash collision probability of SHA1 is less than 10^{-31}. Therefore, the theoretical hash collision scenario can be neglected.

3 Research Gap

The state-of-the-art technique for reducing redundant data transfers in Pre-Copy Migration for memory-intensive workloads proposed by Li et al. (2019) uses SHA1 secure hashes

to identify and eliminate Fake Dirty Pages, mark significant advancements in reducing redundant data transfers. However, this approach raises questions about the efficacy and potential benefits of alternative, more sophisticated hashing methods. While using SHA1 hashes effectively minimizes the transfer of Fake Dirty pages, exploring different hashing techniques could further enhance this process, reducing the data transfer load even more efficiently.

Furthermore, the existing study has not considered applying different optimization techniques along with the Hash-Based Fake Dirty Prevention (HBFDP) algorithm. This application can significantly enhance the efficiency of VM migration by minimizing total migration time and downtime.

By addressing these research gaps, this study aims to improve hash computation speed and improve the performance of Pre-Copy Migration when migrating Virtual Machines running memory-intensive workloads.

4 Research Questions

1. How to mitigate redundant data transfers by identifying fake dirty pages?
    - This research question investigates how different hashing techniques can be used to find page similarities with minimal overhead on migration and Virtual Machines. And explore on optimizing the dirty page tracking mechanism in Qemu to prevent Fake Dirty Page transfers.
2. How to reduce the data transferring load in Pre-Copy Migration to improve the performance of memory-intensive workloads?
    - This research question investigates how to incorporate different optimization techniques *(XBZRLE, Delta compression, etc)* to improve migration efficiency.

5 Aims and Objectives

5.1 Aim

Improving the performance matrices: Total Migration Time, Downtime, and Application Performance Degradation in Live Migration by Mitigating Redundant Memory Page Transfers in Pre-Copy Migration

5.2 Objectives

- To evaluate the performance improvement of using different hashing mechanisms to reduce redundant memory page transfers by considering Fake Dirty Pages
- To evaluate the performance improvement of combining different optimization techniques to reduce the data transferring load in Pre-Copy Migration to improve the performance for memory-intensive workloads?

6 Scope

6.1 In Scope

The following areas will be covered under the scope of this research.

- Develop a prototype of the algorithm proposed by Li et al. (2019) using SHA1 hashes in QEMU-KVM and developing variations of the prototype by replacing SHA1 with different hashing techniques.
- Evaluate the developed prototypes with and without incorporating different optimization techniques to show that the total migration time can be reduced by comparing against Vanilla Pre-Copy, XBZRLE, and the algorithm proposed by Li et al. (2019) *(Prototype developed with SHA1 hashes).*
- The research focuses on running single VM migrations on Ubuntu Server as the host OS within a LAN environment, with potential for future extension to WAN and multiple VM migrations.

7 Significance of the Research

As stated in Section 2.3, the proposed solution of Li et al. (2019) avoids the transfer of Fake Dirty Pages using **SHA1 hashes**. However, their study only used the SHA1 hashing technique in the algorithm and did not explore the improvement that could be achieved using other hashing techniques. Furthermore, the study of Li et al. (2019) hasn't focused on incorporating their algorithm with other optimizations for further improvements.

This study evaluates the performance improvement of various hashing techniques over SHA1 to reduce redundant memory page transfers by considering Fake Dirty Pages. This approach provides a method to enhance the hash computation speed, thus improving the efficiency of Pre-Copy Migration when migrating VMs running memory-intensive workloads.

Another significance of this research is the potential to combine different optimization techniques with the Hash-Based Fake Dirty Prevention algorithm. This combined approach is expected to minimize total migration time and downtime more effectively than existing methods.

This is particularly significant for ensuring uninterrupted services to end-users by improving key performance matrices such as total migration time, downtime, and Application Performance Degradation.

8 Research Approach

8.1 Testbed Setup

The setup testbed contains two physical servers and an NFS server. The two physical servers act as source and destination servers each equipped with a 48-core Intel Xeon E5-2697 v2 @ 3.500GHz processor with 314.8 GiB of RAM. The two servers and the NFS server are inter-connected with Gigabit Ethernet. The host OS is 22.04.3 LTS

x86_64 Server, and the chosen hypervisor is QEMU-KVM v8.1.2, running various Virtual Machines utilizing Linux-based OSs (guest OS). Figure 7 presents the tentative high-level architecture of the testbed that will be used for data collection.

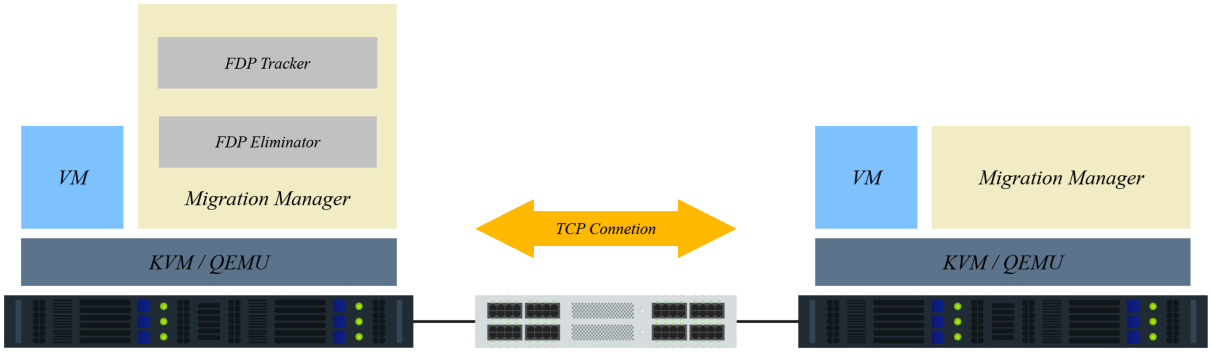

Figure 7: High-level architecture of the Testbed

8.2 Selecting Suitable Workloads

The following workloads are selected to execute on Virtual Machines that are used in migration experiments.

