

Enhancing System Checkpoints for Seamless Fault-Tolerant Live VM Migration

Interim Report
SCS 4224 : Final Year Project

H. K. P. S. Fernando
Student of University of Colombo School of Computing
Email: 2020cs051@stu.ucsc.cmb.ac.lk

November 10, 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 : H. K. P. S. Fernando

Registration Number : 2020/CS/051

Index Number : 20000512

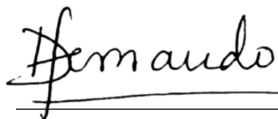


10.11.2024

Signature & Date

This is to certify that this interim report is based on the work of Mr. H. K. P. S. Fernando 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



10-11-2024

Signature & Date

Contents

1	Introduction and Background	1
1.1	Introduction	1
1.2	Motivation	1
1.3	Background	2
1.3.1	Live VM migration vs Non-live migration	2
1.3.2	Live Migration Techniques	2
1.3.3	Pre-copy Migration Technique	3
1.3.4	Post-copy Migration Technique	4
1.3.5	Hybrid Migration Technique	4
1.3.6	Checkpointing in VM migration	5
2	Literature Review and Research Criteria	6
2.1	Literature Review	6
2.1.1	Checkpointing Techniques	6
2.2	Research Gap	9
2.3	Research Questions	10
2.4	Aims and Objectives	10
2.4.1	Aim	10
2.4.2	Objectives	10
2.5	Scope	11
2.5.1	In Scope	11
2.6	Significance of the Research	12
3	Methodology and Design	13
3.1	Research Approach and Methodology	13
3.2	Research Design	15
3.2.1	Identifying the modified page content instead of transmitting the entire page	15
3.2.2	Ability to use checkpoint compression techniques to reduce the content to be transmitted	17
3.3	Preliminary Results and Discussion	17
3.3.1	Byte-level Granularity Mechanism	17
3.3.2	Compression Algorithm Integration	20
4	Tools, Evaluation, and Project Timeline	21
4.1	Research Tools Used	21
4.1.1	QEMU	21
4.1.2	RealVNC Viewer	21
4.1.3	NFS	22
4.1.4	RedisInsight	22
4.2	Evaluation	22
4.2.1	Evaluation of the accuracy of the implemented methodologies	22
4.2.2	Whole system evaluation	22
4.3	Project Timeline	23

List of Figures

1	Pre-copy migration stages (Agarwal & Raina 2012)	3
2	Hybrid migration phases (Choudhary et al. 2017)	5
3	Incremental checkpointing(Fernando et al. 2019)	9
4	Proposed improvement(Fernando et al. 2019)	12
5	TestBed Setup	13
6	Byte Level Granularity Approach	16
7	Page content reduction in byte-level granularity approach	20
8	Page content reduction with compression types	21
9	Project Timeline	23

List of Tables

1	Byte-level granular methodology results	19
2	Comparison of page content transmission with compression techniques and baseline Fernando et al. (2019) model across workloads .	20
3	Comparison of page content reduction percentage compared to baseline Fernando et al. (2019) model	21

Acronyms

AUFS Another Union File System. 6

CC Cloud Computing. 2, 12

CDC Cloud Data Center. 1, 2, 10, 12

COW Copy On Write. 5, 7

FIC Forward Incremental Checkpointing. 8

IaaS Infrastructure as a Service. 1

NVM Non Volatile Memory. 7

OS Operating System. 13

PaaS Platform as a Service. 1

RIC Reverse Incremental Checkpointing. 8

SaaS Software as a Service. 1

ULEB128 Unsigned Little Endian Base 128. 18

VM Virtual Machine. ii, 1–3, 5–10, 12, 13, 15

1 Introduction and Background

1.1 Introduction

Virtual Machine (VM) migration is a key factor in the field of Cloud Computing, where the VMs running on physical nodes are transferred from one source node to a destination node in the instance of failures, maintenance needs, consolidation purposes etc. Live VM migration speaks out as the most widely used migration technique where the VM is transferred from the source to the destination in a lively manner while the VM and its applications remain still usable with seamless experience from the perspective of the VM user during the period of migration (Agarwal & Raina 2012). During this procedure, the VMs should be preserved while data, and states of the VM should be maintained as it is.

During this live migration procedure, the running VM possess a risk of getting corrupted or lost in the event of the failures that could occur at the source node, destination node, or even the network connecting them within the period of migration (Fernando et al. 2019). Such a loss of the VM can cause severe harm to the respective users and the cloud service providers will be liable for such losses. To overcome this issue, VM checkpoints are taken and stored prior and during migrations, to be used in cases of VM losses and failures to recover them which will reduce the replication duration during migrations. Different types of checkpointing mechanisms are used in the migration models available. In parallel to implementing safe and secure checkpointing mechanisms, those mechanisms should have a minimal impact on the application performance, migration time, downtime, and other concerns of the running VM. Additionally, the network bandwidth usage and the costs incurred for such mechanisms should be bearable. Different checkpointing mechanisms have been studied in the state of the art literature throughout the years addressing different aspects and problems associated with checkpointing which are discussed in section 2.1.1. However, there are still areas that could be looked into and optimized in the field of VM checkpointing in live VM migration.

1.2 Motivation

With the technology evolving day by day, a major portion of the things related to computing are moving towards the domain of cloud which falls under one of the three categories SaaS, IaaS or PaaS. The major names resonating when it comes to cloud service providers are Microsoft Azure Microsoft (2024), Google Cloud Platform GCP (2024), Amazon Web Services AWS (2024), Alibaba Cloud Alibaba Cloud (2024)etc., which are giants that offer services to a huge portion of the businesses and industries distributed all across the world. With this huge responsibility at hand, these Cloud Data Center (CDC)s are liable to each customer and the critical operations running within their VMs. In such an environment, with the rapid growth and the demand for Cloud Services, the operations running within them should be optimized and the resources should be utilized wisely. In the context of VM migration, in parallel to everything running in the VM residing

in the nodes, the checkpointing related operations are utilizing the resources of the same node itself. Another major fact is that the checkpoint sizes increase when the respective VM size increases, making it difficult and time consuming to replicate the checkpoints via the networks. The checkpoint sizes should be made smaller as possible to mitigate this problem. Additionally the migration time and downtime overheads, network bandwidth overheads, application performance degradation overheads etc. introduced with these checkpointing mechanisms embedded in the migration techniques still have room to be improved. With successful improvements and optimizations, the operations in the CDCs can be made much more productive and cost effective. Such a contribution will be of good value to the domain of Cloud Computing (CC), which is a rapidly growing industry in today's tech world.

1.3 Background

1.3.1 Live VM migration vs Non-live migration

VM migration refers to the transferring of the VM from one physical host to another seamlessly safeguarding the data, states of execution, device states, etc. of the residing VM. When exploring the concept of migration, there are two main domains identified (Ahmad et al. 2015). They are

- Live migration
- Non-live migration

The process of *non-live migration* as discussed in Milošević et al. (2000), is the process of migrating the VM from the source node to the destination node in an inactive state. That is when a migration needs to be done, the source node is suspended and the migration takes place. Once the migration is completed and the destination node is ready to go online, the VM execution is fully terminated from the source and resumes at the desired destination. Within the period of migration, the users won't be able to interact with the VM to obtain the services.

In contrast to non-live migration, during *live migration* the VM is continuously in a powered-up state within the whole period of migration (Clark et al. 2005). In parallel to the migration happening, the VM remains still usable from the perspective of the user. Clark et al. (2005) highlights that for a successful proper live migration, the user won't feel any noticeable delays or glitches during the execution of the tasks of the VM. This facilitates the supply of uninterrupted services to the users which is an utmost requirement in the field of CC.

