An Improved Pre-Copy Technique for Container Migration

A

Thesis

Submitted to

For the award of

DOCTOR OF PHILOSOPHY (Ph.D.)

in

COMPUTER SCIENCE AND ENGINEERING

By

Gursharan Singh

41500180

Supervised By :

Dr. Parminder Singh

LOVELY FACULTY OF TECHNOLOGY AND SCIENCES

LOVELY PROFESSIONAL UNIVERSITY

PUNJAB

2022

DECLARATION

This thesis is an account of research undertaken between December 2015 and June 2022 at the School of Computer Science and Engineering, Lovely Professional University, Phagwara, India.

Except where acknowledged customarily, the material presented in this thesis is, to the best of my knowledge, original and has not been submitted in whole or part for a degree in any university.

Gursharan Singh
Registration no. 41500180
Department of Computer Science and Engineering
Lovely Professional University, Phagwara, India

CERTIFICATE

This is to certify that the declaration statement made by the student is correct to the best of my knowledge and belief. He has submitted the Ph.D. thesis **An Improved Pre-Copy Technique for Container Migration** under my guidance and supervision. The present work results from his original investigation, effort, and study. No part of the work has ever been submitted for any other degree at any university. The Ph.D. thesis is fit for the submission and fulfillment of the conditions for the award of a Ph.D. degree in Computer Science and Engineering from Lovely Professional University, Phagwara.

Dr. Parminder Singh
Associate Professor
Department of CSE
Lovely Professional University,
Phagwara, India

ABSTRACT

Cloud computing is a cost-effective method of delivering numerous services. The demand for dynamic cloud services is rising day by day, and because of this, data transit across the network is extensive. Virtualization is a significant component, and the cloud servers might be physical or virtual. Containerized services are essential for reducing data transmission costs and time, among other things. Containers are lightweight virtual environments that share the host operating system's kernel. The majority of businesses are transitioning from virtual machines to containers. The primary factor affecting the performance is the amount of data transferred over the network. It has a direct impact on migration time, downtime, and cost.

In this thesis, the pre-copy container live migration is analyzed in a detailed manner to trace out the possibilities to improve the performance. The container migration technique used is pre-copy and, it is further divided into three phases: pre-dump, iterative dump, and final dump. For the pre-dump phase, a migration technique has been proposed through probability-based Particle Swarm Optimization (PSO) to overcome the memory transmission limitations. The dirty pages are predicted during the migration process using the meta-heuristic approach. The active set of pages and their update rate has also been identified, and based on the threshold level of maximum update rate, the pages have been shortlisted to be discarded from pre-dump. The proposed technique is implemented and tested on various scenarios with different batch sizes and thresholds.

The next phase is iterative-dump, and a predictive iterative-dump approach is designed using Long Short-Term Memory (LSTM) to anticipate which memory pages

will be moved by limiting data transmission during the iterative phase. In each loop, the pages are shortlisted to be migrated to the destination host based on predictive analysis of memory alterations. Dirty pages will be predicted and discarded using a prediction technique based on the alteration rate.

Container migration enables load balancing, system maintenance, and fault tolerance, among other things. When a container migrates to another host, after a successful migration, the container image will be removed from the source host. But, in the case of migrating back to the previous host, almost the same amount of data will be retransmitted. A dump reusing approach is proposed for container migration to migrate back to the same host to address this issue. The original image kept on the source host can be reused. The memory pages similar to the source image are not sent back. Only the updated pages will be transferred. This approach helps in reducing the amount of data transmission over the network. Furthermore, if the container image is kept on the source host, it will provide demand paging and help recover from failure at the destination host.

The experiment results show a significant improvement in the reduction of data transmitted, downtime, migration time, and the overall cost of migration compared to existing techniques. The proposed predictive pre-copy approach is profitable for container migration and ensures the live migration with minimal downtime.

ACKNOWLEDGEMENTS

I take this opportunity to express my sincere thanks to everyone who has helped me in various capacities to carry out this research and prepare the report.

I am delighted to thank our respected supervisor, Dr. Parminder Singh, who has offered tremendous support in completing this research. I would also like to thank my former supervisor, Dr. Pooja Gupta, for the encouragement and advice she has provided throughout my time as her student.

I acknowledge the School of Computer Science and Engineering, Lovely Professional University, for providing me with the appropriate resources and financial support to pursue the doctoral degree. I am grateful to the administrative staff at the Centre for Research Degree Programmes for the numerous applications.

I would also like to thank my parents, wife, son, daughter, friends, and colleagues for their co-operation and compliance. Their care and love are indispensable to my achievements. I cannot cherish a greater fortune other than having them in my life.

Gursharan Singh

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
AODC	Application Oriented Docker Container
CPU	Central Processing Unit
CR	Checkpoint and Restore
CRI	Container Run-time Interface
CRIU	Checkpoint and Restore in Userspace
DB	Dirty Bits
ELM	Extreme Learning Machines
EOP	End of Memory Pool
GA	Genetic Algorithm
HW	Hot workspace
IaaS	Infrastructure as a Service
LSTM	Long short-Term Memory
LXC	Linux Containers
MP	Memory Pool
MSB	Most Significant Bit
OCI	Open Container Initiative
OPCA	Optimized Pre-copy Algorithm
PID	Process ID
PSO	Particle Swarm Optimization
RNN	Recurrent Neural Network

RunC	Low-level container runtime
SLA	Service Level Agreements
SRVM	Single-Root Virtual Machine
TPU	Tensor Processing Unit

CHAPTER 1

INTRODUCTION

Since software emulates components of a system, virtual machines have long been the primary way of delivering virtualization in the cloud. Virtualization permit activities to be performed in isolated environments, allowing for greater consistency because an emulated one abstracts the entire underlying system [6]. Containers have emerged as a viable alternative to virtual machines in the past few years. Containers have previously existed, but they witnessed a significant surge in popularity when the container framework Docker has introduced in 2013 and the capabilities that allowed users to quickly construct, distribute, and build upon each other's containers, which helped its growth since users could utilize pre-existing containers [7]. Containers are frequently viewed as lightweight virtual computers with short boot times and low resource consumption [8]. One significant reason for this is that containers, unlike virtual machines, run on the host machine's kernel, as shown in Figure 1.1. This is advantageous in multi-tenant cloud providers and data centers since each bare-metal system is likely to run more instances, and instances may be launched or restarted more effectively, resulting in a more excellent quality of service. Migration is a critical technique in the context of virtual machines and containers [9]. The process of moving an instance of a container that is in running state across hosts is known as migration [10]. Depending upon the nature of the task or according to the demand of the customer, a container migration can be live or non-live. While moving an instance, live migration means the user is unaware

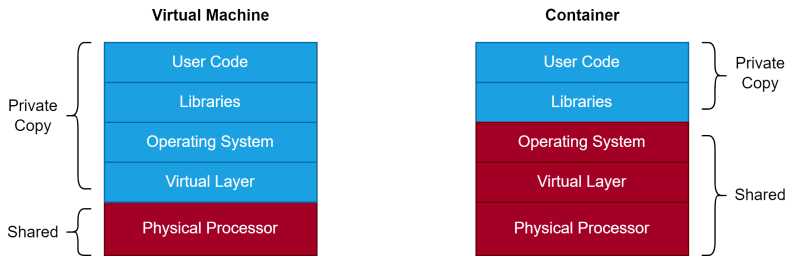

Figure 1.1: Difference in the architecture of of virtual machine and container

of this migration [11]. The state of the container is migrated prior to the container migration [12]. This technology is critical in various virtualized settings because it allows instances of virtual machines or containers to be transferred across hosts while retaining state, enabling effective load balancing and more straightforward maintenance with minimum impact.

1.1 Open Container Initiative

Containers have been around since the chroot system function was added to the Linux kernel in 1979. By enabling programs to alter their apparent root directory, this system function facilitated the isolation of file system hierarchies.

Nowadays, there are several process isolation methods available for Industry 4.0. Many of them are made possible by a Linux kernel feature known as namespaces. The Namespaces enable the isolation of distinct aspects of a process within a namespace. It restricts the process's awareness to its current namespace or nested namespace. The available namespaces are Cgroup, IPC, Network, Mount, PID, Time, User, and UTS. Furthermore, the use of system resources by a process, such as central processing Unit (CPU) and memory, may be restricted and monitored using cgroups, a component of the Linux kernel. This allows us to construct container images made up of layers that individually alter the underlying file system utilized by the container, and then share these images according to the requirement.

Linux Foundation started a project with Docker in 2015. This project is named as Open Container Initiative (OCI). In OCI, the migration process of the container has two main parts: container runtimes and container images, as you can see in Figure 1.2. To manage these two container runtime interfaces (CRI) is required. CRI supports low-level container runtime (RunC) and container daemon. Daemon is taking care of storage

and libraries, and on the other side, Runs is handling container runtime. Daemon will generate the pull request for the required container images from a registry. They work together to get the required libraries and manage container runtime. RunC can initiate the container process by communicating with the host kernel. It passes the command to the kernel to startup the process in a particular namespace, Pid, Cgroups, etc.

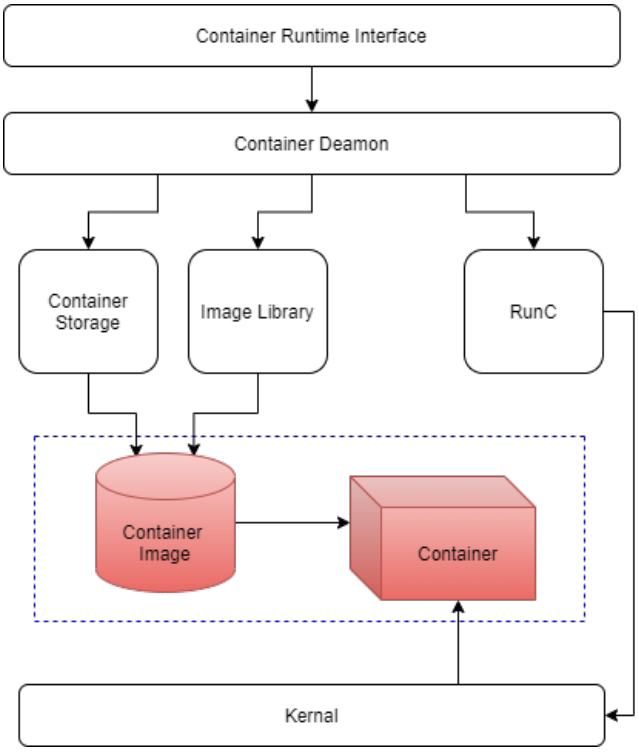

Figure 1.2: Container run-time interface

The main purpose is to standardize the containers for global acceptance with a complete architecture.

1.2 Migration Techniques

Container gained popularity when Docker introduced its natural flexibility [13]. It can run in the kernel of the host's operating system and be moved to another operating system without the bundle of system dependencies. It can be done by using the concept of namespaces as discussed in 1.1. Checkpoint and Restore in userspace (CRIU) has provided the feature to make a checkpoint of a running container and migrate it to another host [14]. The stateless migrations are not concerned with moving any state

across the hosts throughout the migrations, but stateful migrations are. As a result, stateless migration procedures are typically fairly straightforward, with the only steps necessary to conduct a stateless migration being to start a new container on the target server and then delete the container on the source host [15]. Because of the ease of stateless migrations, the migration mechanisms presented are all stateful.

Furthermore, these states are divided into runtime state and permanent storage. The runtime state is generally composed of volatile data, such as CPU state, open file descriptors, and memory pages, lost when the container quits. Permanent storage generally comprises data not destroyed when the container is shut down, such as volumes mounted inside the container. The following migration strategies are primarily concerned with runtime states because shared volume approaches often manage permanent storage.

1.2.1 Cold Migration

This method is likely the easiest, but it has the most significant disadvantage due to its lengthy downtime. Once the container dump is completed, the container is immediately terminated [16]. The dumped state is then transferred to the destination host. The container will be restarted with the same state from the received dump. The container is simply terminated, its state is dumped, destination host will get that. The container is restored with the received state dumped state at the destination host. As a result, long overall migration time and downtime. This approach is frequently despised, as it offers minimal benefits other than automating the migration process and transferring a small quantity of data [17]. It is also known as offline migration.