Workloads	Type	Description
Workingset (Fernando et al. 2016)	Memory Intensive	A benchmark that dirty pages to vary writable working set
Quicksort (Fernando et al. 2016)	CPU Intensive	A benchmark that repeatedly allocates random integers to an integer array of 1024 bytes and performs quicksort on the array.
Sysbench (Sysbench 2024)	CPU Intensive	A benchmark to assess the system performance of a machine planning on running a database under intensive load
Memcached (Memcached 2024)	Multiple Resource Intensive	Memcached is an in-memory key-value store storing arbitrary data returned by a benchmark called Memaslap.
YCSB (Benchbase 2024)	Multiple Resource Intensive	A Suite used to evaluate computer programs' retrieval and maintenance capabilities.

Table 1: Workloads used in Experiments

8.3 Preliminary Experiments

The initial experiment was to observe the presence of Fake Dirty Pages in Vanilla Pre-Copy Migration. The experiment was conducted for six Virtual Machine memory sizes *(1GB, 2GB, 4GB, 8GB, 12GB, 16GB)* where each Virtual Machine is configured with 1 vCPU and the Virtual Machine images are stored on the NFS server, where both source and destination host machines can access the Virtual Machine storage over the LAN. The required readings were collected by executing four workloads in each Virtual Machine memory size. Each data point is an average of 3 rounds of experiment.

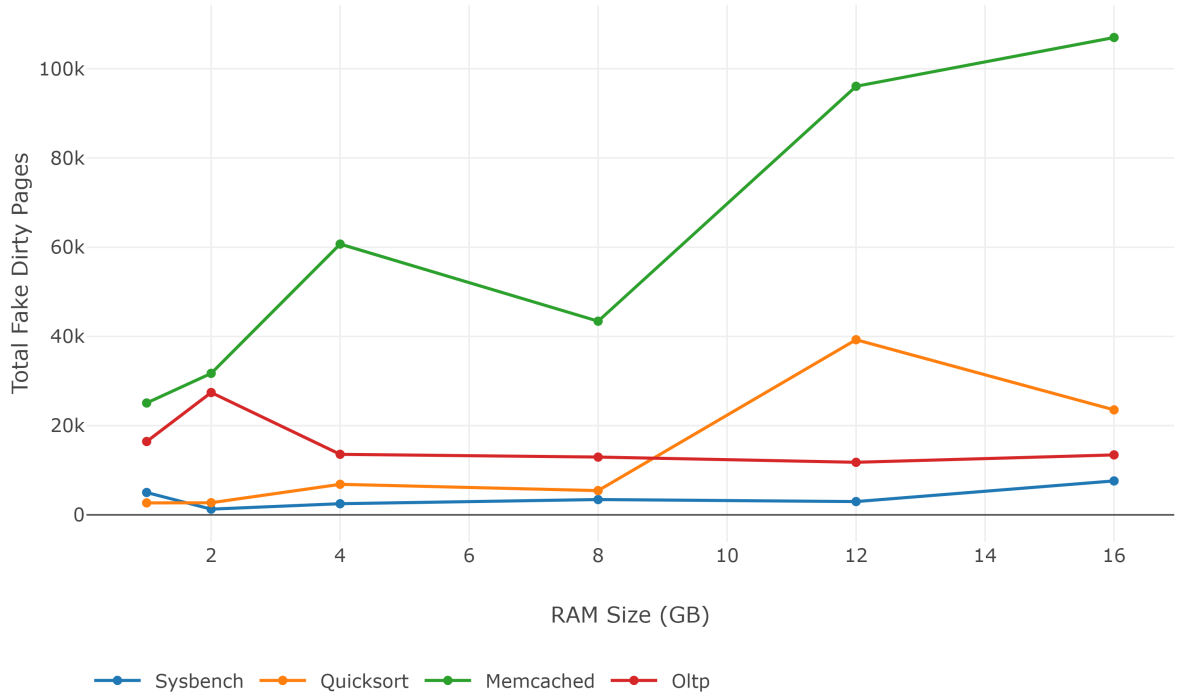

Figure 8: Fake Dirty Page Count in Vanilla Pre-Copy

The used workload is as follows:

- **CPU Intensive Workloads:** Fake Dirty Page generation for CPU-Intensive workloads was observed using Sysbench and Quicksort workloads. The Sysbench benchmark was configured to calculate prime numbers up to 5,000,000 to stress the CPU, and the Quicksort benchmark was configured to repeatedly create an array of size 512, store random numbers in each array index, and sort the array using the quicksort algorithm. The blue and orange lines in Figure 8 show the count of Fake Dirty Pages generated for Sysbench and Quicksort respectively, and it can be seen that both workloads generate a considerable amount of Fake Dirty Pages.
- **Multiple Resource Intensive Workloads:** Fake Dirty Page generation for Multiple-Intensive workloads was observed using Memcached and YCSB workloads. The Memcached workload was configured by allocating 4 threads to handle requests and 60% of the Virtual Machine memory size was allocated as Memcached cache, Memaslap benchmark was configured in the NFS server to send requests to the Memcached in the Virtual Machine with set and get the ratio of 1:0. The YCSB

benchmark was executed in the Virtual Machine to send queries to a PostgreSQL database hosted in the NFS server with 10,000 operations per second. The green and red lines in Figure 8 show the count of Fake Dirty Pages generated for Memcached and YCSB respectively, and it can be seen that both workloads also generate a considerable amount of Fake Dirty Pages.