1.3.2 Live Migration Techniques

There are several live migration techniques discussed and used in the world of CC. The techniques differ in the order in which the memory and the CPU states are transferred from the source-node to the destination-node and the steps carried out

in each of them. There are three main live VM migration techniques available in the state of the art literature which are namely,

1. Pre-copy Migration
2. Post-copy Migration
3. Hybrid Migration

The above-mentioned three techniques will be analyzed in detail in the next sections 1.3.3 - 1.3.5.

1.3.3 Pre-copy Migration Technique

The technique of pre-copy migration is one of the main approaches used in the context of live VM migration. If briefly emphasized, when a migration is needed, the source node continues to execute the VM while the memory content of the VM will be copied to the destination node in several iterations which at the end, the CPU states, device states, together with rest of the dirty pages remaining are sent to the destination and the VM is hand-overed to start its execution at the destination (Fernando et al. 2019). Here the iteration continues until the memory content is converged, and the transferring of the execution states of the CPU is initiated when the estimated time where the VM will be down(downtime) is becoming less than a predefined threshold value.

The pre-copy migration is structured and broken down into six steps as shown in figure 1 (Agarwal & Raina 2012), (Clark et al. 2005). A possible concern

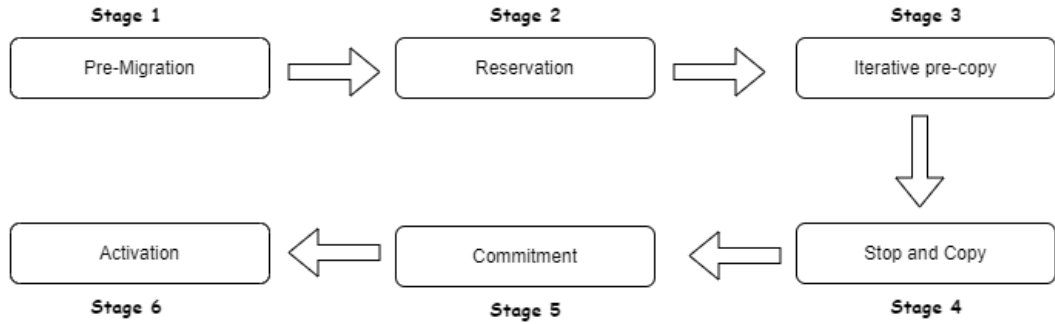


Figure 1: Pre-copy migration stages (Agarwal & Raina 2012)

is that how to decide the point at the page transmission is converged and the *stop_and_copy* phase is to be instantiated. As mentioned in Fernando et al. (2019), pre-copy migration is ideal for memory read-intensive tasks rather than write-intensive tasks. If it is a write-intensive task, the transferring of the memory pages has a tendency to never get converged as memory pages are being continuously dirtied paving the way to not reaching a level below the pre-defined threshold. The post-copy migration that will be discussed next can tackle this problem and it is discussed in section 1.3.4.

1.3.4 Post-copy Migration Technique

The approach that post-copy uses is different from the approach used in pre-copy. As proposed in Hines et al. (2009) post-copy migration uses a strategic way where the CPU states are sent to the destination node first and the VM is set to resume at the destination-node as the first step. Successively, the source memory pages are being copied to the destination-node in a series of iterations. This approach faces a problem when the memory pages which are not yet copied to the destination-node are being accessed. In such an instance, post-copy technique utilizes *on-demand paging*, where a page fault is triggered and the respective page is being requested from the source and received. Post-copy lies ahead of pre-copy in this context as a memory page is transmitted a maximum of once through the networks in post-copy, but in pre-copy, the same memory page can be transmitted multiple times which results in a higher network traffic (Hines et al. 2009). However post-copy suffers in the context of read intensive workloads as the no. of page faults will get increased with the increased number of read operations which will result in VM performance degradation due to the latencies generated for the demand-paging operations.

1.3.5 Hybrid Migration Technique

As explored in the above sections 1.3.3 and 1.3.4, both pre-copy and post-copy have some inherent pros and cons. A hybrid approach has been proposed in Sahni & Varma (2012) which is a mixed approach of both pre-copy and post-copy. As highlighted in Sahni & Varma (2012), in this hybrid approach, a subset of memory which is called “*working set*” that is frequently accessed by the VM is transferred in addition to the CPU execution states and device states as mentioned in post-copy migration. This will yield to produce a lesser number of page faults as already a most frequently accessed set of memory is available in the destination VM since the initial stages of the migration.

As shown in figure 2, hybrid migration can be mainly structured into five phases as discussed in Choudhary et al. (2017) and Damania et al. (2018). As emphasized in the diagram, the hybrid approach is a combined effort of both pre-copy and post-copy to achieve the target of ultimately reducing migration and downtimes without having noticeable overheads in the application performance.

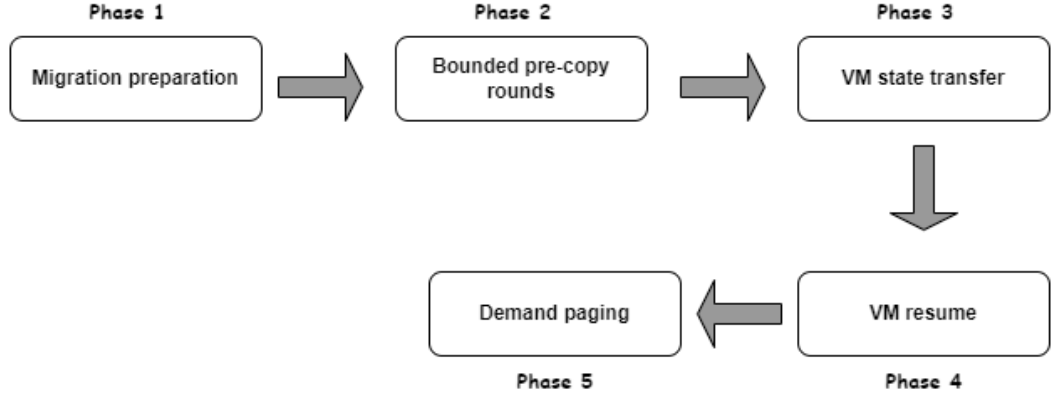


Figure 2: Hybrid migration phases (Choudhary et al. 2017)

1.3.6 Checkpointing in VM migration

The term “checkpoints” also referred to as snapshots, comes into play in VM migration failure scenarios. Checkpoints are versioned entities that contain all the required components to recover a VM to a previous state in the event of any kind of failure, ensuring the consistency and persistence of the data and execution states of the VM. As highlighted in Wang et al. (2010) checkpoints can be taken in several levels such as virtual machine level, operating system level, hardware level, compiler level, application level, and runtime library level. However, in this research, the focus will be the VM-level checkpoints that are used in the process of live migration.

As highlighted in Wang et al. (2010) a large portion of the checkpointing techniques are focussed on live migration. VMware Vmotion (Nelson et al. 2005) and Xen live migration presented in Clark et al. (2005) both focus on dirty page transfer between the source node and the destination node focusing the load balancing and fast VM recovery in the aspect of migration. VNsnap Kangarlou et al. (2009) and CEVM Chanchio et al. (2008) are some disk-based checkpointing techniques that use Copy On Write (COW) mechanism to create replicated images of a VM focussing minimizing the downtime.

Depending on the frequency or the interval the checkpoints are being taken, checkpoints are broadly categorized into two sections as mentioned in Fernando et al. (2019).

1. **Periodic checkpointing** which takes checkpoints of the VMs according to a predefined time interval.
2. **Event-based checkpointing** which does not follow strict time constraints to obtain checkpoints, but follows a method of creating checkpoints with the capturing of several predefined sets of events. With the events fired, a create checkpoint operation is triggered to capture the state of the VM.

