Redundant Memory Page Transfers in Live Migration

Literature Review
SCS 3216 : Research Methods

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

February 16, 2024

Supervisor: Dr. Dinuni K Fernando

Declaration

The literature review 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 literature review is based on the work of Mr. P.L.S.R. Cooray under my supervision. The literature review has been prepared according to the format stipulated and is of an acceptable standard.

Supervisor Name : Dr. D.K.Fernando

Signature & Date

Acknowledgements

My heartfelt appreciation goes to Dr. Dinuni Fernando, whose insightful mentorship and domain knowledge steered me through this literature review. Her unwavering support and constructive feedback were invaluable in ensuring its depth and direction. I am also grateful to my esteemed colleagues at UCSC for fostering an environment of collaborative learning and sharing valuable perspectives. Finally, I express my gratitude to my family, whose constant love and understanding provided the essential foundation for this endeavor. Their unwavering support and encouragement fuelled my resolve throughout this journey.

Abstract

Migrating a VM to a destination host from a source host is called Live VM Migration. Cloud Data Centers (CDCs) use Live VM Migration to provide uninterrupted service during hardware or software failures, planned maintenance, and upgrades. At present, Live VM Migration can be categorized based on the method of transferring the memory content. Pre-Copy Migration is one of those migration techniques which is robust and simple. Although Pre-Copy Migration is a powerful migration mechanism when migrating a VM executing memory-intensive workload Total Migration Time becomes unpredictable which is a significant disadvantage. This is because the iteration count required to transfer memory pages cannot be precisely estimated due to varying dirty page rates and network transmission speeds. Therefore, in extreme cases, excessive dirty page rates and low network bandwidth can lead to excessively prolonged TMT or even outright failure. As mentioned above Rapid Dirty Page Generation is a significant problem in Pre-Copy Migration. However, most of these generated dirty pages are not modified, but identical to their existing copies stored in the destination host, and these pages are called “Fake Dirty Page”.

This report discusses the Fake Dirty Problem, how it occurs in default pre-copy migration, solutions proposed in existing literature, issues in those solutions, and possible research directions.

Contents

List of Figures

List of Tables

Acronyms

AWS Amazon Web Services. 1

CC Cloud Computing. 1, 19

CDC Data Center. 1, 2

CDCs Cloud Data Centers. iii, 1, 19

DT Downtime. 2, 4, 7, 9, 11, 14

FDP Fake Dirty Page. iii, iv, 3, 14–16, 19

FDPS Fake Dirty Pages. 3, 10, 12, 14, 16, 19

GCP Google Cloud Platform. 1

HCM Hybrid Migration. iv, v, 2, 8, 19

PosC Post-Copy. 2, 8

PosCM Post-Copy Migration. iv, v, 2, 7, 8, 19

PreC Pre-Copy. v, 2, 6, 8, 15

PreCM Pre-Copy Migration. iii–v, 2, 3, 5–15, 19

TMT Total Migration Time. iii, iv, 2–4, 9–11, 14, 19

VM Virtual Machine. iii, iv, 1–10, 15, 17, 19, 23

VMMs Virtual Machine Monitors. 1

VMs Virtual Machines. 1, 3, 9, 19

XBZRLE XOR Binary Zero Run Length Encoding. 10

1 Introduction

1.1 Cloud Computing

In the modern world, high processing power, and storage have become very demanding due to the advancement in technology, making Cloud Computing (CC) an essential technology. This is because CC provides software, hardware, storage, security, and infrastructure services efficiently and reliably, increasing customer satisfaction. Platforms like Microsoft Azure, Amazon Web Services (AWS), Alibaba Cloud, Google Cloud Platform (GCP) are some of the CC service providers at present. With the efficiency of cloud services, cloud resource demand has also increased. These cloud resources are modeled as software, platform, and infrastructure and provided by Cloud Data Centers (CDCs). CDCs maintain many server computers with high-speed network connection as they should provide uninterrupted services to users. This is because they provide these services as time-based paid plans. According to the study by Uddin et al. (2013), the servers in CDCs consume a massive amount of electricity where more than 30% of servers consume energy while idle. As a solution, CDCs are attracted to Virtualization (Jul et al. 2005) as OS instances are allowed to concurrently run on a physical machine that provides individual OS isolation, efficient physical resource usage, and high performance (Barham et al. 2003).