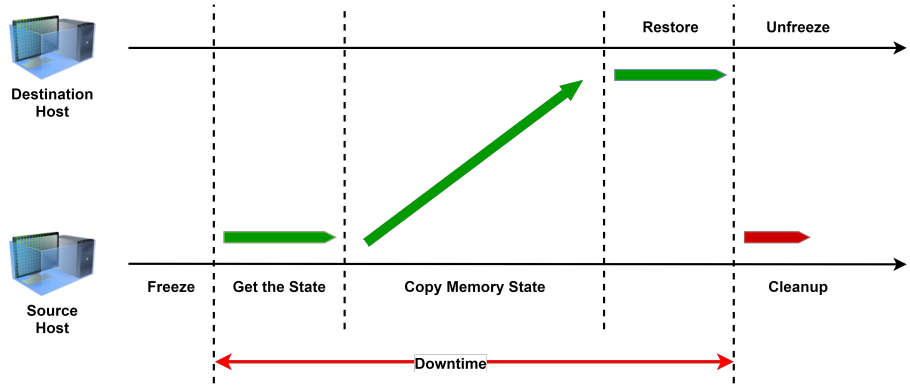

Figure 1.3: Phases of cold migration process of containers [1]

The main drawback of this cold migration is the long downtime. The services provided by the container will be stopped for a long duration which is not accepted according to the industry standards.

1.2.2 Pre-copy Migration

It is suitable for live migrations. While a container is running on the source host and it is decided to migrate a container to the destination host, the pre-dump phase will be initiated, and it starts transferring state and the memory pages related to the container [18]. During this process, the container is still running on the source host. This pre-dumped state often contains only the container's memory pages, not the entire container's state. It may also contain additional running state data. The container state may be modified again while transferring the state in the previous round and marked as updated with dirty bits to avoid any conflict in the complete state of the container [19].

An iterative dump will be done to send the incremental state. Iterations may vary depending upon container dump size or threshold. When the container's iterative dump state is completed, the last transfer will be initiated and called a final dump. It includes all changes made after the last iterative dump and is used to set the state of the container to the latest state [20].

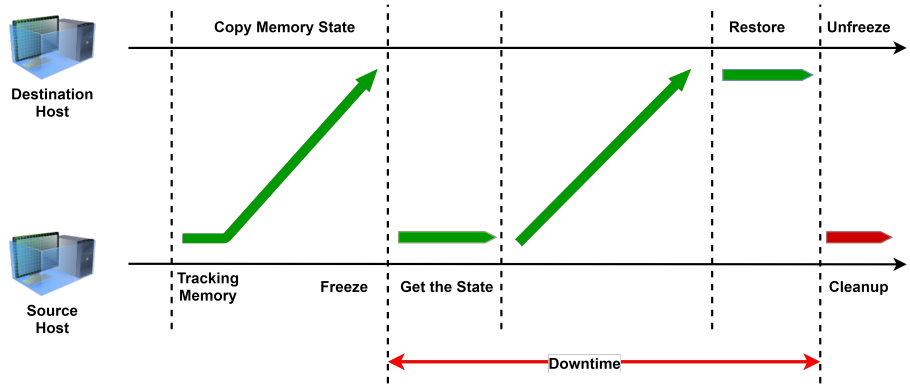

Figure 1.4: Phases of pre-copy container live migration [1]

In pre-copy, the downtime is less, but the total migration time may be longer than in cold migration. If the data transfer during the iterative phase can be controlled so that re-transmission of pages is minimized, then this migration approach is best suited for container migration with low downtime.

1.2.3 Post-copy Migration

Just transferring the state and restoring the container would be the same as cold migration. It stops the container when a migration request is granted and then dumps its state to the destination host. Instead of sending the complete state, it just sends the necessary execution state to start the container at the destination [21]. When the container starts at the destination host then starts generating page faults according to the requirement. The source will handle the request for a demanded page, and the required page will be transferred to the destination. Once all the required pages are transferred, the dump copy can be removed from the source. There are some major drawbacks:

1. If the target host crash in the process of demand paging, then the latest state of the container becomes unrecoverable.

2. If the source host crashes, then the container can not access the remaining pages.

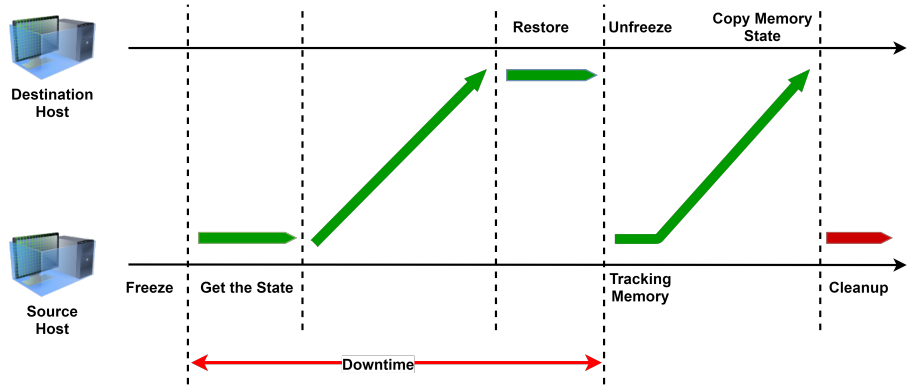

Figure 1.5: Phases of post-copy container live migration [1]

The post-copy and cold migration are not dependent on the rate of update of pages, but in pre-copy, this is then a factor affecting the performance. Downtime will be more in cold and post-copy migration. Considering the quality of service, our model prefers pre-copy because of its minimal downtime. Furthermore, different techniques are applied on pre-copy to reduce re-transmission of pages, and it can be further reduced.

1.2.4 Hybrid Migration

As per the discussion in 1.2.2 and 1.2.3, both the migration schemes has their own limitations. As an alternative, combining both techniques is known as hybrid migration.

First, the source container's state is pre-dumped and transferred while the source is still operating [22]. The source container is then stopped at the source, dump the current state, and transmitted to the destination to restore the container. When the container is restored at the destination, it only needs the modified pages instead of all the pages.

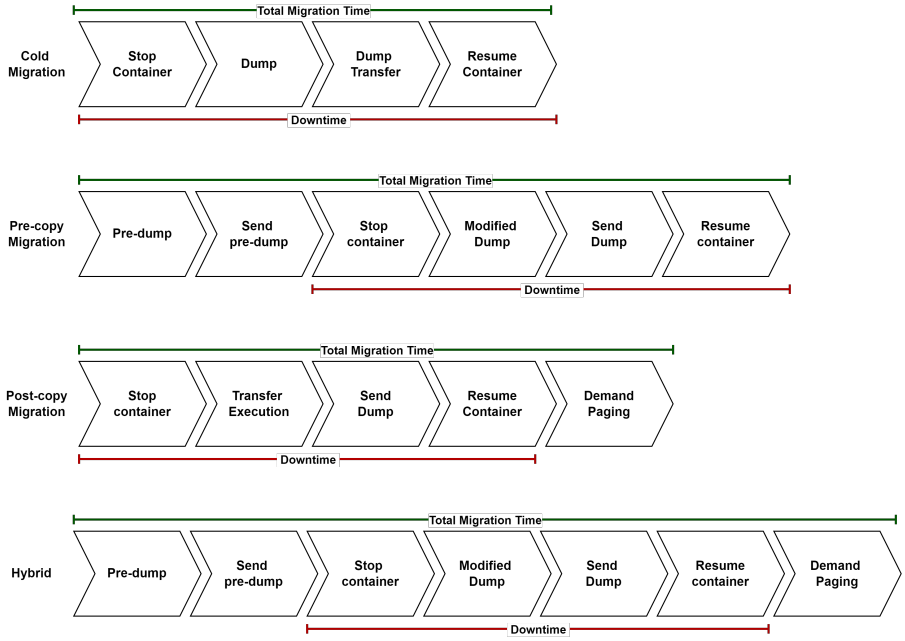

Figure 1.6: Phases of different migration techniques of container [2]

Compared to the pre-copy approach, the rate of update of pages will not affect the downtime. It affects only the total migration time. The drawback is still the same as discussed in post-copy, but the chances of such cases are rare in this approach. As discussed about these four types of migration techniques, it is imperative to understand what type of data is transferred. To understand this, first we should know the phases of these migration techniques as shown in Figure 1.6. In this scenario, the focus is on three main phases: pre-dump, dump, and Demand paging.

1. Cold migration: Nothing will be migrated in pre-dump. However, in the actual dump, the state of the container and memory pages are transferred and complete the migration process.

2. Pre-copy: The state of the container and the memory pages are transferred in pre-dump, and the iterative phase is initiated to send the dirty pages.

3. Post-copy: There is nothing to transfer in the pre-dump phase and execution state

transferred in the dump phase. According to the request generated, all the memory pages will be transferred in the demand paging phase.

4. Hybrid: The state of the container and the memory pages are transferred in pre-dump, and the iterative phase is initiated to send the dirty pages. It also transfers the dirty pages in demand paging.

The actual number of phases is different depending on the type of migration. The total migration time and downtime depend on the phases.

The proposed pre-copy technique addresses the issues of container migration. The main objective is to minimize the amount of data transfer during the iterative dump.

1.3 Checkpoint and Restore in Userspace

The process of storing the runtime state of any running application and restoring the same at a later stage is called checkpoint and restore [23]. It can be done on the same machine or at a remote host. This technique is used to recover from unusual crashes during the runtime of sensitive applications. After a particular time gap, it automatically stores the application's state [24].

In the last few years, demand for containerized applications has been increasing continuously and is widely accepted in cloud computing [25, 26]. Checkpoints and restore are now very useful in container migration to make secure migrations. This technique is introduced by checkpoint and restore in userspace. Live container migration is successful only with CRIU.

Checkpoint and restart allow a running container without a reboot to be transferred from one host to another host. This feature of container migration is referred to as the live migration of containers. In this process, steps should be followed:

1. Get access to the disk of the container file system.

2. The full state of the container, including all resources and processes, is stored in a disk file called dump.

3. The file will be transferred to a second host.

4. On another host from the dump, the container is restarted.

Using live migration is a better option as it provides high availability, dynamic load balancing, and fault tolerance.

1.4 Types of Container

System containers such as Linux Containers (LXC) are similar to virtual machines, as they share the host operating system's kernel and offer isolation of userspace. However, hypervisors are not used in system containers. It allows you to install various libraries, languages, databases, etc. Services running on each container use resources allocated to that container alone. System containers allow you to run multiple simultaneous processes, all under the same operating system. This improves efficiency and provides the advantages of VMs, such as carrying out several processes, along with modern container benefits, such as improved portability and fast startup times.

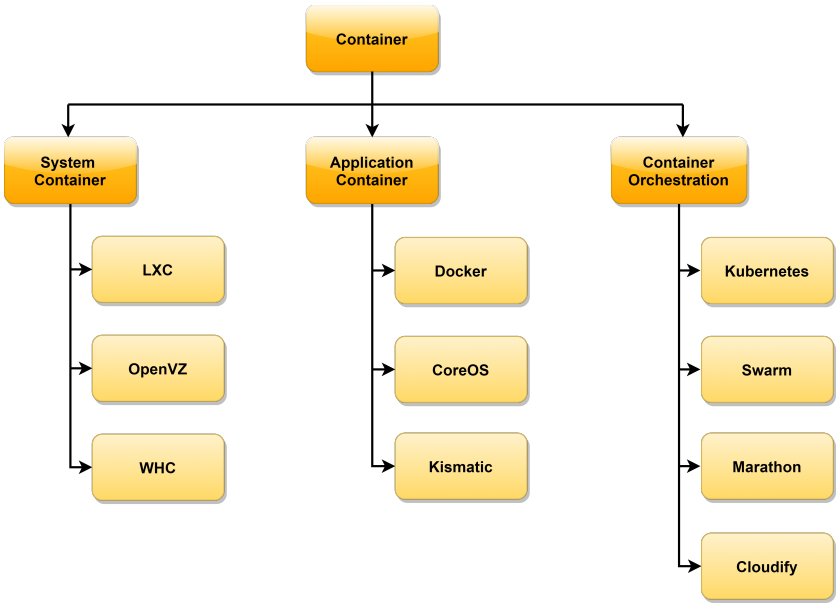

Figure 1.7: Types of container

Configuration containers encapsulate the application's data, dependencies, and libraries to operate on an operating system. Application containers enable the consumer to build and operate a single container for several individual programs or different resources that shape one program.