2 Literature Review and Research Criteria

2.1 Literature Review

For the purpose of exploring the possible gaps in the relevant domain, a comprehensive literature review was done under the topic “**System Checkpoints in Fault-Tolerant Live VM Migration**”. Throughout the state of the art literature, there were numerous approaches taken to optimize the checkpointing mechanisms and to minimize the impact of the additional overheads introduced to VM migration with the checkpointing related operations. Common issues related with the available checkpointing mechanisms include the application performance degradation of the running VM, difficulty to deal with larger checkpointing file sizes when VMs have large memory sizes, additional cost and time (migration and downtimes) incurred for the checkpointing operations, network bandwidth usage overheads, checkpoint corruption risk and complexity in dealing with secure consistent checkpoints in complex systems with distributed data.

2.1.1 Checkpointing Techniques

Cully et al. (2008) proposed a VM checkpointing mechanism targeting the high availability of the VMs. Here the approach used was to have **fine grained OS image-level checkpoints** taken at really high frequencies and asynchronously replicating them in a failover node. The target was to maintain the VM live to the user, as the checkpoints are asynchronously replicated in the failover node in the background. As mentioned in Cully et al. (2008), this methodology requires highly architectural specific environments and the high frequency checkpointing required a considerable computational power and related application logic. In contrast to VM level checkpoints, Liu et al. (2009) introduced a traditional **log based checkpointing** mechanism, where the full system trace is logged, and with the help of synchronization algorithms, they are replayed at a secondary node. This reduced the overheads related to bandwidth, down times and migration times, but it possesses strict assumptions such as log transfer and log replay rates to be greater than log accumulation rate, and was limited only to single processor environments.

Several studies were carried out to optimize the checkpointing related infrastructure as well. Gelenbe (1976) introduced a **multi level checkpointing** mechanisms where several checkpoints are taken at once with varying costs and resiliency levels. Then according to the resiliency levels, they are stored in different locations such as RAM, disks or parallel file systems which are accessed according to the level of failure the VMs encounter (Moody et al. 2010). Moody et al. (2010) introduced a **scalable checkpoint restart library** which made multi level checkpointing mechanism implementable which supported checkpoint caching and having the most recent checkpoints updated in the cache and transmitting them to external file systems only when some period elapses. Íñigo Goiri, Julià, Guitart & Torres (2010) introduced a smarter infrastructure called **AUFS** which was capable of separating the read-only portions of the checkpoints from read-write parts so that

only the read-write parts are needed to be transmitted across the networks.

Kannan et al. (2013) introduced a way of using **node local NVM**, as storage units for the checkpoints. Here checkpoints taken at a higher frequency are stored at local NVM while checkpoints taken at a lower frequency with respect to a periodic time were transmitted to remote locations. Knauth & Fetzer (2015) identified that the nodes where the VMs travel are following a pattern. He introduced a method where VM checkpoints are stored in every node where a VM travels and in the event of the same VM being migrated to a previous node that it has visited, the older checkpoint is used in recycling some parts of the VM so that a minimal amount of data needs to be sent again to construct a consistent checkpoint. This was a ground breaking finding which drastically reduced the overheads related with checkpointing.

Wang et al. (2014) researched with pure software based mechanisms that developed **micro level checkpoints** which were capable of reducing the weight and overhead of the checkpoints. This methodology targeted COW mechanism with dirty page prediction and used the node local memory to store the checkpoints for the purpose of in-place recovery of the crashed VMs. This research also paved the way to analyze the checkpoint corruption possibilities and related recovery mechanisms. Wang et al. (2014) proposed that the detection latency of the corrupted checkpoints is first to be decided using artificial error injection, and then to minimize the risk of checkpoint corruption, the checkpointing interval was given a value which is greater than the detection latency. The authors of Wang et al. (2014) and Li et al. (2015) analyzed the use of **dual checkpointing schemas** to address the problem of checkpoint corruption where two checkpoints are kept at a single instance. In case of one checkpoint gets corrupted the VM is rolled back to the secondary checkpoint to mitigate the error.

Studies were also done regarding the ways to compress the checkpointing data that are transmitted via the networks. Compression techniques such as gzip, delta. Hou et al. (2013) analyzed the usage of such checkpointing techniques and concluded that none of the compression techniques are efficient in all kinds of situations. He explored that gzip is not suited for CPU intensive workloads while delta is not efficient with memory intensive workloads.

There have been numerous studies on memory compression algorithms that are efficient and lossless. **Gzip** is a widely used compression algorithm that is a combined approach of Lempel-Ziv coding (LZ77) with Huffman coding (Sardashti & Wood 2017). This combination allows Gzip to have a relatively high compression ratio highlighting the effective use of it with the data with repetitive patterns which aligns well with memory compression scenarios. Sardashti & Wood (2017) also highlighted that this good compression ratio comes with a price of having a relatively high CPU overhead. **LZ4** is another fast compression algorithm with low latency which makes it particularly suitable for real-time data processing applications like live VM migration. LZ4 is a lossless data compression algorithm that is focused on real-time compression and decompression speed over compres-

sion ratio and it is based on LZ77 algorithm. LZ4 works by simply spotting and compressing any repetitive data patterns with a much high speed compression and decompression rates according to Bartík et al. (2015). LZ4’s fast compression can significantly help in minimizing service downtime and improving resource utilization. **ZSTD** or “Z-standard” is another lossless compression technique discussed in the literature. Developed by Facebook, ZSTD uses a combination of dictionary-based compression and Huffman coding, allowing it to achieve higher compression ratios maintaining competitive speed (Collet 2021).

There have also been studies on how to identify the **page delta** or the memory page difference and efficiently transmit them to destination during VM migrations. The combined use of XOR with run length encoding as highlighted in Bhardwaj & Rama Krishna (2019) is a good finding that shows the possibility of digging deeper to explore the modified content in the a page rather than transmitting the entire page as a whole.

Fernando et al. (2016) proposed the approach of having **incremental checkpoints** to reduce the eviction time of the migration. The incremental checkpoints are taken in a way,

- First, all the memory pages(entire pages) are transmitted as a whole.
- In the successive iterations, only the modified pages(entire pages) since the last iteration are transmitted.

Here the checkpoints are incrementally transmitted to the destination or a third staging node. When the time arrives to migrate the VM, only the remaining memory since the last snapshot is needed to be transmitted which reduced the eviction time drastically. Here the checkpoints are not complete VM images, but portions of the VM memory that will finally contribute to have a consistent memory state at the end with the final memory transmitted at during the eviction time. With this concept the same authors Fernando et al. (2019) introduced **reverse incremental checkpointing** which solved a major issue. During postcopy migration, as the execution states are resumed at the destination when the migration starts, if a network or destination node failure occurs, the VM becomes unrecoverable as the execution state is split between the source and the destination. Here soon as the destination starts its execution, incremental checkpoints are generated and stored in a third node or the source node itself which are utilized in cases of network or destination failures. Combining this technology, the same authors Fernando et al. (2023) introduced a combination of Forward Incremental Checkpointing (FIC) and Reverse Incremental Checkpointing (RIC), where the checkpoints are taken in both directions i.e. the source VM checkpoints are stored in destination or a third staging node while the destination VM checkpoints are stored in the source or a third staging node. With this mechanism the VM migrations can be resilient against source, network and the destination failures, which are the only three points of failure present during migration.

2.2 Research Gap

State of the art literature including Fernando et al. (2016), Fernando et al. (2019), Fernando et al. (2023) have contributed to develop checkpointing mechanisms which are resilient against all the points of failures which could occur during migration which are namely the source node, destination node and the interconnecting network. With the incremental checkpointing introduced, the amount of memory pages transmitted in a checkpoint have drastically reduced due to the sending of only the modified memory pages since the last iteration.