Therefore, as the Figure 8 shows, a considerable amount of Fake Dirty Pages generated throughout the vanilla Pre-Copy Migration for CPU-Intensive, Memory-Intensive, and Network-Intensive workloads.

RAM (GB)	Fake Dirty Count	Transferred Page Count
1	33603	380511
2	46093	607484
4	79643	1211109
8	58229	2248032
12	113848	3458888
16	138258	4552333

Table 2: Number of Pages Transferred vs Number of Fake Dirty Pages Generated when migrating a single Virtual Machine running memory-intensive workload Memcached using Vanilla Pre-Copy

As shown in Table 2 When considering the number of pages transferred in migrating a single VM running memory-intensive workload Memcached using Vanilla Pre-Copy, We can see that more than 3% of the pages transferred in Vanilla Pre-Copy is Fake Dirty. Although this amount is relatively low considering the number of pages transferred, These fake dirty pages take significant time to transfer, increasing total migration time. Therefore, mitigating the transfer of Fake Dirty Pages is important in real-world situations because data centers migrate multiple Virtual Machines from one server to another, so transferring Fake Dirty Pages from all these Virtual Machines affects the total migration time taken to migrate all these Virtual Machines.

The next experiment was to observe the computation time of different hashing methods. SHA1, MD5, CRC32, and Murmur3 hashing methods were used to conduct this experiment. The experiment was done in the C environment. In the experimental setup, a 4KB memory was allocated with random numbers and hashed using each hashing method, and the computation time was recorded. Figure 9 shows the Average Computation Time of SHA1, MD5, CRC32, and Murmur3.

As shown in the Figure 9 MD5 (MD5 2024), CRC32 (Yangyang 2022), and Murmur3 (Murmur3 2022, Scott 2024) performs better than SHA1 when considering the computation time. Therefore, using MD5, CRC32, or Murmur3 to detect Fake Dirty Pages would be more optimal because the main focus is not on security concerns but on computing the hash of a page quickly and efficiently. Furthermore, more hashing methods can perform better than MD5, CRC32, and Murmur3. Therefore, more hashing techniques can be investigated to find the optimal method.

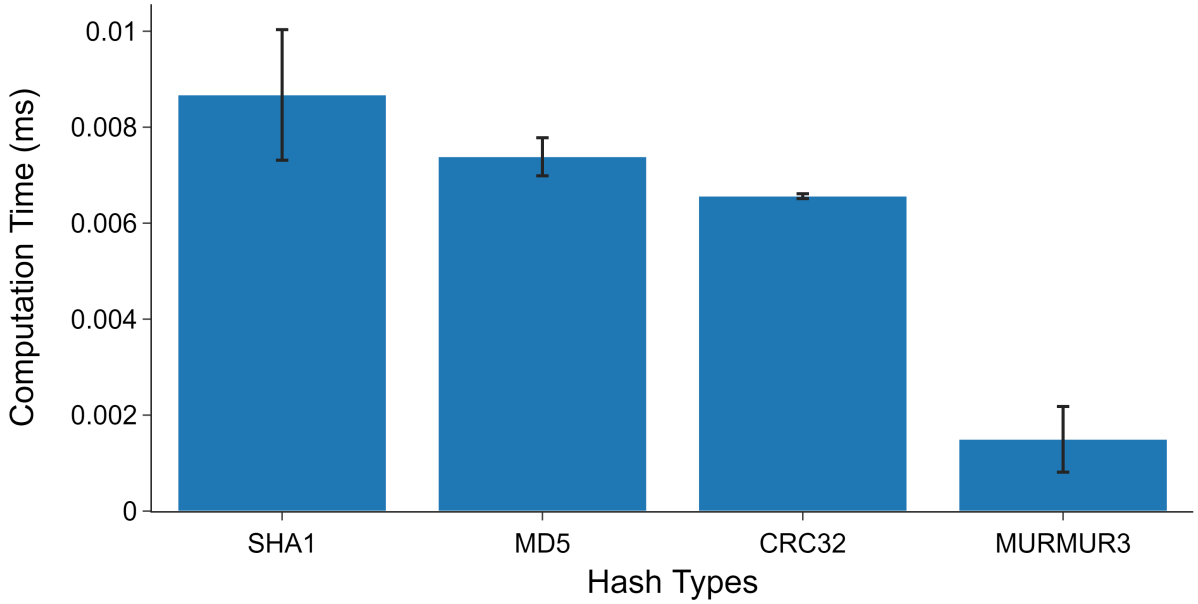

Figure 9: Average Computation Time of SHA1, MD5, CRC32, and Murmur3

8.4 Implementing Prototype

Develop a prototype of the algorithm proposed by Li et al. (2019) using SHA1 hashes in QEMU-KVM to track and avoid the transfer of Fake Dirty Pages in Pre-Copy iterations. Then create prototype variations using different hashing techniques replacing the SHA1 hashing technique.