1.2 Virtualization

Division of a single computer's processor, memory, and storage components into multiple Virtual Machines (VMs) with a Hypervisor is known as **Virtualization** . Hypervisors, also known as Virtual Machine Monitors (VMMs). As mentioned above, virtualization is an essential technique in CC since it provides security isolation, heterogeneous hardware abstraction, and easy management (Barham et al. 2003). Apart from that, Live VM Migration is a main benefit of virtualization (Jul et al. 2005, Nelson et al. 2005).

1.3 Need of Live VM Migration in Cloud Computing

CDCs host multiple VMs in one server. Therefore, it can lead to over-utilizing physical resources due to the dynamic demands of interactive cloud applications and performance degradation. Also, servers in the CDC might fail due to hardware or software failures, affecting the interruptions to the services provided by the VMs hosted in these servers. To address these issues, VM migration between physical servers is used as CDCs should provide continued uninterrupted services.

Migration of VMs originated from Process Migration (Osman et al. 2002) but has residual dependencies that prevent its use in cloud environments. VMs should be allowed to migrate from one server to another within the CDC or between two CDC. Live VM Migration, a variant of this mechanism provides services seamlessly without breaking the connection to end users, efficient resource utilization, and

avoiding Service Level Agreements violations.

1.4 Live VM Migration

Migrating a VM to a destination host machine from a source host machine is called Virtual Machine (VM) Migration. This procedure of migration can be categorized into two categories, Live Migration and Non-Live Migration. When considering Non-Live Migration, also called Process Migration, has multiple problems, such as the "residual dependency" problem; the original host must be available for access after Migration and interruption to provided services during migration (Milojičić et al. 2000). Live VM Migration resolves these problems, and therefore, the original host machine does not have to be available for access, clients are not required to reconnect, and VM operations are not to be concerned of (Jul et al. 2005). And also, the use of power management, online system maintenance, fault tolerance(Li et al. 2019), and load balancing(Kozuch & Satyanarayanan 2002) techniques in CDC's have increased the robustness of Live VM Migration. Minimizing the Downtime and the Total Migration Time is the main concern of Live VM Migration.

Transferring of a VM is categorized into three phases: Pull, Stop-and-Copy, and Push. Based on these phases Live VM migration is categorized into three. They are: Post-Copy Migration, Pre-Copy Migration, and Hybrid Migration.

Pre-Copy Migration initially transfers every memory content, followed by the iterative copying of dirty pages and finally transferring the CPU state into the destination host from the source host. Some Pre-Copy Migration techniques used are dynamic rate-limiting, data compression, migration control method, delta compression, memory compaction technique, adaptive VM downtime control, deduplication, three-phase optimization, and page content reduction technique. Post-Copy Migration is another technique introduced by Hines et al. (2009), which initially transfers the CPU states to the destination host by stopping the VM in the source host. Post-Copy Migration variants include Active Push, Self Ballooning, Pre-Paging, and On-Demand Paging. Although the problem of prolonged TMT for memory-intensive workload in Pre-Copy Migration is resolved by Post-Copy Migration, it inherently contains two weaknesses: Less Robust and Performance degradation an application which runs in the VM which is migrated. Hybrid Migration allows memory-intensive workloads to be handled without migration failures by combining migration techniques Post-Copy and Pre-Copy.

1.5 Problems in Pre-Copy Migration

The efficiency and the performance of a Live VM Migration technique is evaluated using attributes such as Network Traffic, Downtime (DT), Total Migration Time (TMT), Service Degradation, and Network bandwidth utilization. When considering Pre-Copy Migration, TMT is regarded as the primary bottleneck due to its unpredictability. Two significant factors directly impacting the TMT and DT are the dirty page generation rate and VM size. The TMT 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 TMT in Pre-Copy Migration, are Rapid Dirty Page Generation, Page Level Content Redundancy, Lower Network Transfering 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.

1.6 Fake Dirty Problem

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, wasting network bandwidth and migration time. 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.

The primary reason for Fake Dirty Page generation is the issuing of silent store instructions that result in "write-not-dirty" requests. These requests are memory pages marked as dirty but without actual content change. The second reason responsible for Fake Dirty Page generation is the migration thread's tracking mechanism.

In this Literature Review, we mainly focus on and analyze this Fake Dirty Problem, deeply exploring reasons for generating Fake Dirty Pages, existing solutions, and issues in those solutions.