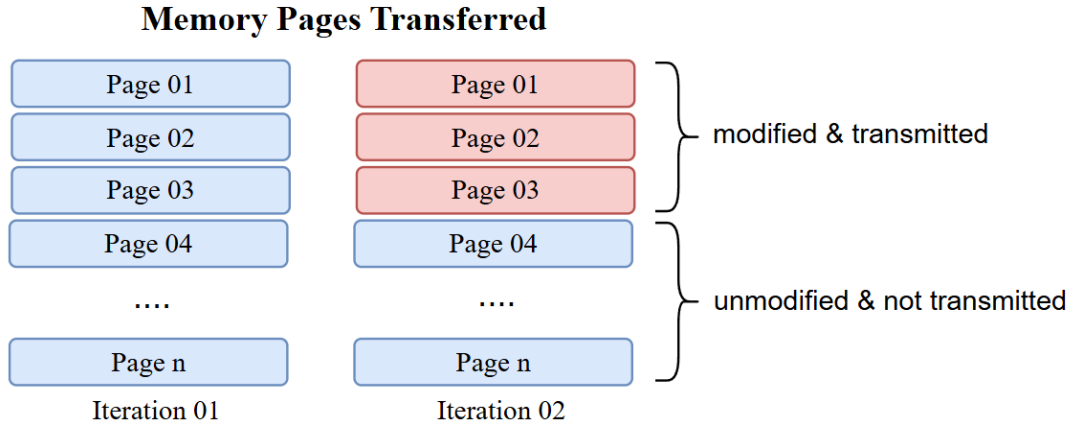


Figure 3: Incremental checkpointing(Fernando et al. 2019)

With this mechanism, the network bandwidth is preserved as the amount of data transmitted within networks is decreased. Additionally, the overheads on the computing resources of the source VM are minimized.

However, even though the whole modified page is transferred as shown in figure 3 in successive iterations, the actual modified portion of the respective memory page can be limited to a few bytes out of several kilo bytes when considering a page of 4KB. In such cases, transmitting of the entire memory page with a few modified bytes and a load of unmodified bytes is in vain which consumes additional computational resources and introduces additional network bandwidth.

Additionally for the relevant mechanism in addition to going to the level of page level modifications, the authors haven't explored the possibility of compressing the checkpointing data before transmitting. The data to be transmitted can be further compressed using checkpoint compression algorithms like Gzip, LZ4, Zstd etc (Sardashti & Wood 2017, Bartík et al. 2015, Collet 2021). This will further reduce the overheads generated with checkpointing mechanisms.

2.3 Research Questions

1. How can the incremental checkpoints be further improved to reduce overheads incurred on VM migration ?
2. How can the state of the art compression techniques be incorporated in reducing checkpoint size during state transfers ?

2.4 Aims and Objectives

2.4.1 Aim

The main aim of this research is to improve and enhance the productivity and cost-effectiveness of incremental checkpointing in VM migration in CDCs by making the checkpointing operations more granular and to analyze the improvements in related overheads with the integration of checkpoint compression techniques.

2.4.2 Objectives

- Design and develop a strategy to make the checkpoint data transmitted more granular for the purpose of reducing the overheads associated with current checkpointing operations.
- Analyze the improvements in the overheads incurred introduced by the new granular methodology with respect to the existing methodologies.
- Integrate checkpoint compression techniques and analyze the improvements made in terms of the overheads introduced.

2.5 Scope

2.5.1 In Scope

- **Baseline model implementation** - The baseline incremental checkpointing model in Fernando et al. (2019) will be reproduced in QEMU/KVM.
- **Byte/Bit-level granular model implementation** - A byte/bit level fine granular checkpointing method will be introduced that goes beyond page level modification tracking and transfer in checkpointing iterations.
- **Implementation of a working prototype** - A working prototype will be implemented with the granular byte/bit-level mechanism.
- **Integration of checkpoint compression techniques** - State of the art checkpoint compression techniques will be integrated to further reduce the checkpoint data transfer overheads.
- **Evaluation** - The results and overhead reductions will be evaluated with approved realistic industrial benchmarks.
- **Scope limited to KVM-QEMU hypervisor** - All implementations will be restricted to the KVM-QEMU hypervisor environment.
- **Single VM failure scenarios** - The focus will be on handling single VM failures, with the potential for future extension to multiple VM failure scenarios.
- **Linux-based Ubuntu OS** - The implementations will be developed and tested exclusively on Linux-based Ubuntu operating systems.

2.6 Significance of the Research

With the CC domain evolving day by day, the need for enhancements and improvements in virtualization and VM migrations in CDCs is vital. So far in the literature, the incremental checkpointing techniques in VM migration highlighted in Fernando et al. (2016), Fernando et al. (2019) and Fernando et al. (2023) etc. follow an order where the first checkpoint captures all memory pages and the successive checkpointing rounds capture and transmit only the modified pages.

However, when a page of 4 KB is considered, although the page is modified, the actual modifications can be limited to several bytes/bits out of 4 KB. In such a case transmitting the whole 4 KB worth of data via the networks is an invain of time and resources which incurs several overheads like network bandwidth, higher migration and downtimes, application performance degradation in running VMs etc. This mechanism can further be improved to make the data transmitted via the networks more granular. Instead if transmitting the entire page where modifications were made, the page can be examined deeper to identify the respective bytes/bits which were modified and only the modified bytes/bits can be transmitted without transmitting the entire memory page as shown in figure 4. This will result in further reducing the network bandwidth usage to transmit the checkpointing data to the checkpoint stores, reduce the costs associated with checkpoint stores and significantly reduce the checkpoint replication time to and from the checkpoint stores to the nodes and vice versa. The overheads on the total migration time and downtime, application performance degradation etc. will also be acceptable. This methodology will also reduce the failure window of the migration and the recovery time window.

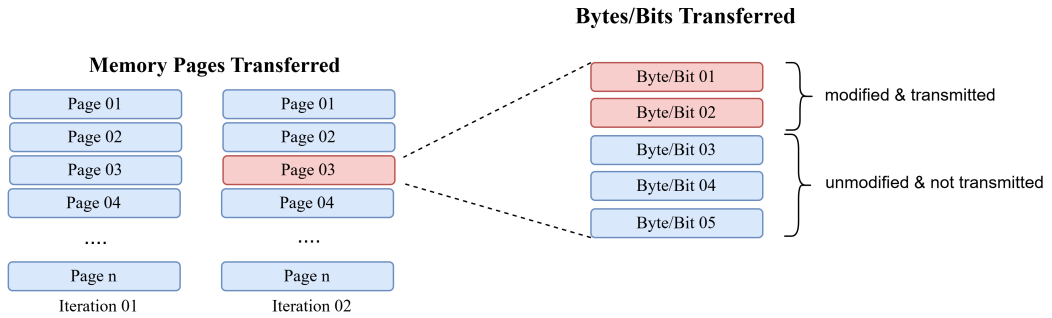


Figure 4: Proposed improvement(Fernando et al. 2019)

With this byte/bit level granularity introduced, only the modified bytes/bits out of the 4KB memory page is transmitted which will be drastically reducing the load of checkpointing data that needs to be transmitted. When a VM bearing around 8GB of memory is considered, the savings resulted through this optimization technique is massive. With this improvement, the VM migration domain will be much more improved, efficient and cost-effective at the same time that leads to a good contribution to the domain of Cloud Computing.

3 Methodology and Design

3.1 Research Approach and Methodology

The structure of the research can be broken down into several phases. The initial stage would be the Testbed setup with servers configured and analyzing the migration methods involving pre-copy, post-copy, hybrid.

- **Testbed Setup** : The testbed consists of two physical servers with adequate processing power and RAM. The host OS chosen is Ubuntu Server with QEMU-KVM utilized as the hypervisor. The VMs running in the servers will have Linux-based OSs. The servers are interconnected via Gigabit Ethernet. Additionally a third server is used as a staging server for checkpointing operations. Figure 5 shows the testbed setup.