9 Results and Discussion

9.1 Evaluation

Evaluate the developed prototypes with and without incorporating different optimization techniques against Vanilla Pre-Copy, XBZRLE, Compression enabled Pre-Copy and the algorithm proposed by Li et al. (2019) *(Prototype developed with SHA1 hashes)*. The evaluation is done by comparing the performance matrices such as total migration time, Downtime, and application performance degradation. These data are collected by migrating Virtual Machines running different workloads mentioned in Table 1.

After implementing the prototypes, An experiment was conducted to collect the Total Migration Time, and Downtime for each baseline method and implemented prototypes. For each algorithm, the experiment was conducted for six Virtual Machine memory sizes *(1GB, 2GB, 4GB, 8GB, 12GB, 16GB)* running Memcached workload by allocating 90% of the RAM as Memcached cache. Each data point is an average of 3 rounds of experiment.

9.2 Results

The Figure 10 shows the Total Migration Time migrations completed using Vanilla Pre-Copy, XBZRLE Pre-Copy, Compress Pre-Copy and Hash-Based Fake Dirty Prevention Algorithm with SHA1, MD5 and Murmur3 enabled Pre-Copy. When considering the

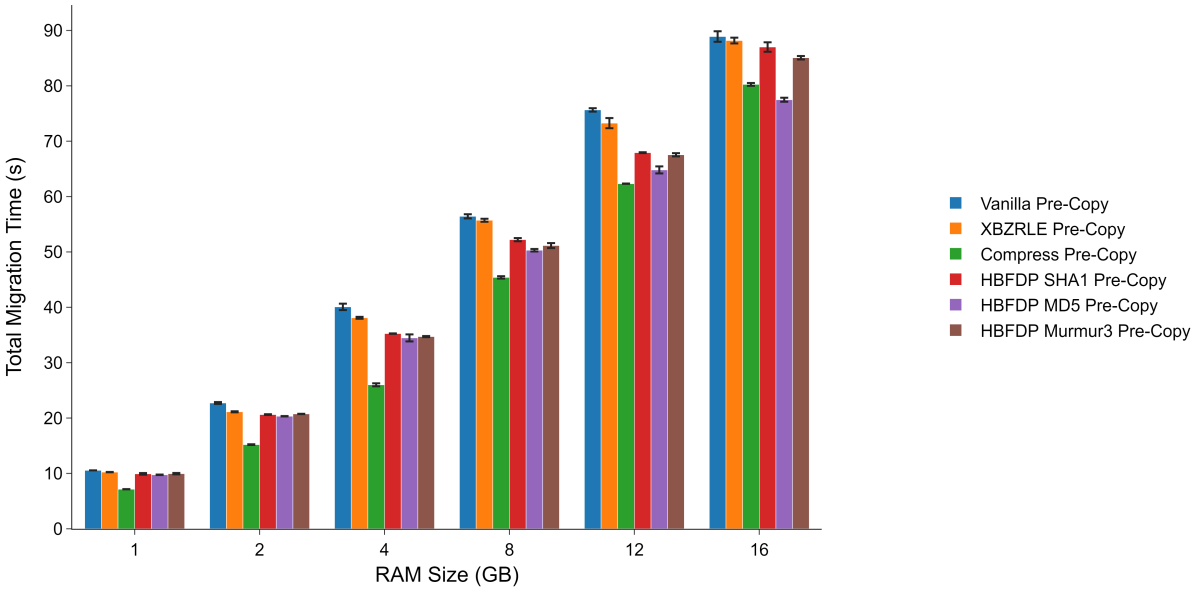

Figure 10: Total Migration Time for Vanilla Pre-Copy, XBZRLE Pre-Copy, Compress Pre-Copy and Pre-Copy with Hash-Based Fake Dirty Prevention Algorithm

Hash-Based Fake Dirty Prevention Algorithm with SHA1, MD5, and Murmur3 they perform better than Vanilla Pre-Copy and XBZRLE Pre-Copy but the compress Pre-Copy technique performs better than HBFDP Pre-Copy although it does not prevent the transfer of Fake Dirty Pages. Also, the HBFDP with MD5 Performs better than the HBFDP with SHA1 and HBFDP with Murmur3.