Nowadays, a number of container orchestration tools are available for the overall management of containers. Some of them are mentioned in Figure 1.7. A container that

runs a complete operation is a system container, and the application containers are used to run isolated application software/processes.

1. To achieve the first objective, the meta-heuristic approach has been applied. The PSO model has been selected for predicting the memory pages to be migrated to the destination host in the pre-dump phase of pre-copy container migration. The proposed model has been developed and analyzed using the container-Cloudsim toolkit with heuristic model libraries. This model is used to decrease the size of the pre-dump, which is directly proportional to the performance of migration.

2. To achieve the second objective, the short-term predictive approach is applied. The model applied for prediction in the iterative dump is the LSTM. Because it is iterative, the model is applied many times depending on the number of maximum iterations and epochs. It will make the decision based on the initial input generated from the first objective, then analyze the time series prediction and produce the list of pages to be migrated in every iteration.

3. To achieve the third objective, a memory reusing mechanism was designed with a page recovery model. It handles the page fault after migration to keep the container in the running state. When the container is migrating back to the same host, then the image stored on the source host for page recovery is reused and avoids the transmission of identical pages back to the host. The simulation is performed on a container-Cloudsim toolkit with executor libraries.

4. The proposed model has been evaluated using a different batch size of the container and with variable threshold levels. The main evaluation parameter is the amount of data transmission, but the model is evaluated on other effecting parameters like migration time, downtime, and dump time.

1.5 Research Objectives

1. To design a predictive algorithm for initial phase of pre-copy container migration to reduce pre-dump size.

2. To design an algorithm for iterative phase of pre-copy container migration to minimize the page transfer rate.

3. To design a recovery algorithm, which provides page recovery to handle failures.

1.6 Thesis Contribution

To give the answer to defined research questions, the thesis contribution is mentioned as per the following:

- A taxonomy and literature survey of pre-copy technique and other container migration techniques in cloud.
    1. A detailed investigation has been done to study various existing migration techniques in cloud computing.
    2. The mentioned techniques classification has been done as per the common characteristics.
    3. Future research direction in the area of container migration is presented.
- A meta-heuristic approach to predict the memory modification in the pre-dump phase of migration.
    1. The prediction based particle swarm optimization model.
    2. A memory classification technique based on dirty bits and updation rate.
    3. A predictive pre-dump approach for forecasting active memory of container.
- A prediction model for iterative-dump to gain the highest accuracy in short-term prediction.
    1. A LSTM based prediction for container migration.
    2. The design and development of prediction approach for minimizing the re-transmission of pages in the iterative phase.
    3. The design of a predictive algorithm to get the container memory status
- A Page recovery technique for container migration to recover from failure.
    1. A recovery technique to handle page fault after migration.
    2. A memory reusing technique is designed to minimize the dump size.

3. The reusing technique identifies the container memory to be discarded from the dump.

1.7 Thesis Organization

The structure of the thesis and its dependencies are shown in Figure 1.8. Chapter 2 is related to the literature survey and taxonomy of container migration techniques. Chapter 3 is focused on the memory prediction of containers in cloud computing. Chapter 4 is presented the iterative prediction technique for the iterative dump. Chapter 5 is presented the page recovery technique and the memory reusing technique in cloud computing. In detail, the organization of the thesis is as follows:

- Chapter 2 presents the methodological survey and taxonomy on migration and memory dump techniques for containers in cloud computing. This chapter is partially derived from:
  - **Gursharan Singh,** Pooja Gupta, "A review on migration techniques and challenges in live virtual machine migration", *5th International Conference on Reliability, Infocom Technologies and Optimization* (Published), 2016. (Scopus)
  - **Gursharan Singh,** Parminder Singh, "A Taxonomy and Survey on Container Migration Techniques in Cloud Computing", *Sustainable Development Through Engineering Innovations* (Published), 2021. (Scopus)
- Chapter 3 describes the proposed model for memory prediction of the pre-dump phase. This chapter is derived from:
  - **Gursharan Singh,** Parminder Singh, Babar Shah, Farman Ali and Daehan Kwak, "A Lightweight Migration Technique for Uninterrupted Health Care System in Cloud", *Computational Intelligence and Neuroscience* (Under Revision), 2022. (Scopus, SCIE 3.63 IF)
- Chapter 4 presents the prediction-based proposed iterative-dump pre-copy technique to minimize the re-transmission of pages during container migration. This chapter is derived from:

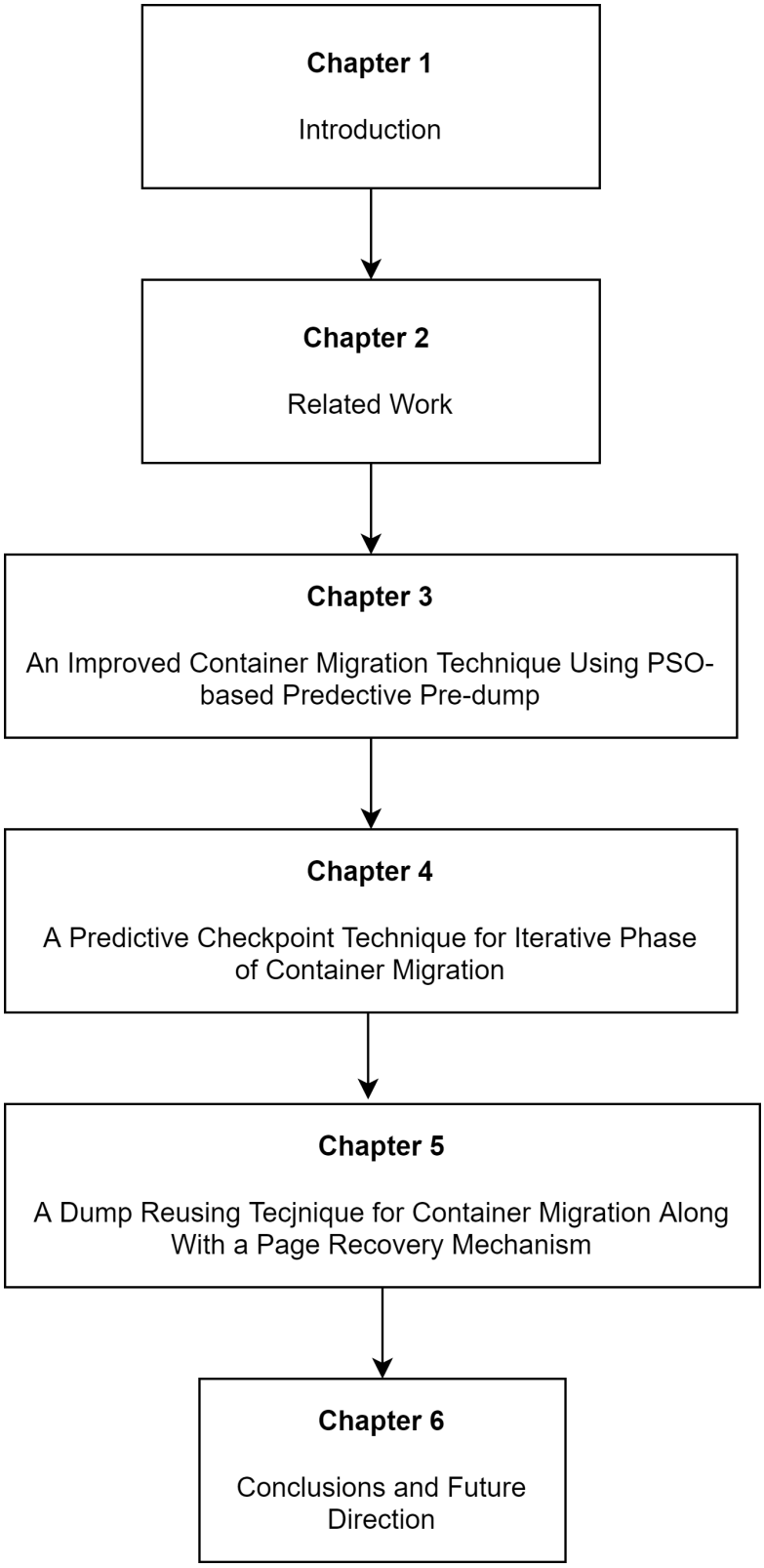

Figure 1.8: Chapter wise thesis organization

  - **Gursharan Singh,** Parminder Singh, Mustapha Hedabou, Mehedi Masud and Sultan S. Alshamrani, "A Predictive Checkpoint Technique for Iterative Phase of Container Migration", *Sustainable Engineering and Science, Sustainability* (Published), 2022. (Scopus, SCIE 3.25 IF)

- Chapter 5 presents the page recovery technique for container migrations along with the memory reusing approach from the source dump file. This chapter is derived from:

  - **Gursharan Singh,** Parminder Singh, Container Migration Technique to Minimize the Network Overhead with Reusable Memory State, *International Journal of Computer Networks and Applications (IJCNA)* (Published), 2022. (Scopus)

- Chapter 6 presents the conclusion of the thesis finding and introduces the possible future directions.

CHAPTER 2

RELATED WORK

The present container migration approaches in cloud computing are investigated and reviewed in this chapter from various perspectives, including migration methodology, environmental architecture, tools used, scope, and performance assessment. Separate taxonomies for each perspective are offered based on a thorough review of the literature. A thorough investigation of existing methodologies is carried out. Finally, the research gaps that need to be filled in order to enhance the container migration paradigm are identified and highlighted[1].

2.1 Introduction

In the live migration of containers and with the increasing popularity and wide adoption, page recovery is an important factor affecting performance. The container's size is already smaller compared to virtual machines, but still, the performance of containers can be enhanced further. The most influential factor is the amount of data migration over the network, which directly affects the cost and performance of container migrations in a lightweight environment.

[1]This chapter is derived from:

Gursharan Singh and Parminder Singh. "A Taxonomy and Survey on Container Migration Techniques in Cloud Computing." In Sustainable Development Through Engineering Innovations, pp. 419-429. Springer, Singapore, 2021.

The checkpoint and restarting of a system have certain preconditions that the OS needs to provide. Container infrastructure contains: Namespaces (Process ID (PID), User, group Id, Network configuration, Mount points, Inter-process communication, Hostname), and the cgroup subsystems (CPU cores, CPU time and utilization, Limit memory consumption, Limit and observe I/O accesses).

- PID virtualization –guarantees that a process can be allocated the same PID during restarting as before.
- Isolation of the process community–to ensure that the interaction between parents does not lead to external containers.
- Network separation and virtualization–to ensure the isolation of all network connections from the host operating system and all other containers.
- Virtualization of resources that is hardware-independent and can restart the container on another server

During the design phase, considering other requirements:

1. A complete set of resources like delegated resources, register set, network connections, address space, and other private information of each process is used by the system to test and restart a container.
2. Size of file dump should be reduced, and all behavior between a freeze and a resume streamlined so that the service is as late as possible.

2.2 Taxonomy and Survey on Container Migration Techniques

For a better understanding of containers, various techniques has been studied and come up with a container taxonomy with six different parameters (Architecture, Tools, Purpose, Scope, migration technique, and Evaluation) Figure 2.1.