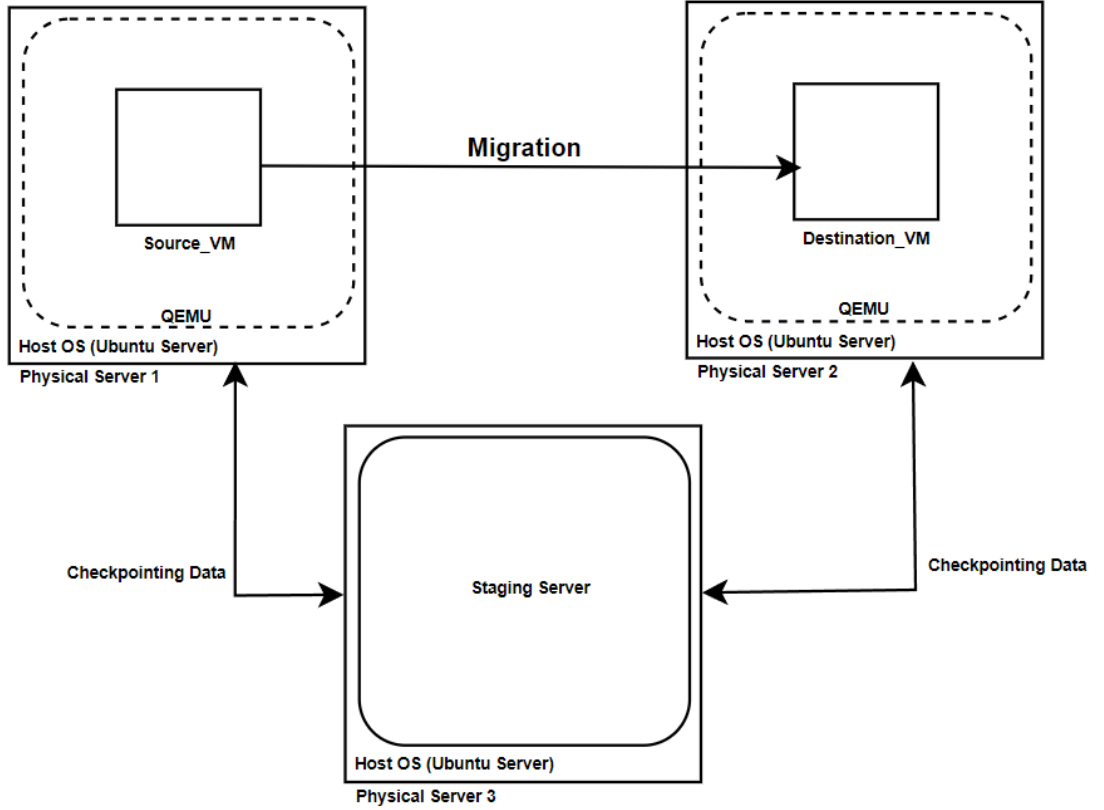


Figure 5: TestBed Setup

- **Implement Baseline Model** - The approach of page level incremental checkpointing utilized in Fernando et al. (2019) will be reproduced in QEMU/KVM and will be analyzed and tested.
- **Introduce Byte/Bit-level Granularity** - Byte/Bit-level granularity will be introduced to the implemented baseline model deviating deeper from page level granularity.

- **Evaluate overhead improvements** - Evaluate the overhead reductions with the byte-level granularity introduced compared to page-level Fernando et al. (2019) findings.
- **Integrate checkpoint compression techniques** - Integrate state of the art checkpoint compression techniques to further reduce the checkpoint data transfer overheads.
- **Evaluate findings with realistic benchmarks** - Evaluate the findings, overhead reductions and CPU, memory, bandwidth utilization with realistic workload benchmarks like iPerf, SysBench, Quicksort, Workingset (Fernando et al. 2023) etc. The improvements in the overhead reductions in the byte/bit level granular mechanism and the compression techniques integration will be compared with the existing page-level mechanism.

3.2 Research Design

The design of the research is mainly divided into two parts to address the two major objectives of making the checkpointing data more granular by introducing a methodology to track the modified content inside a modified page and to investigate the possibility of using checkpoint compression algorithms to further reduce the data to be transmitted as checkpoints.

3.2.1 Identifying the modified page content instead of transmitting the entire page

As in Fernando et al. (2019), in successive checkpointing rounds, the modified pages since the last completed checkpointing iteration are only considered. Earlier without considering the nature of the modified content, if the page was marked as modified it was transmitted in the checkpoint. With the optimized approach introduced, a successful way of identifying the modified content inside the modified page was successfully designed and integrated to the incremental checkpointing mechanism introduced in Fernando et al. (2019). The architecture of the newly proposed methodology is shown in the below diagram 6.

As proposed in the figure 6, the model will be capable of tracking the modified content inside the modified pages, and only transmit them during checkpointing, instead of repeatedly transmitting entire page. Here this proposed methodology will act in a way where, when a migration is initiated, a local in-memory data structure will be created for the purpose of storing the pages which are being sent, to be used in the future to analyze the actual content which were modified. When the checkpointing is initiated, in incremental checkpointing proposed by Fernando et al. (2019), it repeatedly checks the entire RAM of the VM to identify the modified pages and considers only the modified pages since the last iteration during the current iteration. When the modified pages are found, with the help of the page offset, the older version of that particular page stored in the local in-memory page storage is being retrieved. In the case of an older version of the page being not available, that page will be entered to the in-memory page storage structure and it will be treated as normal and sent to the checkpoint store. If an older version is available, the new and the older versions of the pages are XORed to identify the modified bytes inside the page. Identical bytes will yield zero after the XOR operation and the modified content will remain as it is. Then giving priority to the modified content, the XORed version of the page is being encoded using zero run length encoding.

During zero run length encoding, the unmodified part of the page which resulted zeros is treated as a zero run, and that sequence of zero bytes is replaced by the no. of zeros in Unsigned Little Endian Base 128 (ULEB128) format. For the modified bytes or the non-zero runs, it is represented with the length of the non-zero run followed by the modified content. Finally this minimized encoded version is being sent to the checkpoint store instead of the whole 4KB page and the in-memory page store is updated with the later version of the page for future use. In the event