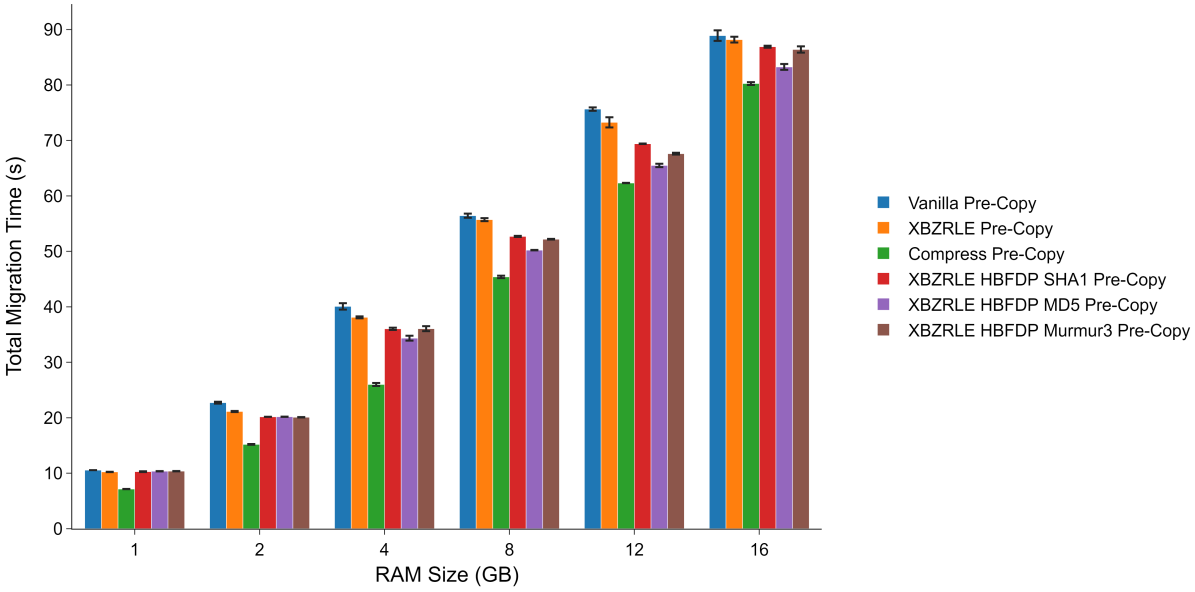

Figure 11: Total Migration Time for Vanilla Pre-Copy, XBZRLE Pre-Copy, Compress Pre-Copy and Pre-Copy with Hash-Based Fake Dirty Prevention Algorithm combined with XBZRLE optimization

The Figure 11 shows the Total Migration Time migrations completed using Vanilla Pre-Copy, XBZRLE Pre-Copy, Compress Pre-Copy and Hash-Based Fake Dirty Prevention Algorithm with SHA1, MD5 and Murmur3 enabled Pre-Copy combined with XBZRLE

optimization. When considering the Hash-Based Fake Dirty Prevention Algorithm with SHA1, MD5, and Murmur3 combined with XBZRLE optimization they perform better than Vanilla Pre-Copy and XBZRLE Pre-Copy but the compress Pre-Copy technique performs better than XBZRLE combined HBFDP Pre-Copy. Also, the XBZRLE combined HBFDP with MD5 Performs better than the XBZRLE combined HBFDP with SHA1 and XBZRLE combined HBFDP with Murmur3.

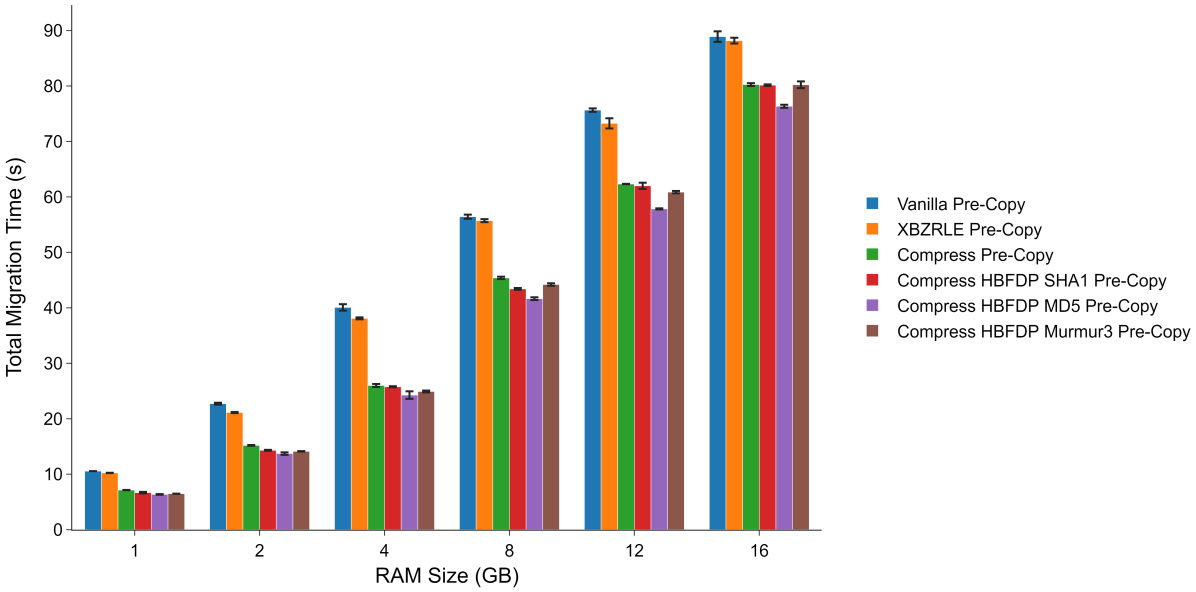

Figure 12: Total Migration Time for Vanilla Pre-Copy, XBZRLE Pre-Copy, Compress Pre-Copy and Pre-Copy with Hash-Based Fake Dirty Prevention Algorithm combined with Compression optimization

The Figure 12 shows the Total Migration Time migrations completed using Vanilla Pre-Copy, XBZRLE Pre-Copy, Compress Pre-Copy and Hash-Based Fake Dirty Prevention Algorithm with SHA1, MD5 and Murmur3 enabled Pre-Copy combined with Compression optimization. When considering the Hash-Based Fake Dirty Prevention Algorithm with SHA1, MD5, and Murmur3 combined with Compression optimization they perform better than all Vanilla Pre-Copy, XBZRLE Pre-Copy and Compress Pre-Copy techniques. Also, the Compress combined HBFDP with MD5 Performs better than the Compress combined HBFDP with SHA1 and Compress combined HBFDP with Murmur3.