Live migration service for container incorporates CRIU-based memory migration, which reduces downtime in migration [10]. Containers will immediately re-operate on the target host with a union view of the data between the source and the target hosts while gradually transferring the disk state in the background. An optimized pre-copy

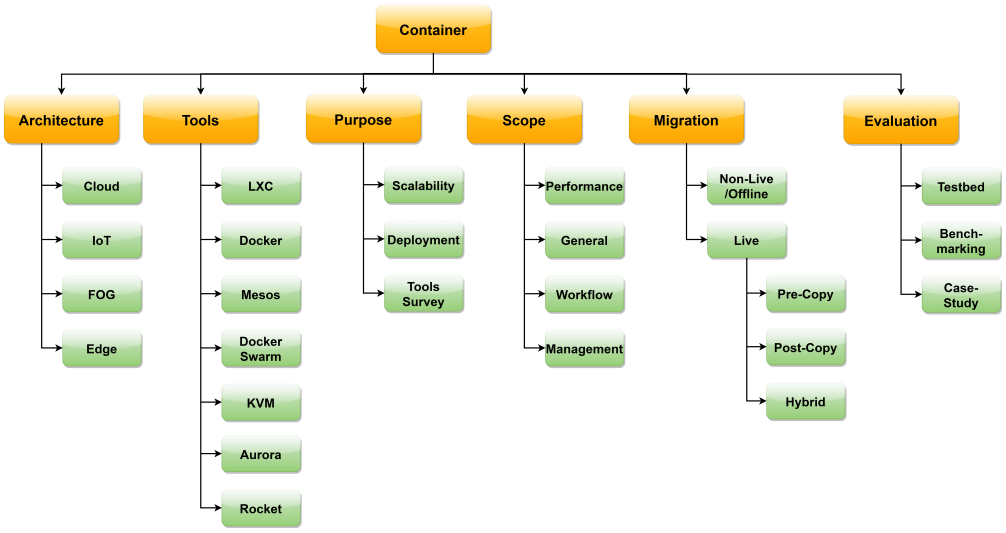

Figure 2.1: A taxonomy on container in cloud computing

algorithm that introduces a Gray-Markov prediction model, in which the memory pages are added to the Hot Workspace by the prediction model with high re-modified rates, and the stop copy is carried through on these memory pages designed by [27]. Experiments show that iterations and downtime are reduced by the Optimized Pre-copy Algorithm (OPCA). [28]. Presented the checkpoint and restart functionality for OpenVZ containers.

The dump file size is reduced, and all activities between freeze and resume should be streamlined so that the service time is sustainable and as short as possible. Instead of initiating the migration based on thresholds, this migration is triggered by predictions, determining performance metrics, addressing device failure, and preventing excessive migration [29]. One of the methods to minimize the extra overheads of migration is to identify the relatively stable memory pages. When the size is enormous, sturdy pages should get the preference to reside in cache [30]. The detailed comparison of various techniques based on the proposed taxonomy has been discussed, as shown in Table 3.1.

1. Characterize the memory pages actions based on the page flags accessible on the proc file system of a Linux kernel.

2. Propose a forecast system that can be used to predict relatively stable memory pages.

3. Analysis of the memory and the proposed prediction method in two applications: memory deductions and live migration.

Table 2.1: A taxonomy based comparison on various container migration techniques

Ref.	**Archi - tecture**	**Scope**	**Purpose**	**Tools**	**Evaluation**	**Migra -tion**	**Technique**
[31]	Cloud	Performance	Deployment	LXD, CRIU	Tested	Live	Pre-Copy
[Bhardwaj and Krishna 2019]	Cloud	Performance	Deployment	LXD/CR	Tested	Live	Pre-Copy
[32]	Cloud	Performance	Scalability	Kubernetes	Tested	Live	
[Al-Dhuraibi et al. 2018]	Cloud	Performance	Scalability	Docker, RUBiS, KVM	Bechmarking		
[Al-Dhuraibi et al. 2017]	Cloud	Performance	Scalability	Kubernetes, Graylog	Bechmarking		
[33]	IOT	Performance	Deployment	Docker	Tested	Live	Pre-copy
[27]	Cloud	General	Tools Survey	Docker , Spark	Tested	Live	Pre-Copy
[Nadgowda et al. 2017]	Cloud	General	Tools Survay	LXC, CRIU	Tested	Live	Post Copy
[34]	Cloud	Performance	Tools Survey	Docker, TOSCA	Tested	Offline	
[28]	Cloud	General	Deployment	OpenVZ, LXC	Tested	Live	Pre-copy
[35]	IOT	General	Tools Survey	OpenStack, Kubernetes	Tested	Live	Pre-copy
[Raghunath and Annappa 2017]	Cloud	General	Tools Survey	XEN, RUBiS	Tested	Live	
[36]	Cloud	General	Tools Survey	Kubernetes	Tested	Live	Pre-copy
[37]	Cloud	General	Tools survey	CPLEX, LXC	Tested	Offline	
[Elghamrawy et al. 2017]	Cloud	Performance	Scalability	LXC	Tested	Live	Pre-Copy
[38]	IOT	General	Deployment	Docker	Tested	Offline	
[39]	Cloud	General	Deployment	OpenVZ, LXC	Tested	Offline	

The updated pages in the last iteration but not modified in the current iteration are separated in the first phase. Division of pages is done as per the second phase into two types, i.e., pages with high dirty rate and normal depending upon the updation in the last some iterations. The filtered pages from the first phase are checked for the number of times the page modification has been done in history. The page that is modified more times is required not to be sent during the current iteration [40].

Another container migration scheme is Application Oriented Docker Container (AODC) to reduce Docker container application cost of deployment and to allow automated scaling as the workload of cloud applications varies [37]. Pre-copy algorithms eliminate repeated transfer of dirty pages based on the probability of prediction of memory pages becoming dirty. When the dirty pages increase significantly, the possibility will fail and raise sustainability issues. An adaptive bandwidth technique that improves transmission bandwidth for decreasing transmission time when the dirty pages rate suddenly increases was proposed [41]. It reduces the iterative transfer total time as well as downtime.

Containers are handy in edge computing. The edge computing platform is based on the concept of containers in which four key criteria were evaluated:

1. Deployment and termination
2. Management of resources and services
3. Fault tolerance
4. Caching

Docker container used in edge computing platform offers fast implementation, small footprints, and good performance based on our assessment and testing [42]. The containers are more compact, powerful, and easier to manage based on lightweight virtualization techniques. For live transportation of docker containers, both downtime and total migration time can be reduced, helping to increase user experience with data center services. The Docker container is migrated from the source host through logging and replaying in the iterative phase. An optimized pre-copying approach based on time series recognizes and transmits frequently updated dirty pages in the past and future

eras to reduce unwanted, repetitive dirty page transmission. This technique can reduce the overall migration time significantly [39].

The pre-copy migration methods was proposed, including post-copying, pre-copy, pre-copy methods, etc., are presented to solve the problem of the too-long pre-copy migration period [36]. Pre-copy approaches can reduce the total migration time but lead to more extended downtime. The suggested method can choose the best time to carry out memory migrations according to the features of memory pages to reduce unnecessary migrations and the total time of migration. Using the associated dirty rate to enhance live migration, you choose which pages you want to migrate in the iterative pre-copy stage or the stop-and-copy process.

Al-Dhuraibi et al. [12] introduced the container that autonomously supplies vertical elasticity. The autonomous computing concept scales up and down both memory allocation and CPU utilization of the container according to the workloads, which reduces the running cost of containers. This vertical elasticity improves the performance by 37.63 percent compared to Horizontal elasticity by improving product quality and maximizing resource usage. In the overall process of migration the downtime is reduces with a huge difference by managing resources compared to auto-scaling in Kubernetes.

Ma lele et al. [33] introduced a new service management method that automatically switches offloading services as the mobile customer travels to the nearest edge server. His study revealed that hand-off service is accomplished by container migration. Further, they identified a vital performance problem during container migration. A migration approach was proposed to minimize overhead file synchronization without relying on the distributed file system, further based on a systematic analysis of container layer management and file stacking. Testing results show that the framework reduces the overall time of service delivery compared to modern hand-off systems designed for edge computing platforms. This approach improves service transfer across edge servers by leveraging the container layered storage system to enhance migration efficiency. The technology helps the cutting-edge computing environment continuously offer download services with low-end latency while retaining high consumer mobility by using the layered file structure of containers to remove unnecessary transfers of a redundant and substantial portion of the application file system. When the base memory image is transferred before the conversion and the incremental memory gap is only

transmitted by migration, the total transfer size decreases.

Authors in [39] illustrated the Pre-copy phases and how dirty pages are found when migrate to live. Experimental results showed that all received packets could be efficiently migrated by (SRVM) to ensure the integrity of application data as SRVM do not maximize here. The equivalent framework helps to minimize container migration downtime. Another approach is proposed by [43] to work on resource containers and safety containers for standard, time-shared systems. They present the design and implementation of Linux-VServer as a representative instance of container-based systems. It also compares Linux-architecture VServer to current Xen generations and illustrates how Linux-VServer supports comparably and superior device efficiency.

The method based on compression and deduplication of data is elaborated by Luo et al. [44]. The authors used the RLE approach for migration to eliminate duplicate memory data using the runtime storage image identity. For page similarity computations, hash-based fingerprints have been used, and for implementation, LRU Hash tables FNHash and FPHash were utilized. The study made it clear that migration efficiency improves in space with overhead CPU resources.

The basic technique of pre-copy transfer memory pages repeatedly at high replacement rates during the incremental copying process is provided by Ansar et al. [45]. The paper presented thus an OPCA, which includes a Gray-Markov model. They found out the modification rate of every memory page based on the proposed prediction model and shortlisted the pages with a high speed of modification. Results showed that the total migration time and downtime are decreased with the decrease in the number of iterations. Moreover, the resource utilization is also increased by OPCA. In turn, the use of resources is enhanced, and the user experience is improved. Experiments show that OPCA minimizes iterations and downtimes, raises resource usage levels and improves user experience. Additionally, it optimizes the conventional pre-copy algorithm to minimize downtime and displays Markow's Gray model and the current space definition. To get better results, the work area is divided by calculating the re-shaped probability of the memory tab in which Highly changed memory pages are stored in the HW and copied throughout the final process. The WW shall therefore be used to reduce errors.

The live container just-in-time migration service built-in line with OCI principles was introduced by Nadgowda et al. [10]. A new file system and supplier-agnostic migra-

tion software voyager provides CRIU memory-based migration and union mount data federation capabilities that minimize migration downtime. Containers are re-operated on the server with combined data between the source and destination host, but the status of the disk is increasingly transferred. Design and implementation employ a data federation in several sources and host goals to allow this application to restart on time at a host with remote readers and local-write access.

Molto et al. [34] describes an open standard tool-based workflow for implementing consistent HDCI applications in VM as well as Docker containers. DevOps activities have been introduced by using and enhancing the TOSCA standard and have resulted from incoherent objects used to execute applications on different platforms. Ansible Roles' implementation and automated building capabilities allowed the Docker images to be distributed simultaneously to containers. Applications can be automatically deployed on vanilla VMs using DevOps. Additional modifications to the Simple TOSCA YAML Profile 1.0 will support the implementation of new non-normal formats in other scientific communities in future work.

Mirkin et al. [28] presented the OpenVZ container checkpoint and resume function. This function makes it possible for the container to scan and restart apps and network connections on the same host or an additional host transparently. Checkpoints and restarts are implemented as loadable kernel modules plus various userspace utilities. The fact is established that the size of the dump file should be minimized to reduce the service delay as all the operations between stop at source and resume on destination should be streamlined.

Elghamrawy et al. [30] described that there is a significant divergence between various prediction mechanisms in the present memory pages' behavior. They describe the behavior of the memory pages based on the prediction method for predicting reasonably stable memory pages. That is the void they hope will be reduced. The memory page characterization to prioritize certain pages with live migration because these pages will progressively be updated in the following iterations. The non-dominated Sorting Genetic Algorithm (GA) is suggested to optimize containers' assignment and management elasticity to the extent of the proficient results on other cloud management problems achieved through this algorithm. Clusters, containers, micro resources, and four optimization goals are supplied with a model. Experimental findings indicate that this

approach is a solution to the issue of container allocation and elasticity, achieving more robust ethical standards than container management policies at Kubernetes.