2 Background - Live VM Migration

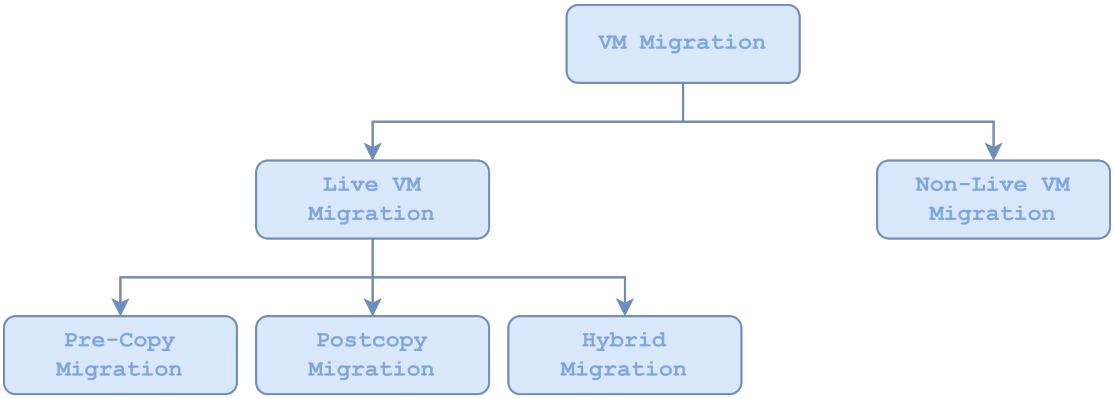

Figure 1: Classification of VM Migration

Migrating a VM to a destination host machine from a source host machine is called Virtual Machine (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. In Live Migration unlike Non-Live Migration, before starting the migration procedure the VM is not stopped, where it provides 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 down-time (Zhu et al. 2013), efficiently utilize the network bandwidth (Svärd et al. 2011), and decreasing the performance degradation (Ibrahim et al. 2011).

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)

2.1.1 Pre-Copy Migration

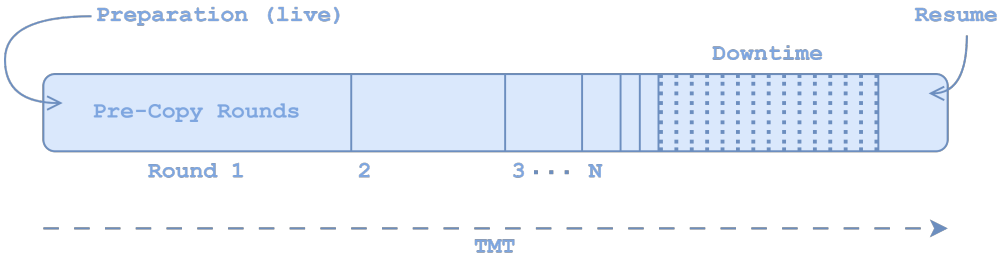

Figure 2: Pre-Copy Migration Timeline (Hines et al. 2009).

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.

2.1.1.1 Pre-Copy Migration Stages

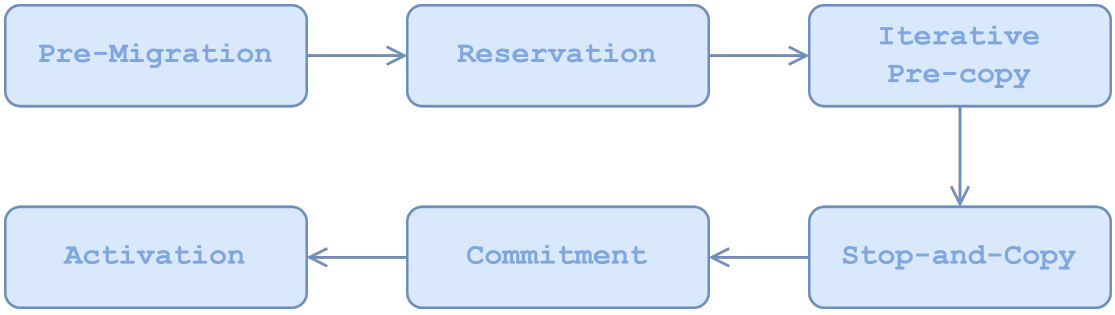

Figure 3: Stages in Pre-Copy VM Migration (Jul et al. 2005).