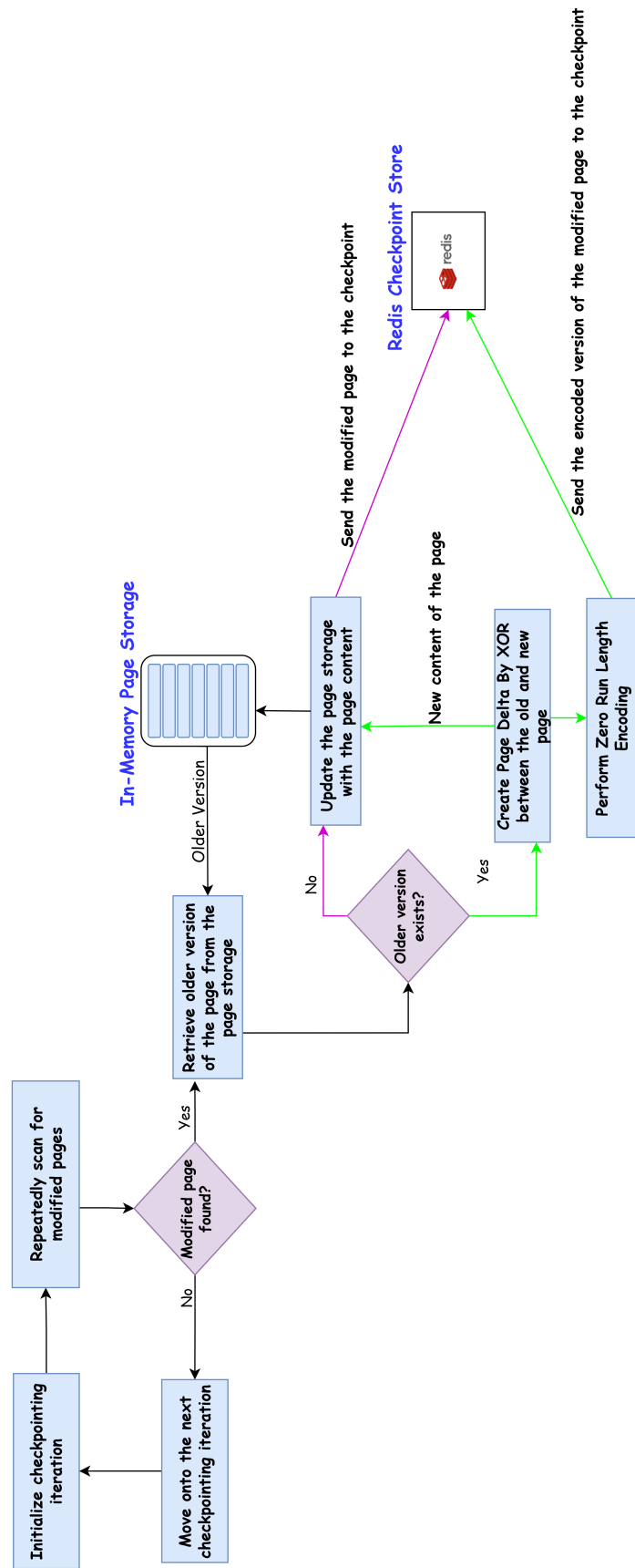


Figure 6: Byte Level Granularity Approach

of a failure, at the recovery end, when loading the memory pages, the encoded pages are decoded to the original format and the new page is reconstructed with the help of the decoded version and the respective older page content available. The encoding/decoding is clearly elaborated in section 3.3.2.

3.2.2 Ability to use checkpoint compression techniques to reduce the content to be transmitted

To achieve the second goal of applying the compression techniques to reduce the content to be transmitted, the model proposed at Fernando et al. (2019) is optimized by integrating a checkpoint compression mechanism before transmitting the pages into the checkpoint store. Here, three compression algorithms were taken into account which are namely Gzip, Lz4 and Zstd which were discussed in section 2.1.1. Before transmitting a modified page to the checkpoint store, it is compressed using a compression algorithm that can be specified by the user. In the event of a failure, at the recovery end, when the memory pages are loaded, they are decompressed to the original size and loaded into the memory.

3.3 Preliminary Results and Discussion

3.3.1 Byte-level Granularity Mechanism

As explained in section 3.2, two models were created. The first model integrated a byte-level granularity mechanism, capable of tracking modified content within a modified page and prioritizing only the modified content instead of transmitting the entire page.

To elaborate further with an example, the modified content of a page is identified and encoded as follows:

Initial Buffers

Consider an old memory page buffer and a new memory page buffer that differ only in a few bytes. The buffers are as follows:

- **Old Buffer:**

```
75 zeros
11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 00 00 11 23 25
4000 zeros
```

- **New Buffer:**

```
75 zeros
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 20 00 00 11 22 24
4000 zeros
```

In this example, the only differences between the `old` and `new` buffers are a few bytes in the middle of the buffer. Instead of sending the entire new buffer during checkpointing, this byte-level granularity mechanism allows us to encode just the differences, thereby reducing the amount of data transferred.

Encoded Buffer

Using this mechanism, the data is compressed into a series of **runs**. Each run is either:

- A *zero run*, which specifies the number of consecutive zero bytes, or
- A *non-zero run*, which specifies the length of the run followed by the actual data bytes.

The encoded buffer for this example is as follows:

```
Encoded Length: 21
4b 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 04 02 22 24
```

Breaking down each component of this encoded buffer:

Decoding the Encoded Buffer

At the decoding end, the older version of the page is loaded to memory and it is updated using the encoded version of the page difference as explained below.

- `4b` – This represents a **zero run** of length 75 bytes in ULEB128 format, indicating that the first 75 bytes in the new buffer are unchanged from the old buffer. Hence nothing needs to be done to the loaded earlier version.
- `0f` – This represents a **non-zero run** of length 15 bytes in ULEB128 format. Following this length byte, we have the actual 15 bytes of modified data:

```
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e
```

These bytes are the updates in the new buffer which are then updated in the loaded earlier version of the page.

- `04` – This represents another **zero run**, this time with a length of 4 bytes. These bytes remain unchanged in the new buffer.
- `02 22 24` – This represents a **non-zero run** of length 2 bytes followed by the modified content 22 and 24. These 2 bytes are also then updated in the loaded earlier version of the page.
- The last 4000 bytes of zeros are ignored in the encoding as that content is equal in both old and new buffer which represents a zero run.

Once this action is completed, the initially loaded memory page will be reflecting the content of the new version of that same memory page.

Discussion

Using this byte-level granularity mechanism, we can significantly minimize the amount of data transferred by encoding only the differences between the `old` and `new` buffers. For this example, the encoded buffer length is just 21 bytes, which is much smaller than the full memory page which is 4KB, thereby reducing the bandwidth required for transmitting checkpoints.

Experimental Results

When analyzing the results of the preliminary experiments it is clear that using this byte-level granularity mechanism which is capable of tracking the modified bytes out of a 4KB page, the content transmitted as checkpoints to the destination shows a decrement in a drastic scale. The experiments were done by implementing this mechanism in QEMU/KVM and migrating a VM of 8GB memory. When compared with the earlier version of incremental checkpointing proposed in Fernando et al. (2019), the bandwidth savings due to the decrement in checkpoint content transfer is massive.

Workload nature	Idle	7000(MB) -Working Set
Average dirty pages found	6168	6480
Average page content(bytes) transmitted including pages without having an earlier older version. (All pages considered)	1563.17	1540.1
Percentage of content reduced (including pages without older version)	61.84%	62.40%
Average page content(bytes) considering only the pages having an older version	47.41	54.47
Percentage of content reduced (considering pages with older versions only)	98.84%	98.67%

Table 1: Byte-level granular methodology results

As explained in table 1 and illustrated in figure 7, when this byte level granular methodology was implemented, the average page content transmitted is reduced by a drastic scale. When all the pages were considered including the pages that were transmitted as 4096 bytes due to them being transmitted during the first instance without having an older version of them in the in-memory page store, and the encoded pages where an older version of the page were available, the average page content transmitted was observed to be **1563-1540 bytes** depending on the workload. This is a drastic decrement when compared with the traditional way of transmitting 4096 bytes repeatedly as a whole. When considering only the pages that had an earlier version in the in-memory page store to perform encoding it was observed that the traditional 4096 bytes worth of page data were reduced to around **47-54 bytes**. This highlights the efficiency of the byte level granular methodology and ability to extract the modified content of a modified page can result the valuable content of the page to be reduced by a huge scale.