2.3 Container Memory Dump Techniques

From the memory compression perspective, there are several drawbacks to consider, and suggest an efficient scheduling model [12]. The container is scheduled on the corresponding server to accomplish load balancing. Likewise, dynamic migration core implementation scenarios are elaborated [35]. To recover large data structures from a catastrophe, a wide-scale dynamic migration mechanism relying on network conditions, migration policies, controller categories, and memory allocation needs to be built. They forecasted the often-changed memory pages using the prediction model of the time series and then moved them in the last copying round to the destination container [47]. The TPO method is to reduce the number of pages transferred in each iteration and optimize the migration process.There are several techniques proposed to minimize the dump size as illustrated in 2.2.

C Puliafito et al. [15] have carried out a detailed evaluation of various migration techniques under four parameters like total migration time, downtime, dump-time, and amount of data transferred data. They recognized several situations and suggested which strategy would be most suited to them. The findings demonstrate that the cold migration suffers from significant downtime, whereas hybrid migration suffers from a longer overall migration time. Pre-copy and post-copy migrations may thus be the best alternatives under specific scenarios.

Several standard memory compression methods are mentioned and studied from a memory standpoint, including RLE, Huffman Coding, and a novel strategy for minimizing migration time. There are numerous difficulties to consider regarding memory compression, and an effective scheduling strategy is suggested [12], but the compression overhead affects migration time. To provide load balancing, the container is scheduled on the relevant server. The core implementation of dynamic migration scenarios has been developed for IoT [35], the main motive is resource provisioning. It can be more effective if it is manageable to reduce memory transfer. A successful migration is needed to restore massive data structures following a disaster. Moreover, to reduce the data transfer, prediction methods can be applied [47]. It can be reduced further

Table 2.2: A prediction based comparison on various container migration techniques

Author /year	**Migration Technique**	**Prediction Method**	**Outcome**	**Platform Used**
[36]	pre-copy	Markov Prediction, Related Dirty Memory	Reduced Migration Time	MATLAB
[27]	pre-copy	Grey-Markov Prediction Model	Reduces downtime	Docker
[46]	pre-copy	Average Dirty page rate method	Reduces Migration time by 11.76 and Downtime by 13.94	XEN Hypervisor
[47]	pre-copy	() Model	Reduces Migration time by 19.16 and Downtime by 10.76	XEN Hypervisor
[48]	pre-copy	Bitmap, Shadow Paging	Reduce downtime by 3	XEN Hypervisor
[40]	pre-copy	Time series Method of Dirty Pages	Minimize data transfer and downtime	CloudSim
[49]	pre-copy	Linear Regression	Minimize migration cost	LXD, CRIU
[50]	Hybrid	Most Significant bit (MSB)	Reduce Memory Write cost	MATLAB
[51]	pre-copy	Extreme Learning Machines (ELM)	Reduce mean prediction error up to 99 per.	Google cluster trace
[52]	pre-copy	meta-model-based prediction	Reduce migration cost	MAPE with regression and LSTM

with LSTM as suggested by [53]. The reuse distance concept is used, and the changed memory pages are traced back during the copy process [54]. A model includes clusters, containers, and micro resources with four optimization goals. The experimental results show that this strategy is a solution to the issue of container allocation and flexibility, attaining higher ethical standards than Kubernetes container management regulations. Each method's implementation scenarios are examined in [55]. For containers, pre-copying is the predominant way of migration.

Elghamrawy et al. [30] described the fact that there is a significant discrepancy between distinct prediction systems in the behavior of current memory pages. That is the hole they want to fill. They characterize the behavior of memory pages using a prediction approach for relatively stable memory pages and using the memory page characterization to prioritize specific pages with live migration since these pages will be updated gradually in subsequent cycles. The GA technique employing the non-dominated Sorting Genetic Algorithm is recommended to maximize container assignment and management elasticity to the degree that this algorithm has produced good results on other cloud management challenges.

Mirkin et al. [56] The OpenVZ container checkpoint and resume mechanism were provided. This function allows the container to scan and restart programs and network connections. Checkpoints and restarts work with the kernel directly to reduce service delays and the size of the dump file.

Dump size can be further reduced for memory-intensive applications. Molto et al. [34] designed a hybrid distributed computing for VMs and containers for a better synchronization among hosts. In memory-intensive application, the repetition of memory may increase.

The live container just-in-time migration service was designed following OCI standards introduced by Nadgowda et al. [46]. Containers run on the server with the shared file system, and some are with the local file system. CRIU helps to perform a live migration of the container and ensures the restart on the destination host. A union mount file system and CRIU helps to reduce migration time and data transfer.

Luo et al. [44] developed a technique based on data compression and deduplication. The authors employed the RLE technique and the runtime storage image identity to reduce duplicate memory data. Hash-based fingerprints were employed for page similar-

ity calculations. LRU, Hash tables FNHash and FPHash were used for implementation. The efficiency of migration has increased in terms of space with the overhead of CPU resources.

During the incremental copying procedure, the primary approach of pre-copy transfer memory pages are duplicated repeatedly at high replacement rates is provided by Ansar et al. [45]. Thus, the article developed an OPCA model with a Gray-Markov model. They shortlist the pages based on modification rate, which decreases the number of iterations and other parameters. This increases resource utilization, which is managed by using a hot and warm working set to categorize the pages. Now, the pages only from the HW set will go through the process. Chronopoulos et al. [57] show the effective use of machine learning to build an artificial neural network for speech-language therapy.

Slominski et al. [58] Creates a repeatable architectural pattern to transition a legacy application to a cloud service. The workflow service verified the architecture because it was built with minimum changes to a traditional workflow engine that only serviced a single organization and wasn't designed to be a cloud service. To quickly build a multi-tenant, elastic, and highly available cloud service utilizing Docker containers to manage various engines for each tenant, as well as a cloudant persistent layer and a content-based router. The architectural pattern may be used by other legacy programs that need to transfer to the cloud.

Guerrero et al. [59] Based on the results gained with resource management optimization challenges in cloud architectures, a genetic algorithm technique is presented to improve container allocation and elasticity management using the non-dominated sorting GA II. The optimization technique improves system provisioning, system performance, system failure, and network overhead, and it achieves better objective values than the Kubernetes container management policies. The expenses of container-based live migration might be investigated.

Huan et al. [39] when using containers based on the lightweight virtualization approach, live migration time may be drastically reduced, which is extremely useful to data centers. This study described a live transfer of Docker containers using logging and replay. The downtime and overall migration time in the trial were both minimized using this method. Under three different scenarios, downtime has been reduced by

65%, 55%, and 44%, while overall migration time has been decreased by 27%, 47%, and 38%, respectively, when compared to the technique for typical virtual machines.

Du et al. [60] Memory propagation has been accomplished successfully in the suggested microwiper approach during live migration. It has two different strategies: ordered memory propagation and transfer throttling. Within VM memory, page rewriting speeds vary dramatically. Pages with greater rewriting rates have a much higher chance of being rewritten. If we send these pages, they will very likely get dirty again, requiring re-transmission. The pages with the lowest rewriting rates, on the other hand, will be the greatest candidates for priority transfer. They calculated the memory stripes rewriting rates from the previous iteration in each iteration, then arranged the memory stripes according to the rewriting rates. They shifted dirty pages of stripes in that order and dynamically estimated network bandwidth; the sum of transferred stripes rewriting rates will be calculated and compared to network bandwidth estimation; the next iteration will begin immediately if the rewriting rate calculated after accumulation is greater than the estimated bandwidth.

Chen et al. [41] The pre-copy method repeatedly transfers memory pages. Iterative means "three rounds" in pre-copying, which means that the pages that were relocated in round n were those that were modified in round n-1. The more often alteration pages are sent, the more likely they will be sent. As a result, the total volume of transmission has increased. If the page changing often judgement was made in the previous round but not during the current iterations to skip judgement, it would need to be delivered again. There's a good chance that these pages may change again in the next edition. As a result, when page changes are frequent, the pre-copy approach does not operate well. As a result of these flaws, this study recommends delaying the transfer of unclean pages. This approach adds a new bitmap to delay for flagged pages that were changed both in the previous iteration and the present iteration.

Yanqing et al. [61] as they mentions, that without a workload, the total migration time will be directly proportional to the size of the VM memory. The address of metadata will be converted to the address of the real page. Then, they got page numbers of complete VM memory. Bitmap bits will be set accordingly. Before each pre-copy cycle, migration daemon would ask memory explorer for a bitmap. According to the bitmap, the migration daemon will only send one byte data to the destination daemon

for unallocated pages. The RLE technique will be used by the migration daemon to encode allocated pages. If encoding is cost-effective, the migration daemon will send one byte compressed in order for the destination migration daemon to decode compressed data. If this is not the case, it will transmit a single byte to the destination with no changes.

Bubai et al. [62] The source machine memory pages are divided into fixed-size groups known as blocks. The size of the block will be determined by the number of pages available on the system. Every page will have an array, the size of which is determined by the threshold value, which is the maximum number of iterations a machine can do. This array will hold the state of the page during each iteration. 1 indicates that the page has been updated, whereas 0 indicates that the page has not been changed. When a page was last transmitted to destination, the first member of this array will hold an iteration number. With the aid of the amount of 1's a block has, the total number of pages dirty throughout each iteration will be counted. Until the last iteration, the blocks will be arranged based on dirty counts. The block with the least dirty count (minimum of one) will be chosen. The array of pages with 0 in the current iteration will now be examined for that block with the least dirty count. Pages with a number of 0s equal to or more than the number of 1s in the array must be moved to the destination. From the time it was last submitted to the final iteration, this will be verified. When the page has been transmitted, the first element will now have an iteration number. This prevents the page from being resent if it hasn't been updated since the last iteration. If a block does not have a page that meets the criterion, the block with the second-lowest number of dirty counts is chosen. Only the pages with a low unclean probability (pages that do not change frequently) were provided during the middle round of iteration. As a result, the total number of pages that will be relocated to destination in the intermediate rounds will drop. When the number of repetitions reaches a certain threshold, the source VM is paused, all dirty pages that were not delivered to the destination are sent to the destination, and the virtual machine at the target is resumed.

Jain et al. The suggested strategy is divided into two parts. The first phase separates the pages that have been changed in the previous iteration but have not been amended in the current iteration. Pages are divided into two groups in the second phase, pages with a high unclean rate and regular pages, based on the updation in the previous iterations.

The number of times the page has been modified from history is verified on the filtered pages from the first step. The page that has been updated the most times must not be transmitted during the current iteration. Send page to target VM on the other hand. The transfer of a page is determined by a simple idea. The page will be unclean if the number of changes on it exceeds the number of un-modifications in the history record. It is implemented in the CloudSim simulator to evaluate the intended technique's performance and to compare it to an existing pre-copy strategy based on time series in terms of overall migration time and down time. The outcome confirms that the recommended technique [40] produces superior results.

Weiwen et al. looked at container migration strategies among cloud servers in order to reduce migration costs while preserving load balance. Both techniques are stated as optimization problems with constraints. To handle optimization difficulties across different time scales, they present the balance aware container placement and adaptive threshold container migration algorithms. In addition, container workload prediction is presented as a way to improve the approach. Experiments indicate that we can achieve load balancing while lowering migration costs. Further in [63], the suggested technique outperforms meta-heuristic algorithms such as PSO and computer resource optimization.

Mathematical modelling, heuristic, machine learning, and meta-heuristic are the four basic types of container scheduling algorithms. Machine Learning is the ideal solution for anticipating workloads and performance indicators because of its great capacity to validate the system to anticipate outputs based on prior data and training. In complex work contexts, such a view helps schedulers with better resource allocation while dealing with shifting user request rates [64].

Gundall et al. [65] provides a unique paradigm is offered that relies on both existing migration methodologies and virtualization technologies, with the primary goal of reducing service downtime. A test set is also used to examine the notion. The results suggest that the proposed strategy can achieve a reduced downtime. Furthermore, the overall migration time for the maximum performance option is in milliseconds.