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 an acknowledgment to this message is sent by the source. The source discards the VM.
6. **Activation:** The destination host becomes the primary host and runs a post-migration code.

2.1.1.2 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*a*)

2.1.2 Post-Copy Migration

Pre-Copy Migration effectively minimizes the Application 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.

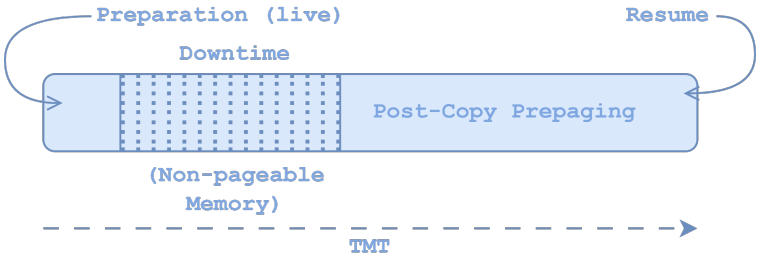

Figure 4: Post-Copy Migration Timeline (Hines et al. 2009).

2.1.2.1 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)

2.1.2.2 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.

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).

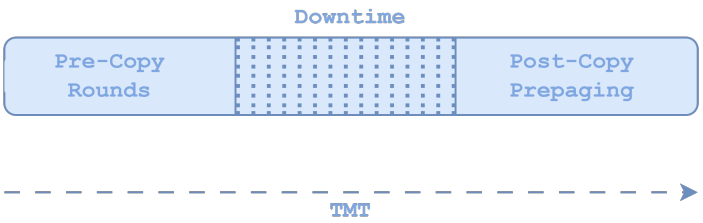

Figure 5: Hybrid Migration Timeline

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.

3 Problems in Pre-Copy Migration

There are multiple attributes used in Live VM Migration to evaluate the efficiency and the performance of a technique. These attributes are,

1. **Total Migration Time (TMT)** Time Duration from starting and completion of a migration.
2. **Downtime (DT)** Time Duration from suspending the VM on the source host and successfully starting it on the destination host.
3. **Service Degradation** Impact on the services executing in a VM due migration.
4. **Network Traffic** Total Data Flow through Migration.
5. **Network bandwidth utilization** Combined measure of *TMT* and *Network Traffic*

When considering Pre-Copy Migration technique, Previous studies, and experiments have consistently identified TMT as the primary bottleneck in it. Additionally, performance overhead is also considerable.

This section explores more deeply on the challenges in Pre-Copy Migration mentioned previously.

3.1 Unpredictable Total Migration Time In Pre-Copy Migration

As stated by Elsaid et al. (2022) the unpredictability of TMT is a significant disadvantage of Pre-Copy Migration. The iteration count required to transfer memory pages cannot be precisely estimated due to varying dirty page rates and network transmission speeds. In extreme cases, excessive dirty page rates and low network bandwidth can lead to excessively prolonged TMT or even outright failure.

According to the experiments carried out by Bhardwaj & Krishna (2019*a*) the two major factors that directly impact TMT and Downtime are:

1. **VM Size:** Larger VMs necessitates the transfer of more memory pages, inherently extending the migration process.
2. **Dirty Page Rate:** A higher frequency of memory page modifications directly translates to prolonged Total Migration Time.

Further more Piao et al. (2014) state that the TMT of Pre-Copy Migration, depends in two distinct phases:

1. **Transferring Entire Memory Pages:** This initial phase involves copying the entire memory of the VM to the destination host.

2. **Transferring of Newly Modified Pages in Iterations:** This iterative phase focuses on repeatedly transferring memory pages modified since the previous iteration.

The time allocation between these phases hinges heavily on the VM's memory behavior. Relatively static memory content leads to more time spent in the initial transfer phase, while frequent memory updates shift the focus towards the iterative phase. Therefore, a critical weakness within the iterative phase is that if the volume of dirty pages identified during iterations remains excessively large, the migration process may become trapped in a perpetual loop of transferring dirty pages, preventing the completion and potentially leading to failure.

According to the above-mentioned details, the TMT depends on the *push-phase* in Pre-Copy Migration. Problems that directly affect the time consumption in *push-phase* resulting in higher TMT in Pre-Copy Migration are Rapid Dirty Page Generation, Page Level Content Redundancy, Lower Network Transfering Rates, and Fake Dirty Pages.