9.3 Discussion

According to the Figure 13 HBFDP with MD5 combined with Compression Optimization gives a better improvement in reducing Total Migration Time comparing to Vanilla Pre-Copy, XBZRLE, Compression enabled Pre-Copy and the algorithm proposed by Li et al. (2019) *(Prototype developed with SHA1 hashes)*. According to the data, it can be seen that there is a reduction in Total Migration Time up to 40% in HBFDP with MD5 combined with Compression Optimization compared to Vanilla Pre-Copy.

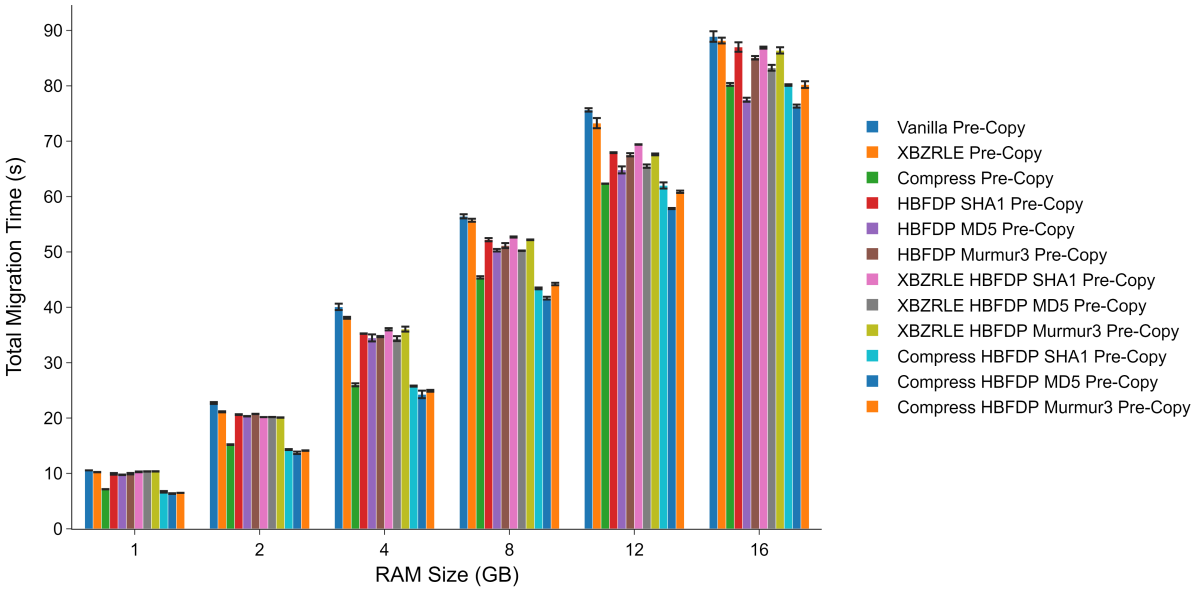

Figure 13: Total Migration Time for Different Pre-Copy Migration Techniques

10 Exploration of Optimizations to Dirty Page Tracking Mechanism

The primary feedback of the evaluation panel was to explore optimizing the dirty page tracking mechanism in Qemu to prevent Fake Dirty Page transfers as it is one of the reasons for generating fake dirty pages which will be the ideal way to prevent the transferring Fake Dirty Pages without computational overhead on the migration thread.

Initial approach was to find function(s) in QEMU that handles *write requests* issued by the Virtual Machine. After some extensive searching, it was found that there is no single function that all guest writes go through. As a primary developer confirmed in cases where a guest write is directed to host RAM, an optimization is employed within the Just-In-Time (JIT) compilation process. Therefore the writes happen directly from the host code that the JIT generates, without going out to any C code in QEMU itself.

Due to the limitations identified in the initial approach, the next approach was to implement a memory listener method to capture write requests using the SIGSEGV signal issued by the operating system's kernel. The idea was to lock the memory allocated to the Virtual Machine and capture segfault signals issued when Virtual Machine tries to write to the memory. This approach was also ineffective since the signals the Virtual Machine issued can not be captured as a SIGSEGV signal in the QEMU source code.

Following the limitations observed in the first and second approaches, The current approach focuses on maintaining a dirty page tracking bitmap which is synced with the Kernel's dirty bitmap before sending a page in an iteration. As shown in Figure 14 the new bitmap is to identify pages that are modified before transferring in the current iteration and not sending them in the current iteration as those pages are sent the next iteration resulting in preventing the transfer of Fake Dirty Pages generated due to QEMUs' dirty page tracking mechanism.