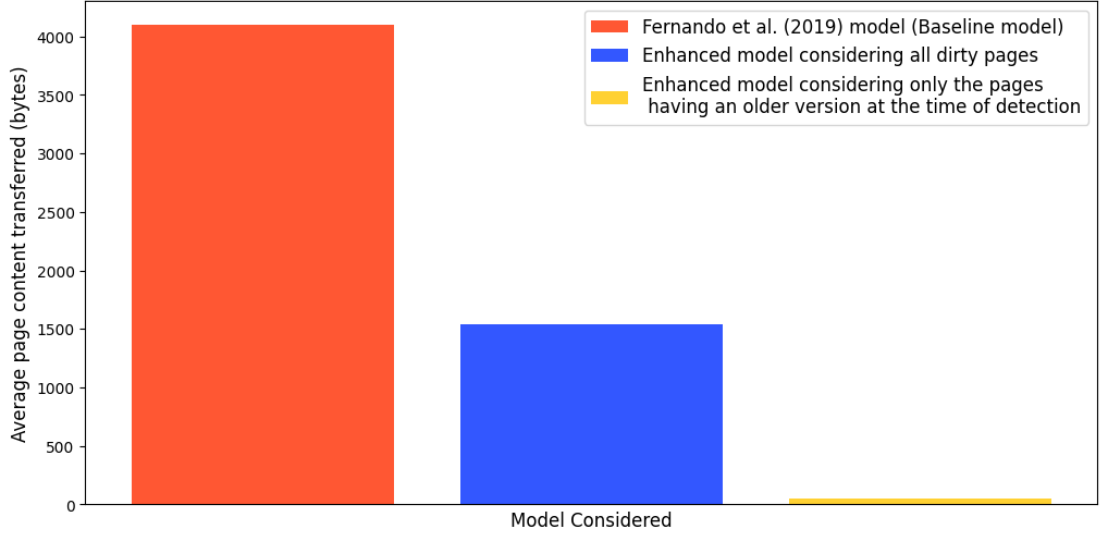


Figure 7: Page content reduction in byte-level granularity approach

3.3.2 Compression Algorithm Integration

The second model implemented was to integrate checkpoint compression algorithms to reduce the page content transferred as checkpoints. As emphasized in section 3.2.2, three lossless compression algorithms namely Gzip, LZ4 and ZSTD were integrated to the model proposed by Fernando et al. (2019), to reduce the content to be transmitted. With the compression enabled, the original page content of 4096 bytes were reduced by a large factor preserving the bandwidth to transmit the memory checkpoint pages to the checkpoint store.

Nature of the workload	GZIP	LZ4	ZSTD	Fernando et al. (2019) model
Idle	877.47	1159.11	2046.60	4096
7000(MB) - Working set	957.49	1255.94	1926.20	4096

Table 2: Comparison of page content transmission with compression techniques and baseline Fernando et al. (2019) model across workloads

As depicted in table 2 and illustrated in figure 8, when a VM with 8GB memory was migrated with the integrated compression techniques enabled, in the checkpointing rounds when modified pages were identified they were first compressed and then sent to the external checkpoint store. With this methodology, instead of transmitting a whole chunk of 4096 bytes of the page, a compressed version of the page was sent. With GZIP the average size of the page transmitted summed upto 877.47 - 957.49 bytes, with LZ4 1159.11 - 1255.94 bytes and with ZSTD 1926.20 - 2046.60 bytes depending on the workload. With Gzip it showed an average page content decrement of 76.62 - 78.57%, with LZ4 a decrement of 69.33 - 71.70% and finally with Zstd a decrement of 50.03 - 52.97% depending on the workload. This results a good decrement when compared with the traditional way of transmitting whole 4096 bytes each and every time.

Compression Algorithm	GZIP	LZ4	Zstd
Page Content Reduction(Idle)	78.57%	71.70%	50.03%
Page Content Reduction(7000 (MB) - workingset)	76.62%	69.33%	52.97%

Table 3: Comparison of page content reduction percentage compared to baseline Fernando et al. (2019) model

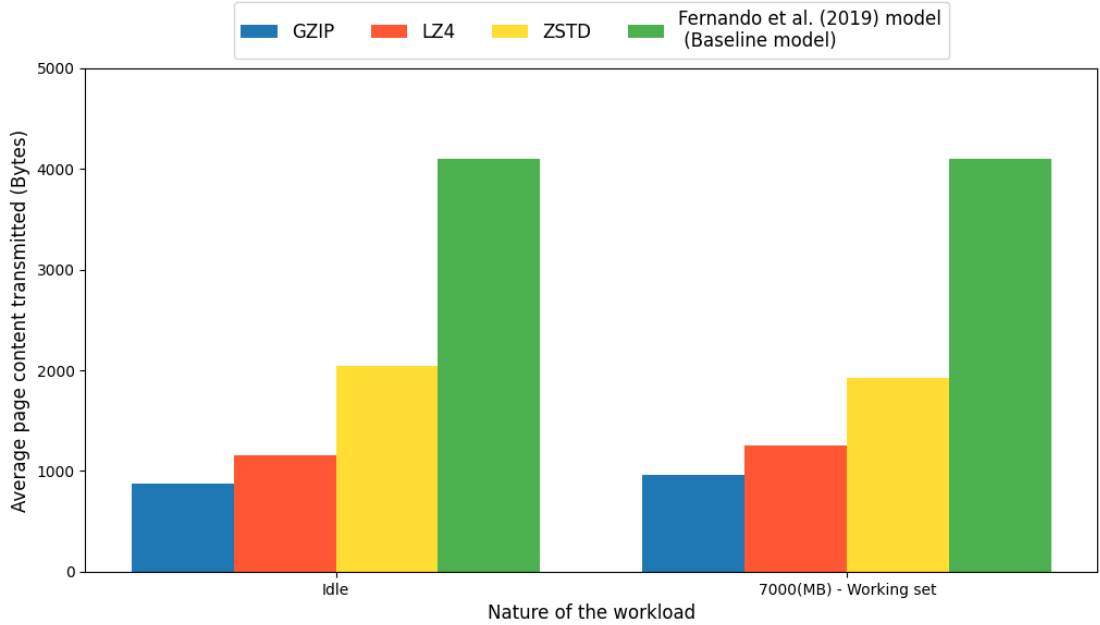


Figure 8: Page content reduction with compression types

4 Tools, Evaluation, and Project Timeline

4.1 Research Tools Used

4.1.1 QEMU

QEMU is a generic open-source emulator and virtualizer, utilized in this research to handle virtualization tasks. It integrates with the KVM hypervisor on the Linux servers within the research environment, providing robust capabilities for VM management. QEMU supports various migration techniques, including pre-copy and hybrid migration. The experiments in this research were conducted using QEMU Emulator version 2.8.1.1.

4.1.2 RealVNC Viewer

RealVNC Viewer is used to remotely access VMs on both the source and destination servers. This enables convenient monitoring and control of the VMs during migration, allowing access from a personal computer to observe the VMs' status and performance.

4.1.3 NFS

NFS facilitates shared storage between the source and destination servers, allowing both servers to access the VMs' disk images without the need to migrate disk data during LAN migrations. The destination server hosts the NFS server, sharing a directory over the network, while the source server connects as an NFS client, accessing this shared storage.

4.1.4 RedisInsight

RedisInsight is a GUI tool used to monitor and interact with the Redis database, which is employed as a checkpoint store in this research. This tool provides real-time visibility into the memory pages and data stored within Redis during migration, enabling analysis of checkpointing efficiency and integrity.

4.2 Evaluation

4.2.1 Evaluation of the accuracy of the implemented methodologies

When byte-level granularity methodology as well as compression techniques are integrated to the working reproduced version of the Fernando et al. (2019) model, with each change added, VMs are migrated and recovered to check the usability of the checkpoints. With the new techniques integrated, QEMU is rebuilt and installed and VMs are migrated and failures will be manually triggered to initiate recovery process. With the completion of the recovery process, if the VM is still running smoothly, it implies that the captured memory and CPU checkpoints are accurate and the methods are working efficiently.

4.2.2 Whole system evaluation

The improvements to the incremental checkpointing methodology from the byte-level modification tracking granular mechanism and the integration of the compression techniques will be thoroughly analyzed in terms of the **checkpointing data content reduction/network bandwidth reduction, checkpoint replication time reduction, faulty window time reduction, total migration time overhead, total downtime overhead, VM application performance degradation and recovery time overhead**. These factors will be compared with the baseline model proposed at Fernando et al. (2019) which uses page level modification tracking and transfer during checkpointing. Standard industrial workloads and synthetic benchmarks which are namely workingset, iperf, sysbench, and quicksort will be used in generating the workloads to evaluate the above mentioned overheads with respect to the baseline model in Fernando et al. (2019).