Terneborg et al. [66] expanding container migration with a proposed method that supports fail-over and live migration, which means it might be incorporated into existing container tools. Furthermore, evaluation results are supplied, which may be used to

compare to an existing migration approach. They have discussed the current migration methodologies and metrics for assessing different migration approaches. They have accomplish a lower total migration time and downtime similar as of pre-copy migration [67].

Zhi et al. [68] intend to save cost on resources by using as little machine resources as feasible by using a suitable dynamic container migration capability, therefore cluster-scale layout of container has been the topic of this paper. A method is presented to decrease fragmentation, hence improving machine resource efficiency and achieving the cost-cutting aim. Experiments indicate that the method efficiently prevents fragmentation and reduces resource consumption in container layouts on a wide scale.

Zheng et al. [69] offers a scheduling technique for a two-level approach for container real-time resources. To decide on container migration, LTSM is utilized to estimate resource use and select the environment. In addition, for simulation trials, they used CloudSim, an open source programme. The results demonstrate that the method may increase global resource utilization of containers while lowering data center energy usage.

Yang et al. [70] to address the challenge in prediction accuracy, an online prediction approach called user trajectories is given. A scheduling algorithm is designed to identify servers based on user movement speeds and latency to reduce duplicate network traffic. The results of our tests indicate that the proposed prediction methods outperform the usual technique. It reduces network traffic by 65% while meeting task delay standards. Furthermore, it adapts to changes in the user's journey speed and surroundings to ensure service stability.

Chen et al. [71] In the container migration, the PSO is used for hyper-parameter adjustment in order to enhance the model's prediction performance. The findings of the experiments suggest that autonomous hyper-parameter adjustment can increase prediction accuracy. Meanwhile, in MSE, R2, and MAE, the prediction performance is better than the previous system without managing hyper-parameters by 19.3%, 4%, and 11.7%, respectively, compared to the existing system without managing hyper-parameters. Furthermore, the PSL beats other algorithms like as RNN, GRU, and LSTM in terms of prediction performance.

Dai et al. [72] predicting failure before it occurs is critical for making the cloud ser-

vice more effective. The ability to forecast defective nodes allows service to be migrated to healthy nodes, increasing service availability. To successfully handle this problem, proactive fault prediction approaches to forecast future failures can be employed. In this research, using time series data to forecast the failure in a cluster using the bidirectional LSTM model.

2.4 Memory Prediction

There are other techniques as well; those are suitable for predicting memory. Sobia Pervaiz et al. [73] conducted a detailed review of various Variants of PSO and highlighted the key features of every variant. It helps to identify the best-suited variant of PSO depending on the nature of the problem. Waqas Haider Bangyal et al. A unique quasirandom sequence termed the WELL sequence to initiate the PSO particles was used [74]. The velocity and position vectors of particles are changed in random order. The results show that the WE-PSO strategy outperforms the PSO, S-PSO, and H-PSO approaches. Three sequence strategies Torus, Knuth, and WELL was proposed by [75]. All these techniques are tested with low-discrepancy sequences. The result shows that the proposed methods outperform as compared to standard PSO and its other variants.

Migrating a container application can be accomplished using various container migration techniques, as shown in Table 2.3. Because of the container's limited lifespan, they used a pre-copy strategy to facilitate the migration procedure [27].

This research work carried out a review of various container migration techniques. The essential factor is the amount of memory transfer from the source to the destination host and the repetition of the same in an iterative phase of memory migration. As discussed in related work, some of the research work is carried out to predict memory change. With the help of different prediction algorithms, researchers try to decrease the number of pages transferred during the iterative migration phase. But the prediction of memory change will also help in the pre-dump phase of pre-copy container migration. Pre-copy is the most commonly used migration technique, and it is best suited for containers. As per the review of migrations techniques of containers, the pre-copy technique outperforms in most of the cases as depicted in Table 3.1 and in Table 2.2. Identification of research works related to container migration is recorded in the following parameters: Architecture, Scope of the research, Purpose, Tools used, Evaluation,

Table 2.3: A prediction based comparison on various container migration techniques

Ref.	**Migration Technique**	**Prediction Method**	**Achieved**	**Outcome**
[12]	pre-copy	RLE, Huffman Coding	memory compression	Compression overhead effects migration time
[35]	CloudIoT	LXC	vertical offloading	main motive is resource provisioning
[47]	pre-copy	ARIMA	Predict dirty pages, reduce and compress	can be reduced further with LSTM [53]
[54]	pre-copy	ARIMA	Memory forecast	can be reduced further using containers with LSTM [53]
[55]	pre-copy	LXD	Container resource management	Resource provisioning
[56]	pre-copy	OpenVZ	Incremental Checkpoint	dirty pages transmission can be minimized
[27]	pre-copy	Gray-Markov prediction model	reduce iterative cycle	it shortlist the active pages
[53]	pre-copy	LSTM and ARIMA	dirty page prediction	600 times faster prediction time than ARIMA
[76]	pre-copy	LSTM	reduce the size of iterative-dump	amount of data transfer is reduced by 31.04%

Kind of migration method opted, and migration techniques.

Suppose the source has a running container that will be required to be shifted to the destination. Pre-dump is the very first step to initialize the pre-copy container migration. In this phase, the complete set of container memory is transferred to the destination host, and the container is still running on the source host. Following are the steps of the pre-copy migration technique.

1. First Phase: Only unmodified pages of memory (from the start of the container up to the current state) will be transferred to the destination. Pages that have been modified have a strong probability of updating in the next phase, need not to send them in the initial phase.

2. Second Phase: It will be decided based on the proposed prediction model which pages can be considered for transfer in the iterative phase. In the basic approach, all updated pages are set to be sent in the same iteration. ,

3. Third Phase: If there is any page fault or any failure generated at the destination, we can get it from the source host according to the proposed model. It also supports decreasing data transmission while migrating back to the same host.

2.5 Summary

Prediction of memory changes is the core component of pre-copy container migration. It should be chosen wisely according to pre-copy migration's pre-dump, iterative, and final dump phases. A prediction scheme or set of multiple methods should be applied to the memory pages to be migrated to the destination host to improve the prediction mechanism. Some of the popular schemes available are PSO-based prediction schemes, LSTM-based schemes, and Artificial Neural Network (ANN) based schemes.

It is found that pre-copy needs to be appropriately discussed because of its popularity in migration techniques. The pre-migration ensures the availability of resources between destination and source. To pick a suitable destination host and confirm that, source hosts share information like memory size and running application details with the destination host. First, ensure the resources at the target host and acknowledge the source host to start the migration for the resource reservation. For the pre-dump phase, a

probability-based PSO migration technique is proposed and implemented to reduce the size of the memory dump. Furthermore, in iterative pre-copy, the source host starts the iterative copying of memory pages. It sends all the memory pages updated in the previous iteration in the coming iteration. A predictive checkpoint for the iterative dump is proposed to reduce the amount of data transfer. At last, the "Stop-and-Copy" container will stop operating and move the remaining high dirty index memory pages to the destination. In some cases, like fault tolerance, load balancing, etc., the container migrates back to the same host. A memory reusing approach is proposed to minimize the overall migration size, and it also provides support for page faults.

CHAPTER 3

AN IMPROVED CONTAINER MIGRATION TECHNIQUE USING PSO-BASED PREDICTIVE PRE-DUMP

In this chapter, the proposed pre-copy container live migration using probability-based particle swarm optimization to overcome the memory transmission limitations. The pre-dump is the first phase of sending the container file system to the destination host. Along with that, dirty pages are predicted during a migration process using the meta-heuristic approach. After that, the active set of pages and their update rate has also been identified. In addition, based on the threshold level of maximum update rate, the pages have been shortlisted to be discarded from pre-dump[1].

[1]This chapter is derived from:

Gursharan Singh, Parminder Singh, Babar Shah, Farman Ali and Daehan Kwak. "A Lightweight Migration Technique using PSO based Predictive Pre-dump in Cloud." (Under Review).

3.1 Introduction

In recent years, lightweight virtualization has ignited the interest of eminent researchers around the globe because of its vital scope. The present research has found that container virtualization systems are more effective and more straightforward to handle than traditional virtualization systems. The live migration processes can be substantially reduced by using lightweight containers developed on the virtualization techniques. It shrinks the suspension of service by transmitting minimum memory of the source without stopping the migration process. Therefore, transferring many memory pages leads to long migration time and downtime. Furthermore, it affects the overall system's performance, worsening the over long distance and slowing down of networks.

The container migration is the process of detaching all the processes running in the container's context. Further, they are transferred to a different host and synchronized on the other side of the new operating system [77]. Live migration of containers between different physical hosts is known to be achievable with checkpoint and restore techniques [78] [79]. To migrate a container while retaining the containerized software log and keeping open network connections is achievable through live container migration techniques. Its significant benefits include clean hardware and software separation, low-level device maintenance, fault tolerance, easy data access, and dynamic load balancing, which are checked with different physical hosts [80]. Migration may be used to decrease the power consumption by placing containers on a specific host together and enabling the release of hardware resources that are currently idle. It is found that many resources are passed across the network during migration, such as CPU state, network state, and memory state. The volume of the memory state depends on the form of applications being migrated [38]. The transfer of the whole memory state during migration takes too long to be a realistic solution. This issue becomes especially problematic as containerized programs change large amounts of memory more quickly than a container can transmit across the network.

A significant concern in container live migration is to identify the method for transmitting the container state of memory to a target host [33]. Reducing the downtime and total migration time is the primary goal of improving the live migration algorithm's efficiency. The total migration time is the period between the migration cycle initiation

and completion. Downtime is the time when neither on the source nor on the destination, the container application migrated does not run. Live migration algorithms aim at reducing downtime, during which the application process becomes completely inactive, while maintaining the overall migration period as short as possible [81].

Over the past few years, various container live migration algorithms have been proposed. These algorithms are divided into three groups of migration techniques. The first is the pre-copy migration technique, and this migration starts by transferring the state of the memory to the destination host. The root host stays attentive when copying and proceeds to progress all operating applications. On the source platform, the memory pages may be modified, and the solution utilizes methods to track page changes only when they are transferred to the destination network [30]. The second migration process is post-copy. It first suspends the running application on the source host to restore the applications on the destination host. Another migration technique is hybrid migration. This method uses a combination of both pre and post-copy. It first begins pre-copying the application migration, which proceeds to operate on the source platform. When the destination platform receives all the memory pages, the application is suspended and copied to the processor state. Next, the application is automatically restored at the destination host, and the post-copy algorithm further synchronizes the remaining memory pages [41]. Additionally, it transfers a minimum processor state, and then it starts gathering memory pages from the source over the network.

The research reveals that most container migration implementations use the pre-copy migration technique [82] and this method helps the transition of pages to be streamlined and decreases the overall migration period and downtime by holding the memory pages in volatile memory. The performance of pre-copy migration depends on the intensity of application memory, and the total migration period and downtime rely on the amount of memory consumed for the migration process. The live migration process has drawn further publicity with the increasing interest in container technologies. This method can be used by moving a running process from one machine to another to allow load balancing or fault tolerance. Moreover, live migration is the process of moving an application to its destination host and restoring it to the original state [28]. The live migration function can be applied to the process hierarchy, rendering it a suitable container migration-based technology. Early checkpoint and restore implementation

were introduced in the context of an in-kernel approach. It is better to use a system that will reduce the size of the pre-dump and reuse it in the subsequent phases.

A swarm of n particles communicates to find global optimum solutions by updating their position based on experience or its neighbor's experience. The meta-heuristic algorithm inspired by bird flocking and the intelligence of swarms to solve computationally complex problems is incorporated with migration techniques to boost the performance. It is a robust technique used for a wide variety of search and optimization problems [83]. There are a few initial parameters like size, particle position, particle velocity, and several iterations [84]. In the same way, the control parameters are size, inertial weight, acceleration coefficients, and iterations. We have applied PSO to predict the pages to be updated to minimize the data transfer during container migration.

3.1.1 Major Contributions

The paper's primary contributions include: addressing the issue of dirty pages in container checkpoint files being repeatedly copied and communicated, resulting in more extended application downtime and increased migration costs. A pre-copy migration technique is developed that uses a probability-based PSO approach to detect dirty pages. It decreases the size of the pre-dump, which further reduces data transmission over the network and pre-dump transfer time because most container checkpoint methods include all pages in the pre-dump that may be re-transmitted during the subsequent phase (iterative dump) of the container migration.