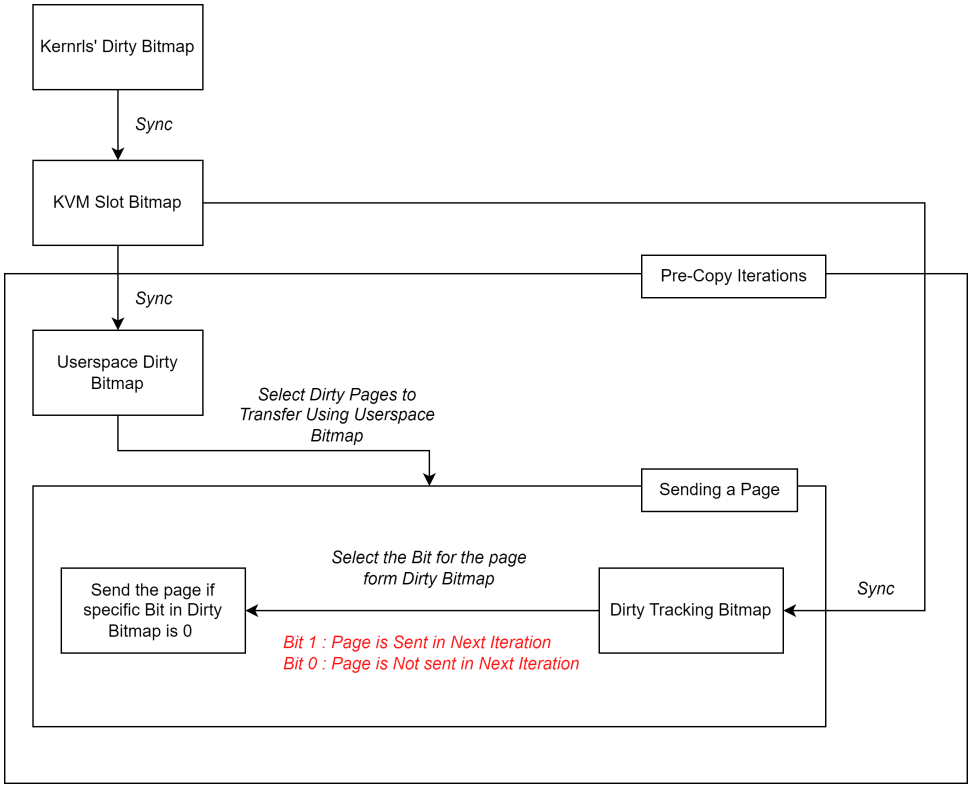

Figure 14: High-Level Diagram of using a Dirty Tracking Bitmap to prevent Fake Dirty Pages

11 Project Timeline

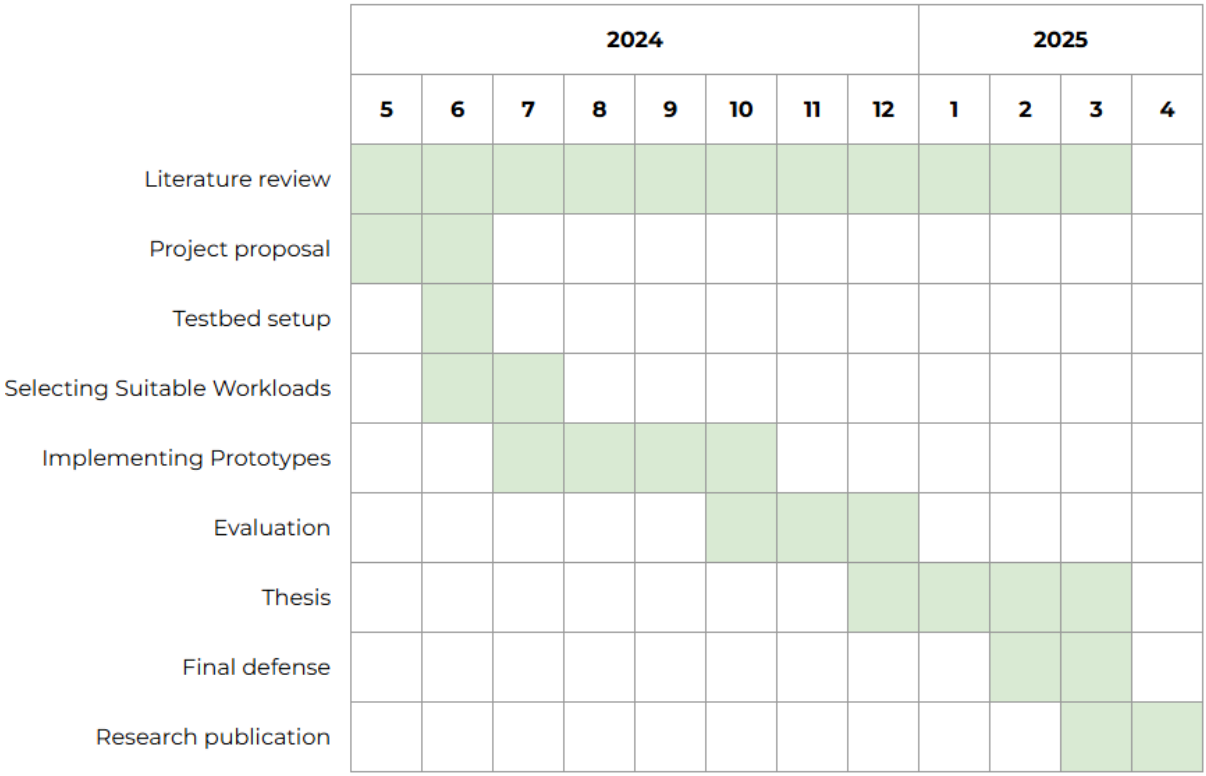

Figure 15: Project Timeline

References

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. & Warfield, A. (2003), Xen and the art of virtualization, Vol. 37.

Bellard, F. (2005), Qemu, a fast and portable dynamic translator.

Benchbase (2024), 'Benchbase: Benchbase (formerly oltpbench) is a multi-dbms sql benchmarking framework via jdbc.'. https://github.com/cmu-db/benchbase.