3.1.1 Rapid Dirty Page Generation Problem

Pre-Copy Migration faces a significant challenge when dealing with memory-intensive applications that generate dirty pages at a rate exceeding the network bandwidth available for transferring them. This phenomenon, known as the **Rapid Dirty page Generation Problem**, can lead to several bad consequences:

1. **Prolonged TMT:** Palanivel (2015) and Sharma & Chawla (2016) emphasize the negative impact on migration time when the page dirtying rate surpasses the transfer rate. This results in prolonged TMT and an increase in data migration, potentially missing the chance of optimal migration.
2. **Migration Failure:** Shribman & Hudzia (2013) stated that the migration procedure might be completely delayed if the dirty pages can't be transferred faster than the dirty page generation rate, which will eventually lead to migration failure.
3. **Missed Opportunities:** According to Li et al. (2019), prolonged TMT due to rapid dirty page generation may result in losing the chance for efficient migration, which could have an impact on system performance and service continuity.

Shribman & Hudzia (2013) have proposed two solutions to address this challenge:

1. **Page Reordering Policies:** Shribman & Hudzia (2013) recommend prioritizing Less Recently Used (LRU) pages for migration, as they are statistically less likely to be modified again, reducing redundant transfers.
2. **Data Compression Techniques:** Shribman & Hudzia (2013) also propose using XOR Binary Zero Run Length Encoding (XBZRLE) to compress dirty

pages, minimizing overall data transfer amount and potentially mitigating the bandwidth bottleneck.

Addressing the *Rapid Dirty page Generation Problem* remains a crucial area of research in optimizing pre-copy live migration performance and ensuring reliable VM relocation. Further research could explore adaptive techniques and intelligent page selection algorithms to dynamically adjust and optimize migration based on workload characteristics and network conditions.

3.1.2 Page Level Content Redundancy Problem

Within the Pre-Copy Migration's *push-phase*, a phenomenon known as **Page Level Content Redundancy** can significantly slow performance.(Bhardwaj & Krishna 2019*b*).

The memory pages are repeatedly transferred to the destination from the source during the iterations in the *push-phase*, as the pages become dirty. However, only a tiny portion of a page's content might be modified for memory-intensive workloads between iterations. Despite these minimal modifications, the traditional Pre-Copy Migration approach transfers the entire page, even if only a fraction of their content has changed. This results in redundant data transfer, leading to several drawbacks:

- Prolonged Downtime
- Prolonged Total Migration Time
- Excessive Network Traffic

To address this challenge, Bhardwaj & Krishna (2019*b*) have proposed the following solution.

Page Content Reduction Technique: Without transferring the complete page, the logical XOR operator is applied to the old and new page versions, resulting in a differential page and then sent to the destination after encoding. This encoded page is decoded at the destination, and the original page is obtained again by applying the XOR operator to the decoded page. Then, the page is stored in the destination's cache.

3.1.3 Lower Network Transmission Rates

The available network bandwidth size is another factor affecting the prolonged TMT. As stated in section 3.1.1, most of the time, the available network bandwidth size can't accommodate the excessive page dirtying rates. Also, transferring the *writable-working set* can be substantial, potentially causing noticeable downtime.(Sharma & Chawla 2016).

Jul et al. (2005) proposed **Dynamic Rate-Limiting** a solution to handle the bandwidth utilization. First, the administrator establishes minimum and maximum bandwidth thresholds, providing flexibility and control over the migration

process. Then, the first round of data transfer commences at the minimum bandwidth, ensuring a cautious start that avoids overwhelming the network. Subsequent rounds monitor the rate of page modifications (dirtied pages per time unit) and dynamically adjust the bandwidth for the next round. This reactive approach ensures responsiveness to changing conditions. An empirically determined increment value is added to the previous round's dirtying rate to calculate the bandwidth for the upcoming round. Pre-copying concludes when the calculated bandwidth exceeds the administrator's maximum limit or the remaining data falls below a specified threshold. The bandwidth is maximized to expedite the transfer and minimize service downtime in the *stop-and-copy phase*.

This adaptive bandwidth management strategy demonstrates the potential for optimizing migration efficiency and minimizing service disruptions within diverse network environments.

3.1.4 Fake Dirty Problem

The recent work of Nathan et al. (2016) discovered a previously unidentified phenomenon: the presence of **Fake Dirty Pages (FDPS)** 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. This unnecessary duplication wastes network bandwidth and migration time.