The major contributions of this chapter are:

1. A predictive pre-dump approach is designed to minimize the data transfer in Phase 1 of container migration.

2. An algorithm is designed to keep track of memory pages and maintain activity status.

3. Algorithms are framed to calculate the update rate of pages and to finalize the checkpoint.

4. To identify the memory pages to be migrated or discarded, a probability-based PSO algorithm is originated.

5. The experiments have been conducted on a container-Cloudsim 4.0 simulator, and results show that the proposed probability-based PSO has reduced the size of pre-dump.

3.1.2 Chapter Structure

In this chapter, the related research findings are discussed in Section 3.2. According to the review conducted, the problem statement is discussed in Section 3.3. System architecture is elaborated in Section 3.4.1 along with the proposed pre-dump model in Section 3.4.2. The methodology used for the proposed pre-dump is discussed in detail. The Probability-based PSO is elaborated in Section 3.4.3, apart from it, the experimental evaluation and results are discussed in Section 3.5 and 3.5.1 and concluded in Section 3.6.

3.2 Related Work

The live migration of containers refers to transferring applications among different physical devices or clouds without disconnecting the clients. It is vitally observed that the file system, memory, and network configuration of the containers running on top of bare metal hardware are moved from the original host machine to the destination holding the state with minimum possible downtime [36]. There are two main directions of container migration: one is the selection of the container which can be moved, and the second is the procedure of moving a container. The process of moving a container application is executed by using any one of these methods, such as pre-copy migration, post-copy migration, or hybrid migration approach [27]. Container migration's main task is to migrate memory pages. In pre-copy, the first copy of the memory page is performed, and in phase two, the copying of the pages is performed iteratively. The first stage in postponed copying is to shut off the process and transfer the container's state-specific information to its destination host. Then the container process is rebooted on the destination host, and synchronized memory pages are received. Both the migration techniques take the appropriate time to configure the required resources. Delayed copy output is primarily based on the pace of the workload and the network connection. Based on the container's limited life span, they opt for a pre-copy approach to simplify

the process of migration.

Authors in [39] illustrated the Pre-copy phases and how dirty pages are found when we migrate to live. Experimental results showed that all received packets could be efficiently migrated to ensure the integrity of application data as SRVM do not maximize here. The equivalent framework helps to minimize container migration downtime. Another approach is proposed by [43] to work on resource containers and safety containers for standard, time-shared systems. They present the design and implementation of Linux-VServer as a representative instance of container-based systems. It also compares Linux-architecture VServer to current Xen generations and illustrates how Linux-VServer supports comparably and superior device efficiency [85].

The method based on compression and deduplication of data is elaborated by Luo et al. [44]. The authors used the RLE approach for migration to eliminate duplicate memory data using the runtime storage image identity. For page similarity computations, hash-based fingerprints have been used, and for implementation, LRU Hash tables FNHash and FPHash were utilized. The study made it clear that migration efficiency improves in space with overhead CPU resources.

The basic technique of pre-copy transfer memory pages repeatedly at high replacement rates during the incremental copying process is provided by Ansar et al. [45]. The paper presented thus an OPCA, which includes a Gray-Markov model. They found out the modification rate of every memory page based on the proposed prediction model and shortlisted the pages with a high speed of modification. Results showed that the total migration time and downtime are decreased with the decrease in the number of iterations. Moreover, the resource utilization is also increased by OPCA. In turn, the use of resources is enhanced, and the user experience is improved. Experiments show that OPCA minimizes iterations and downtimes, raises resource usage levels and improves user experience. Additionally, it optimizes the conventional pre-copy algorithm to minimize downtime and displays Markow's Gray model and the current space definition. The work area is divided by calculating the re-shaped probability of the memory tab in which Highly changed memory pages are stored in the Hot Workspace (HW) and copied throughout the final process. The WW shall therefore be used to reduce errors.

The live container just-in-time migration service built-in line with OCI principles was introduced by Nadgowda et al. [10]. Voyager's new file system and supplier-

agnostic migration software provide CRIU memory-based migration and minimize migration downtime. Design and implementation employ a data federation in several sources and host goals to allow this application to restart on time at a host with remote readers and local-write access. Containers are re-operated on the server with combined data between the source and destination host, but the status of the disk is increasingly transferred [86].

Each method's implementation scenarios by contrasting the pre-copy, delayed copy, and Hybrid copy are tested in [55] and pre-copying is the primary mode of migration. The proposed pre-copy is further divided into sub-phases to minimize the number of memory transfers and to minimize the container migration by forecasting the operations trend [82]. The problem of migration from the traditional energy-saving viewpoint is addressed by Patel et al. [46], which optimizes the actual migration technique to reduce the consumption of energy [87]. However, the levels of energy use are not uniform, so Comparison and assessment are not straightforward. The idea of reuse distance is adopted, and the modified memory pages are tracked back within the copy process [54], but the algorithm's difficulty is hard to enforce. Several popular memory compression algorithms are listed and analyzed from the memory perspective, including RLE and Huffman Coding, and it proposes a new technique to minimize migration time.
From the memory compression perspective, there are several drawbacks to consider, and suggest an efficient scheduling model [12]. The container is scheduled on the corresponding server to accomplish load balancing. Likewise, dynamic migration core implementation scenarios are elaborated [35]. To recover large data structures from a catastrophe, a wide-scale dynamic migration mechanism relies on network conditions, migration policies, controller categories, and memory allocation [88]. They forecasted the often-changed memory pages using the prediction model of the time series and then moved them in the last copying round to the destination container [47]. The TPO method reduces the number of pages transferred in each iteration and optimizes the migration process.

It is found that pre-copy needs to be adequately discussed because of its popularity in migration techniques. The pre-migration ensures the availability of resources between destination and source. To pick a suitable destination host and confirm that, source hosts share information like memory size and running application details with

the destination host. First, ensure the resources at the target host and acknowledge the source host to start the migration for the resource reservation. Furthermore, in iterative pre-copy, the source host starts the iterative copying of memory pages. It sends all the memory pages in the first iteration, and the pages updated in the previous iteration will be transferred in the upcoming iteration. At last, the "Stop-and-Copy" container will stop operating and move the remaining high dirty index memory pages to the destination. To improve the throughput of this migration process, the prediction model should be used to forecast the probability of memory pages being modified.

3.3 Problem Formulation

The containers provide a reliable environment for virtual computing. The selection of container migration techniques is an essential factor affecting the performance of container migration. All these techniques were discussed [82]. The Comparison shows that pre-copy and hybrid copy perform better than "post-copy" and "stop and copy" migration techniques. Focusing on the pre-copy live migration technique, the task is divided into three stages: pre-dump, iterative, and final. During these stages, the container is running to perform live migration. All the memory pages related to that particular container are sent to the destination host in the pre-dump phase. The rest of the pages are sent iteratively in the second phase. The final set of memory pages is sent in the third stage.

Almost all the research and findings related to container migration are either in the iterative dump phase or final dump phase. The performance of containers has been improved with these findings, to enhance the pre-dump phase. So, the primary objective is to minimize the data transfer across different hosts during migration in the pre-dump phase. Therefore, we conducted a detailed review of container migration techniques to determine the scope of improvement which leads to the performance enhancement.

A predictive algorithm is designed to determine if the memory pages are updated before sending them to the destination host. The study originated the idea that if the content of memory pages changes, then the page becomes a dirty page. It identifies recently updated pages according to the concept of dirty bits. In the proposed algorithm, only shortlisted pages will be sent to the destination in stage one of the migration process. In this way, data transfer will be reduced, which will impact the cost of net-

work overhead. The concept of pages with dirty bits provides us with the history and pattern of page modification in recent times, but it is not sufficient to identify the set of pages to be migrated to the destination host. Therefore, considering the possibility of change soon, a predictive method is required. The PSO is selected to predict the chances of modification of a memory page based on the history of a page. PSO will shortlist the pages by using the page updation rate. Together, these methods can better identify memory pages that need not be sent in the pre-dump phase.

The process of pre-copy container migration is divided into three phases known as phase 1 (pre-dump), phase 2 (iterative-dump), and phase 3 (final-dump), as shown in Figure 3.1.

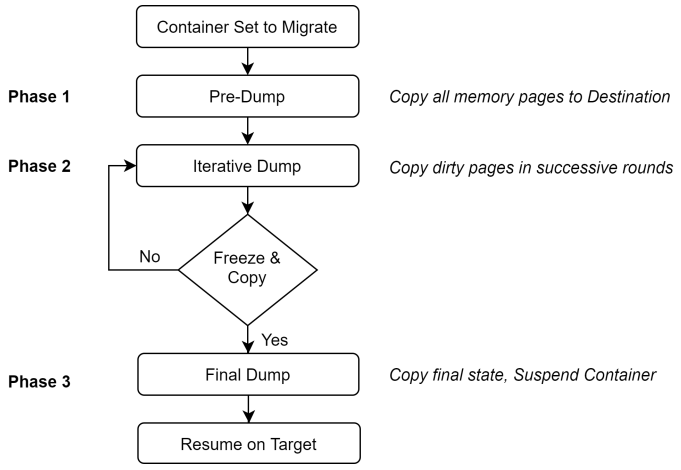

Figure 3.1: The set of various phases of container migration process along with the flow of execution.

These phases of the standard migration approach take a long time to perform the migration of containers. This is due to the multiple transmissions of the same memory in different phases. Although containers have already increased the performance of migrations compared to virtual machines, there is still scope for further enhancement. Our objective is to minimize the amount of data transfer over the network in the migration process by reducing the dump size during the pre-dump phase. The proposed prediction scheme for the pre-dump stage to reduce the size of memory pages is applicable to phase 1 in Figure 3.1.

3.4 Modeling and Methodology

The present study will first present the system model in an explanation of all its components in section 3.4.1 then the proposed methodology will be discussed in section 3.4.2.

3.4.1 System Model

The system architecture on the container is mentioned in Figure 3.2. It is observed that when a container is created for a specific purpose, some sets of files get associated with it. Working set, code, operating system, configuration files, and kernel dependencies are required to run a container as container executions start once all the resources are granted. Its engine monitors all the activities of a container, and then the container engine handles all the activities of containers. For ease, multiple containers can be created and monitored simultaneously. However, lightweight containers are designed for a single task or process. But a container engine can create multiple containers to handle various processes with complete isolation. If there is any need to migrate a container to another host due to any reason, then a container image will be created with all its dependencies and current status. Furthermore, you can migrate the container to its destination host. In the case of container virtualization, no additional hypervisor is required. It can directly run on the Kernel of the operating system.

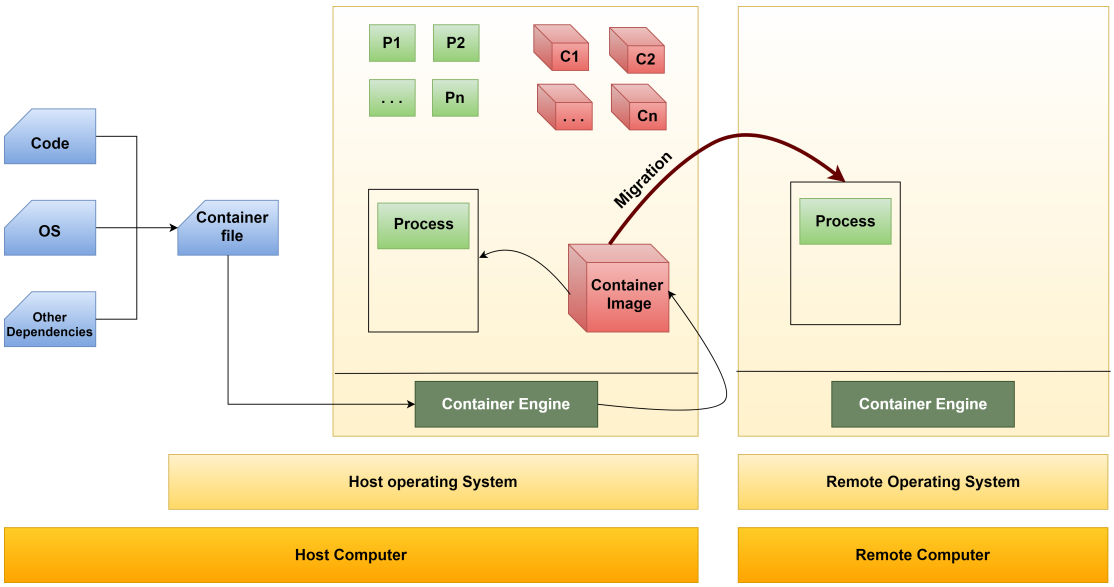

Figure 3.2: The basic system architecture of container migration

3.4.2 Proposed Pre-dump Technique

In the proposed pre-dump technique, it is not required to dump all the pages at the beginning as pages can be stored according to the steps mentioned in section 3.3.