Bhardwaj, A. & Krishna, C. R. (2019), Impact of factors affecting pre-copy virtual machine migration technique for cloud computing, Vol. 18.

Deshpande, U., Wang, X. & Gopalan, K. (2011), Live gang migration of virtual machines.

Elsaid, M. E., Abbas, H. M. & Meinel, C. (2022), 'Virtual machines pre-copy live migration cost modeling and prediction: a survey', *Distributed and Parallel Databases* **40**.

Fernando, D., Bagdi, H., Hu, Y., Yang, P., Gopalan, K., Kamhoua, C. & Kwiat, K. (2016), Quick eviction of virtual machines through proactive live snapshots?

Hines, M. R., Deshpande, U. & Gopalan, K. (2009), Post-copy live migration of virtual machines, Vol. 43, pp. 14–26.

Ibrahim, K. Z., Hofmeyr, S., Iancu, C. & Roman, E. (2011), Optimized pre-copy live migration for memory intensive applications.

Jin, H., Li, D., Wu, S., Shi, X. & Pan, X. (2009), Live virtual machine migration with adaptive memory compression.

Jul, E., Warfield, A., Clark, C., Fraser, K., Hand, S., Hansen, J. G., Limpach, C. & Pratt, I. (2005), Live migration of virtual machines.
URL: https://www.researchgate.net/publication/220831959

Kozuch, M. & Satyanarayanan, M. (2002), Internet suspend/resume.

KVM (2024), 'Kvm'. https://linux-kvm.org/page/Main_Page.

Lepak, K. M. & Lipasti, M. H. (2000*a*), 'On the value locality of store instructions', *ACM SIGARCH Computer Architecture News* **28**, 182–191.

Lepak, K. M. & Lipasti, M. H. (2000*b*), Silent stores for free, IEEE, pp. 22–31.

Li, C., Feng, D., Hua, Y. & Qin, L. (2019), 'Efficient live virtual machine migration for memory write-intensive workloads', *Future Generation Computer Systems* **95**.

MD5 (2024), 'Md5'. https://www.geeksforgeeks.org/sha-1-hash-in-java/.

Memcached (2024), 'Memcached: Free & open source, high-performance, distributed memory object caching system.'. https://memcached.org/.

Miloǰičić, D. S., Douglis, F., Paindaveine, Y., Wheeler, R. & Zhou, S. (2000), 'Process migration', *ACM Computing Surveys* **32**.

Molina, C., Gonzàlez, A. & Tubella, J. (1999), Reducing memory traffic via redundant store instructions, Vol. 1593.

Murmur3 (2022), 'What is murmur? and what is murmur3a and murmer3f?'. https://greenrobot.org/what-is-murmur-and-what-is-murmur-3a-and-murmer3f/.

Nathan, S., Bellur, U. & Kulkarni, P. (2016), On selecting the right optimizations for virtual machine migration.

Piao, G., Oh, Y., Sung, B. & Park, C. (2014), Efficient pre-copy live migration with memory compaction and adaptive vm downtime control.

QEMU (2024), 'Qemu'. https://www.qemu.org/.

Rayaprolu, A. (2024), 'How Many Companies Use Cloud Computing in 2024? All You Need To Know'.
URL: https://techjury.net/blog/how-many-companies-use-cloud-computing/

Sahni, S. & Varma, V. (2012), A hybrid approach to live migration of virtual machines.

Sapuntzakis, C. R., Chandra, R., Pfaff, B., Lain, M. S., Rosenblum, M. & Chow, J. (2002), Optimizing the migration of virtual computers, Vol. 36.

Scott, P. (2024), 'C port of murmur3 hash'. https://github.com/PeterScott/murmur3.

SHA1 (2024), 'Sha1'. https://www.geeksforgeeks.org/sha-1-hash-in-java/.

Sharma, S. & Chawla, M. (2016), 'A three phase optimization method for precopy based vm live migration', *SpringerPlus* **5**.

Shribman, A. & Hudzia, B. (2013), Pre-copy and post-copy vm live migration for memory intensive applications, Vol. 7640 LNCS.

Svärd, P., Hudzia, B., Tordsson, J. & Elmroth, E. (2011), 'Evaluation of delta compression techniques for efficient live migration of large virtual machines', *ACM SIGPLAN Notices* **46**, 111–120.

Sysbench (2024), 'How to benchmark your system (cpu, file io, mysql) with sysbench'. https://www.howtoforge.com/how-to-benchmark-your-system-cpu-file-io-mysql-with-sysbench.

Wood, T., Ramakrishnan, K. K., Shenoy, P., Merwe, J. V. D., Hwang, J., Liu, G. & Chaufournier, L. (2015), 'Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines', *IEEE/ACM Transactions on Networking* **23**.

XEN (2024), 'Xen'. https://xenproject.org/.

Yangyang, G. (2022), 'What is crc cyclic redundancy check?'. https://info.support.huawei.com/info-finder/encyclopedia/en/CRC.html.

Zayas, E. R. (1987), Attacking the process migration bottleneck.

Zhang, X., Huo, Z., Ma, J. & Meng, D. (2010), Exploiting data deduplication to accelerate live virtual machine migration.

Zhu, L., Chen, J., He, Q., Huang, D. & Wu, S. (2013), 'Lncs 8147 - itc-lm: A smart iteration-termination criterion based live virtual machine migration'.

Index