4.3 Project Timeline

Task	June	July	August	September	October	November	December	January	February	March	April
Literature Survey											
Background research											
Project proposal											
Background incremental checkpointing analysis and base model implementation											
Design the modifications for the granular byte level methodology											
Implement the design in KVM/QEMU platform											
Test and analyze the overhead reduction											
Integrate compression techniques and evaluate the results											
Overall performance evaluation											
Thesis writing											
Research publication											

Figure 9: Project Timeline

References

- Agarwal, A. & Raina, S. (2012), ‘Live migration of virtual machines in cloud’, *International Journal of Scientific and Research Publications* **2**(6), 1–5.
- Ahmad, R. W., Gani, A., Ab. Hamid, S. H., Shiraz, M., Xia, F. & Madani, S. A. (2015), ‘Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues’, *The Journal of Supercomputing* **71**, 2473–2515.
- Alibaba Cloud (2024), ‘Alibaba cloud’.
URL: https://in.alibabacloud.com/?utm_key=se_1007710386&utm_content=se_1007710386&gclid=Cj0KCQiA8a0eBhCWARIsANRFrQFZhZk5PnatY5otgaHs5v4Z0-vFwNnjqDHoWZRGN3uXSJOW_BnYIWQaAik3EALw_wcB
- AWS (2024), ‘Amazon web services’.
URL: <https://aws.amazon.com/>
- Bartík, M., Ubik, S. & Kubalik, P. (2015), Lz4 compression algorithm on fpga, *in* ‘2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)’, IEEE, pp. 179–182.
- Bhardwaj, A. & Rama Krishna, C. (2019), Improving the performance of pre-copy virtual machine migration technique, *in* ‘Proceedings of 2nd International Conference on Communication, Computing and Networking: ICCCN 2018, NITTTR Chandigarh, India’, Springer, pp. 1021–1032.
- Chanchio, K., Leangsuksun, C., Ong, H., Ratanasamoot, V. & Shafi, A. (2008), An efficient virtual machine checkpointing mechanism for hypervisor-based hpc systems, *in* ‘High Availability and Performance Computing Workshop’.
- Choudhary, A., Govil, M. C., Singh, G., Awasthi, L. K., Pilli, E. S. & Kapil, D. (2017), ‘A critical survey of live virtual machine migration techniques’, *Journal of Cloud Computing* **6**(1), 1–41.
- Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I. & Warfield, A. (2005), Live migration of virtual machines, *in* ‘Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2’, pp. 273–286.
- Collet, Y. (2021), ‘Rfc 8878: Zstandard compression and the application/zstd media type’.
- Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N. & Warfield, A. (2008), Remus: High availability via asynchronous virtual machine replication, *in* ‘Proceedings of the 5th USENIX symposium on networked systems design and implementation’, San Francisco, pp. 161–174.
- Damania, K., Holmukhe, S., Singhai, V. & Bhavathankar, P. (2018), An overview of vm live migration strategies and technologies, *in* ‘2018 Second Interna-

- tional Conference on Electronics, Communication and Aerospace Technology (ICECA)', pp. 1185–1190.
- Fernando, D., Bagdi, H., Hu, Y., Yang, P., Gopalan, K., Kamhoua, C. & Kwiat, K. (2016), Quick eviction of virtual machines through proactive live snapshots, *in* 'Proceedings of the 9th International Conference on Utility and Cloud Computing', pp. 99–107.
- Fernando, D., Turner, J., Gopalan, K. & Yang, P. (2019), Live migration ate my vm: Recovering a virtual machine after failure of post-copy live migration, *in* 'IEEE INFOCOM 2019-IEEE Conference on Computer Communications', IEEE, pp. 343–351.
- Fernando, D., Turner, J., Yang, P. & Gopalan, K. (2023), 'V-recover: Virtual machine recovery when live migration fails', *IEEE Transactions on Cloud Computing*.
- GCP (2024), 'Google cloud platform'.
URL: <https://cloud.google.com/>
- Gelenbe, E. (1976), A model of roll-back recovery with multiple checkpoints, *in* 'Proceedings of the 2nd international conference on Software engineering', pp. 251–255.
- Hines, M. R., Deshpande, U. & Gopalan, K. (2009), 'Post-copy live migration of virtual machines', *ACM SIGOPS operating systems review* **43**(3), 14–26.
- Hou, K.-Y., Shin, K. G., Turner, Y. & Singhal, S. (2013), Tradeoffs in compressing virtual machine checkpoints, *in* 'Proceedings of the 7th international workshop on Virtualization technologies in distributed computing', pp. 41–48.
- Kangarlou, A., Eugster, P. & Xu, D. (2009), Vnsnap: Taking snapshots of virtual networked environments with minimal downtime, *in* '2009 IEEE/IFIP International Conference on Dependable Systems & Networks', IEEE, pp. 524–533.
- Kannan, S., Gavrilovska, A., Schwan, K. & Milojicic, D. (2013), Optimizing checkpoints using nvm as virtual memory, *in* '2013 IEEE 27th International Symposium on Parallel and Distributed Processing', IEEE, pp. 29–40.
- Knauth, T. & Fetzer, C. (2015), Vecycle: Recycling vm checkpoints for faster migrations, *in* 'Proceedings of the 16th Annual Middleware Conference', pp. 210–221.
- Li, G., Pattabiraman, K., Cher, C.-Y. & Bose, P. (2015), Experience report: An application-specific checkpointing technique for minimizing checkpoint corruption, *in* '2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)', IEEE, pp. 141–152.
- Liu, H., Jin, H., Liao, X., Hu, L. & Yu, C. (2009), Live migration of virtual machine based on full system trace and replay, *in* 'Proceedings of the 18th

- ACM international symposium on High performance distributed computing’, pp. 101–110.
- Microsoft (2024), ‘Microsoft azure’.
URL: <https://azure.microsoft.com/en-us>
- Milojičić, D. S., Douglass, F., Paindaveine, Y., Wheeler, R. & Zhou, S. (2000), ‘Process migration’, *ACM Computing Surveys (CSUR)* **32**(3), 241–299.
- Moody, A., Bronevetsky, G., Mohror, K. & De Supinski, B. R. (2010), Design, modeling, and evaluation of a scalable multi-level checkpointing system, *in* ‘SC’10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis’, IEEE, pp. 1–11.
- Nelson, M., Lim, B.-H., Hutchins, G. et al. (2005), Fast transparent migration for virtual machines., *in* ‘USENIX Annual technical conference, general track’, pp. 391–394.
- Sahni, S. & Varma, V. (2012), A hybrid approach to live migration of virtual machines, *in* ‘2012 IEEE international conference on cloud computing in emerging markets (CCEM)’, IEEE, pp. 1–5.
- Sardashti, S. & Wood, D. A. (2017), ‘Could compression be of general use? evaluating memory compression across domains’, *ACM Transactions on Architecture and Code Optimization (TACO)* **14**(4), 1–24.
- Wang, L., Kalbarczyk, Z., Iyer, R. K. & Iyengar, A. (2010), Checkpointing virtual machines against transient errors, *in* ‘2010 IEEE 16th International On-Line Testing Symposium’, IEEE, pp. 97–102.
- Wang, L., Kalbarczyk, Z., Iyer, R. K. & Iyengar, A. (2014), ‘Vm- μ checkpoint: Design, modeling, and assessment of lightweight in-memory vm checkpointing’, *IEEE Transactions on Dependable and Secure Computing* **12**(2), 243–255.
- ÓÍñigo Goiri et al.
- Íñigo Goiri, Julià, F., Guitart, J. & Torres, J. (2010), Checkpoint-based fault-tolerant infrastructure for virtualized service providers, IEEE Computer Society, pp. 455–462.