Exploring the root of this inefficiency, Li et al. (2019) identified two reasons:

1. **Silent Store Instructions:** Certain memory management operations, specifically "write-not-dirty" requests triggered by silent store instructions, can flag pages as dirty despite no modifications to the content. These incorrectly marked pages are then unnecessarily migrated.
2. **Defective Dirty Page Tracking:** The mechanism for tracking dirty pages may be flawed, rendering system maintenance activities and load balancing efforts less effective. This can lead to inaccurate dirty page identification and subsequent unnecessary transmission.

Details on Fake Dirty Pages are discussed in section 4

3.2 Energy Consumption Concerns

Performance Concerns

1. **Service Degradation:** During migration, the service hosted on the VM might experience performance degradation due to resource competition and data transfer overhead. (Rybina & Schill 2017)

Energy Concerns

1. **Increased CPU Activity:** The migration process demands additional CPU cycles for managing the transfer and memory updates, leading to higher energy consumption. (Rybina & Schill 2017, Strunk & Dargie 2013)

2. **VM Size and Network Bandwidth:** Larger VM sizes lead to greater energy consumption during migration, while wider network bandwidth can help reduce it.(Strunk & Dargie 2013)
3. **Resource Consumption on Hosts:** Both source and destination hosts experience increased CPU and I/O activity during iterative memory copying, impacting the overall migration time and energy consumption. (Wu & Zhao 2011)
4. **State Transfer and I/O Bandwidth:** Data transfer rate through the network affects I/O bandwidth usage and, consequently, energy consumption. Minimizing the state transfer can lessen this impact. (Wu & Zhao 2011)

Therefore, the energy consumption of Pre-Copy Migration should not be neglected.

Workloads	Description
Memcached	Memcached is an in-memory key-value store storing arbitrary data returned by a benchmark called Memaslap.
Zeus-MP	A floating point benchmark, which performs astrophysical fluid dynamics simulation.
mcf	An integer benchmark, which stimulates schedules for public transport using a network simplex algorithm.
bzip2	An integer benchmark, which performs single file compression.
cactusADM	A floating point benchmark, which stimulates numerical relativity.
milc	A floating point benchmark, which stimulates numerical application in quantum chromodynamics.
LBM	A floating point benchmark, which stimulates fluid using computational fluid dynamics.
kernel compilation	A system-call-intensive workload, which is expensive for virtualization.

Table 1: Memory-intensive workloads used by Li et al. (2019)

4 Critical Analysis - Fake Dirty Problem

As mentioned in the section 3.1.1, *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 such as bzip2, Memcached, Zeus-MP, and mcf as described in Table 1. According to the experiments, high amounts of are generated by Fake Dirty Pages bzip2, mcf, Zeus-MP, and Memcached workloads. The remaining workloads also have generated a significant amount of Fake Dirty Pages. Therefore, Li et al. (2019) have stated that by avoiding Fake Dirty Pages transfer while during Pre-Copy Migration, the DT and TMT can be reduced.

4.1 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.

4.1.1 "Write-not-dirty" Pages

Memory pages that are marked as dirty but do not have actual content change are referred to as **Write-not-dirty** pages(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 TMT (Li et al. 2019).

4.1.2 Defective Dirty Page Tracking

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

4.1.2.1 KVM

The virtualization platform KVM/QEMU 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.

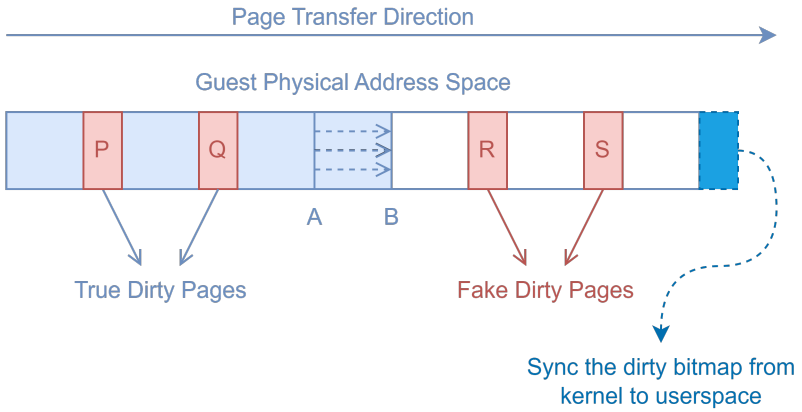

Figure 6: Tracking Mechanism of Dirty Pages in KVM (Li et al. 2019).

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. But the 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.

4.1.2.2 XEN

XEN 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, let's explore the proposed solutions to avoid the transmission of Fake Dirty Pages (FDPS) during migration.

4.2 Exsisting Solutions

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 solutions 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 it is 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 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.

4.3 Issues in the Existing Solutions

The experiments by Li et al. (2019) point out that delta compression and deduplication optimizations are inefficient methods of reducing the transfer of Fake Dirty Page. They are:

1. **Memory overhead in Delta Compression.**

   To find identical pages and similar content, a HashSimilarityDetector(Gupta et al. 2019) along with XBZRLE is used in *delta compression*. To perform this, the old version of the pages should be cached in the memory, leading to memory overhead.

2. **Ineffective Performance Improvement**

   In a single VM, less than 10% of duplicate pages are generated when running the abovementioned memory-intensive workloads. Therefore, only a tiny performance improvement is obtained using *deduplication*.

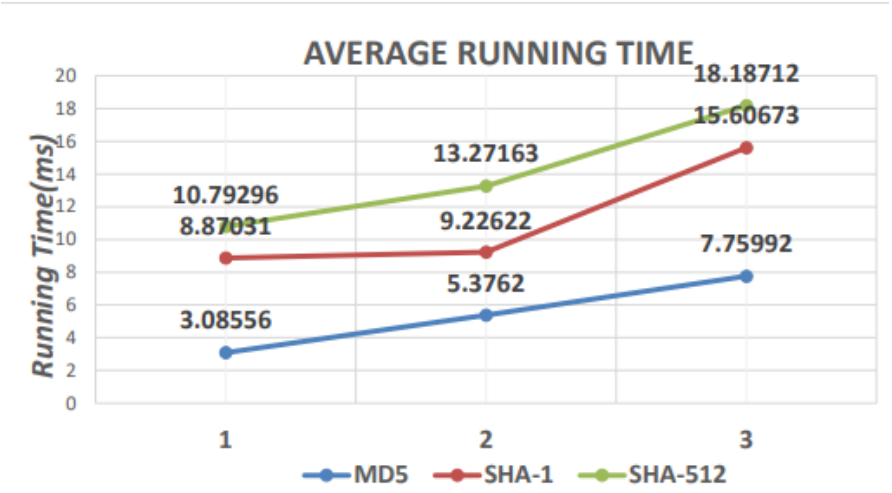

Figure 7: Comparison of average runtime of MD5 and SHA 1 and SHA512 (Long 2019)

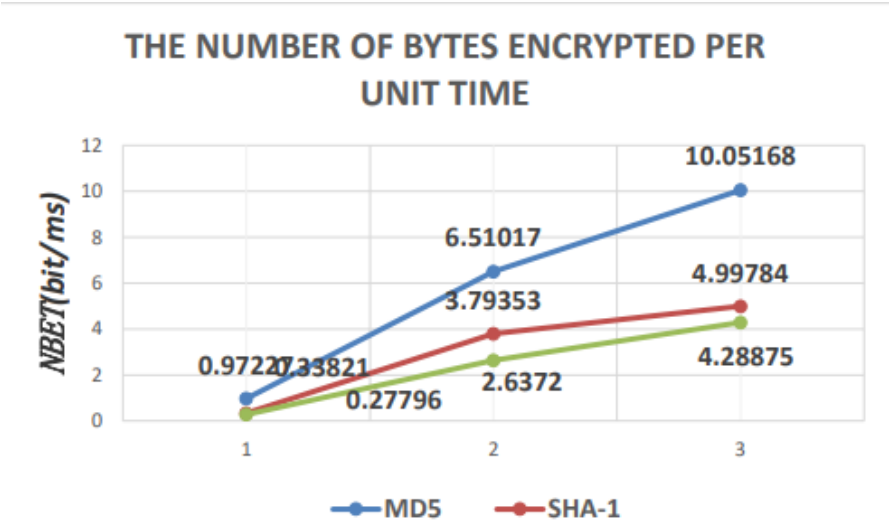

Figure 8: Comparison of NBET of MD5 and SHA 1 and SHA512 (Long 2019)

	MD5	SHA1	SHA512
Length of the hash (bits)	128	160	512
Time Complexity	$\theta(N)$	$\theta(N)$	$\theta(N)$
Run Time (Relatively)	Fast	Medium	Slow
NBET (Relatively)	High	Fast	Medium
Security (Relatively)	Weak	Medium	Strong

Table 2: Comparison of MD5 and SHA1 and SHA512 (Long 2019)

As mentioned in the section 4.2, Li et al. (2019) have used SHA1 secure hashes to compute the hash of a page. However, using the MD5 checksum to achieve this goal would be more optimal because we don't have to focus on security concerns but on computing the hash of a page quickly and efficiently. The results of the experiments conducted by Gupta & Kumar (2014) and Long (2019) support this above claim. Figures 7 and 8 show the results obtained by Long (2019). According to the results, MD5 is faster than SHA1.

5 Conclusion and Future Work

Cloud Computing has become an essential technology providing users cloud services such as software, platform, and infrastructure through Cloud Data Centers. CDCs use virtualization to reduce inefficiency and wasteful energy consumption when using multiple servers by replacing these servers with multiple VMs in a single server. VM migration mechanisms are used by CDCs to provide uninterrupted service during hardware or software failures, planned maintenance, and upgrades. Live VM Migration is preferred by CDCs because of the problems such as "residual dependency" available in Non-Live Migration (Process Migration).

When considering Live VM Migration, the VM remains operational, where it provides uninterrupted services to end-users throughout the migration. There are three main approaches when considering the Live VM Migration techniques and each of them differs according to the phases they use to transfer memory. Pre-Copy Migration: Contains a push-phase and stop-and-copy phase, Post-Copy Migration: Contains stop-and-copy and pull phase and Hybrid Migration: Combines a short push-phase with a stop-and-copy and pull phases. Furthermore, I discussed several optimizations and weaknesses of each Live VM Migration technique. Further, I discussed multiple problems in Pre-Copy Migration that affect the prolonged TMT and migration failures. The identified problems are Rapid Dirty Page Generation, Page Level Content Redundancy, Lower Network Transferring Rates, and Fake Dirty Pages. Other than that, I discussed some of the performance concerns and energy consumption concerns that arise while doing Pre-Copy Migration

This Literature Review mainly focused on the **Redundant Memory Page Transfers in Live Migration**, which means the Fake Dirty Page generation in Pre-Copy Migration. When considering current literature, only Nathan et al. (2016) and Li et al. (2019) have examined this problem of generating Fake Dirty Pages. And, there are only few solutions that address this problem. This review further discussed on optimizations that can be applied to the proposed solution of Li et al. (2019) by applying MD5 secure hashes instead of SHA1 secure hashes. Further research on detecting and minimizing silent store instructions in VM, and optimizations to dirty page tracking mechanism to minimize TMT can be conducted to solving this Fake Dirty Problem.

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.

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

Bhardwaj, A. & Krishna, C. R. (2019*b*), 'Improving the performance of pre-copy virtual machine migration technique'.

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**.

Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., Voelker, G. M. & Vahdat, A. (2019), Difference engine: Harnessing memory redundancy in virtual machines.

Gupta, P. & Kumar, S. (2014), 'A comparative analysis of sha and md5 algorithm', *International Journal of Computer Science and Information Technologies* **5**.

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.

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**.

Long, S. (2019), A comparative analysis of the application of hashing encryption algorithms for md5, sha-1, and sha-512, Vol. 1314.

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.

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

Nelson, M., Lim, B. H. & Hutchins, G. (2005), Fast transparent migration for virtual machines.

Osman, S., Subhraveti, D., Su, G. & Nieh, J. (2002), The design and implementation of zap: A system for migrating computing environments, Vol. 36.

Palanivel, K. (2015), 'A survey of virtual machine placement algorithms in cloud computing environment', *Computer Engineering and Intelligent Systems* **6**.

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

Rybina, K. & Schill, A. (2017), Estimating energy consumption during live migration of virtual machines.

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.

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.

Strunk, A. & Dargie, W. (2013), Does live migration of virtual machines cost energy?

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.

Uddin, M., Shah, A., Alsaqour, R. & Memon, J. (2013), 'Measuring efficiency of tier level data centers to implement green energy efficient data centers', *Middle East Journal of Scientific Research* **15**.

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**.

Wu, Y. & Zhao, M. (2011), Performance modeling of virtual machine live migration.

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