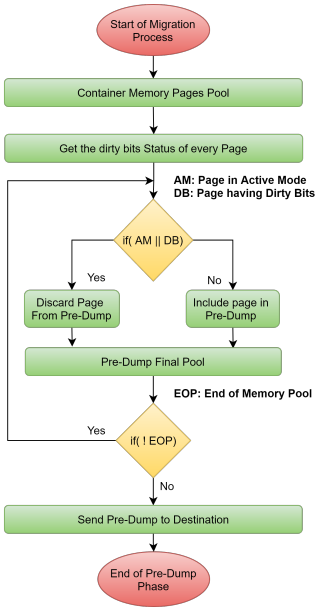

Figure 3.3: The process of pre-dump steps to be followed in proposed pre-copy container migration to shortlist the pages for transmission.

Following are the steps of the proposed pre-dump technique for container migration:

1. When it is decided to migrate a container, gather complete memory information and make a pool of pages.
2. Track the activity and Dirty Bits (DB) status of every page based on the last few minutes of execution.
3. If the page is not in Active Mode () or has dirty bits, then that page will be in the sending pool. Include the pages with an updation rate less than the threshold value. (50%)
4. If a page is in write/update mode, it need not be included in pre-dump. (it will be changed again).
5. Repeat step3 until the End of Memory Pool (EOP) is not reached.
6. Once a decision is made for the Memory Pool (MP), then send the pre-dump collection to the destination.

7. Read-only pages can be included now. (it will save time in the last dump).

3.4.3 Probability-based Particle Swarm Optimization for Container Migration

To execute the study, researchers minimized the pre-dump size by predicting the active set of pages. The proposed prediction Algorithm 1 identifies the recently updated pages, and these pages can be discarded from the pre-dump pool.

Algorithm 1 Pseudo code to identify active pages in pre-dump of container migration process

Result: Pool of Active Pages

$P_i = 0, R_i = 1, R_{max} = 5$

while $P_i \leq 5$ **do**
  $R_i = 1$
  while $R_i \leq R_{max}$ **do**
    if $D_{Flag}[P_i][R_i] \neq 0$ **then**
      $D_{Flag}[P_i][R_i] = 1$
      $Active[P_i] = 0$
    end
    $D_{Flag}[P_i][R_i] = 0$
    $Active[P_i] -= 1 \quad R_i += 1$
  end
  $P_i += 1$
end
return (Active$[P_i]$)

Why PSO is selected for finding the probability of page update in container migration. There are several reasons behind the selection. As compared to the other genetic algorithm, the PSO outperforms [89]. In Comparison to GA, PSO maintains a constant situation when locating the best solution. There is no need to incorporate population destruction procedures (like with genetic algorithms, which destroy populations that have saturated after a specific number of iterations to increase their accuracy) because PSO can obtain optimum solutions under a variety of scenarios. It produces the optimal

solution every time with the complexity of $O(n)$. With more variables and constraints, the number of iterations required will increase, but PSO can handle the same with few iterations [90]. No additional technique is needed to produce an optimal solution. Furthermore, the process of PSO is suitable for using the updation rate of pages as input to the algorithm to find the final set of pages.

The parameters used in this algorithm are as: P_i = 0 and $R_i = 1$ is used to select the page from memory pool, R_{max} is the maximum number of iterations and D_{Flag} is used for dirty bits. According to the selected memory pool, the $0 \leq P_i \leq 5$ $and 0 \leq R_i \leq R_{max}$ and $R_{max} = 5$.

Algorithm 2 Pseudo code to get the update rate of pages

Result: Updation Rate of page Pi

$P_{db} = 0, R_{max} = 10$

while $R_i \leq R_{max}$ **do**

  if $D_{Flag}[P_i][R_i]1$ **then**

    $P_{db} += 1$

  end

  $R_i += 1$

end

return (P_{db}/R_{max})

The dirty bits rate of memory pages is also taken into consideration. It will be decided based on the probability of a page becoming modified, whether a page is included in a dump or not, and it will be decided at the initial state of the migration process called a pre-dump phase. Algorithm 2 is used to get the rate of update of every page belonging to a container with a unique id. P_{db} is the number of dirty bits, R_{max} is the maximum number of rounds, P_i is the page number and R_i is the i^{th} iteration.

One way to finalize the checkpoint with a pool of memory pages to migrate to the destination host is done with the help of Algorithm 3. The $S_{Pool}[]$ is the pool of shortlisted pages for transmission, and the pool of pages that are to be discarded are in $D_{Pool}[]$. To enhance the performance, the probability-based PSO is chosen.

Algorithm 3 Pre dump checkpoint

Result: Final set of memory pages to be migrated

Access the memory pool of container Get the read write status of every memory page

while $Mpool$ **do**

  if $UR(Pi) \leq threshold || Active(Pi)$ **then**

    $S_{Pool}.append(Pi)$ ▷ Page will be added to Send pool

  else

    $D_{Pool}.append(Pi)$ ▷ Page will be added to Discard pool

  end

end

Transfer the Spool[] to the destination host.

3.4.4 Probability-based Particle Swarm Optimization

A particular swarm consists of a certain number of particles within the PSO algorithm. At every iteration, all particles move with a certain speed to locate the global optimum in the N-dimensional problem field and i (1...N), where D_i^n is the vector at iteration n.

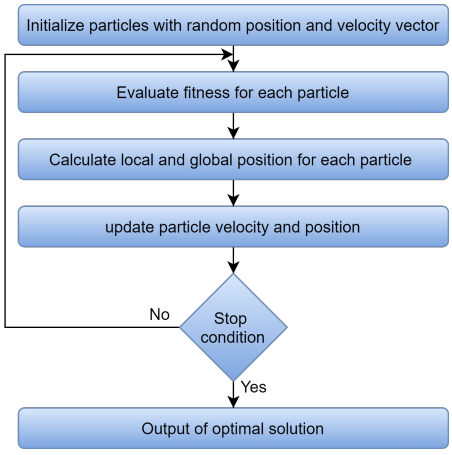

Figure 3.4: The step wise process of particle swarm optimization to achieve a optimal solution

$$v_i^{n+1} = wv_i^n + c_1 r_1 (B_i^n - D_i^n) + c_2 r_2 (B_g^n - D_i^n) \tag{3.1}$$

Table 3.1: Symbols and letters

Symbol	**Description**
B_i^n	Local best position of i^{th} particle at iteration n.
B_g^n	Global best position of all the particles at iteration n.
r1 and r2	are random numbers between 0 and 1.
w	The inertia weight.
c1 and c2	Constant cognitive confidence coefficient numbers.
D_i^n	indicates the particle orientation at iteration n.
V_i^n	indicates the velocity of the vector at iteration n.
$R1$ and $R2$	Matrices for the representation of random numbers and updation rate ranging from 0 to 1.
$\ominus$	Subtraction from the matrix.
$\oplus$	refers to the addition of matrices.
$\times$	Multiplication of the elements.
$\otimes$	Modified matrix multiplication, where the values over 1 are set to 1 .

$$D_i^{n+1} = D_i^n + v_i^{n+1} \tag{3.2}$$

Particle's Representation

Usually, the feasible solution is immediately shown as a particle D_i that can be modified according to Equation (3.1) while solving continuous optimization problems in Equation (3.2). However, it is mainly pointless to upgrade a viable solution to discrete problems until we have a possible solution. In other words, the main point is the relation to the solution of the problem between particles. In this process, the memory pages are represented as particles. Sequencing problems can be considered most discrete, and the planning problem for a five-activity $1,2,3,4,5$ can be called a five-component sequence problem.

A permutation consists of such fundamental elements such as $(1,2,3,4,5)$ permutation, composed of essential elements $(1,2),(2,3),(3,4)$ and $(4,5)$ and each basic element is called sub-permutation. Any relatively strong sub-permutations must be a superior permutation. The permutation is designed with two-dimensional arrays called the adjacency matrix. In this matrix, the elements fit the sub-mutations set at 1. The other elements are set at 0 to classify these successful sub-permutations according to Equation (3.1), Two examples of adjacent matrices that represent permutations are present: The representation of $(1,2,3,4,5)$ is :

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.3}$$

and (2, 3, 1, 4, 5) can be described as

$$D_i^n = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.4}$$

In above adjacent matrix, where the equation a $i, j = 1$,
The number $'i'$ is above the number $'j'$ and adjacent to the element's permutation. The location of the particle in our algorithm is a $(0,1)$ in the adjacent matrix, that is a permutation.

Operator Notation and Definitions

A matrix is used as a representation of both location and velocity, therefore operators in Equation(3.1) and Equation (3.2) are redefined.

$$v_i^{n+1} = wV_i^n \oplus c_1 \otimes R_1 \times (B_i^n \ominus D_i^n) \oplus c_2 \otimes R_2 \times (B_g^n \ominus D_i^n) \tag{3.5}$$

$$BD_i^n = f_p(D_i^n) \tag{3.6}$$

$$BD_i^{n+1} = BD_i^n \oplus V_i^{n+1} \tag{3.7}$$

$$D_i^{n+1} = f_s(BD_i^{n+1}) \tag{3.8}$$

Where the D_i^n indicates the particle orientation at iteration n and the V_i^n indicates the velocity of the vector at iteration n. B_i^n is the best location at the i^{th} portion at iteration n, and B_g^n is the best global position at n^{th} iteration for all of the particles. D_i^n, B_i^n and B_{gd}^n are all matrices that fit their permutations, which are $(0,1)$. $R1$ and $R2$ matrices are the same sizes as D_i for the representation of random numbers and updation rate ranging from 0 to 1. w is considered the weight of inertia which has a value from 0 to 1. c_1 and c_2 are constant numbers and cognitive trust coefficients. The $'\ominus'$ symbol implies subtraction from the matrix to the negative elements in the resulting subtraction. The symbol $'\oplus'$ refers to the addition of matrices, with the addition product of all elements larger than 1 set to 1. The symbol $'\times'$ refers to a multiplication of the elements of each matrix. There are two same-sized matrices, A and B. If A and B are identically sized, then $A \times B$ is also identical in size with elements as respective A and B. The symbol $'\otimes'$ is used to denote a modified multiplication. c is a real integer, and then c $\otimes$ A implies that all elements of matrix A are multiplied by c, and the values over 1 are set to 1.

The explanation for separating one permutation set from the other is its sub-permutation. Consequently, a good permutation set must contain good and non-existent permutations in other permutations. Probability coefficient selection approach has been developed based on the above concepts. The basic principle is to produce new permutations based on the specification of probabilities and to pick those successful sub-permutations. Using the PSO algorithm to locate these relatively strong sub-permutations and make them more likely to be chosen for new permutations. According to Equation (3.8), V_i^{n+1} incorporates three components of the vector of transfer probability: $wV_i^n, c_1 \otimes R_1 \times (B_i^n \ominus D_i^n)$, and $c_2 \otimes R_2 \times (B_g^n \ominus D_i^n).wV_i^n$ is used to assess to what degree the old svelocity V_i^n

specifies the current velocity V_i^{n+1}. Typically w=0.9 and c_1 and c_2 are both learning variables to monitor the impact on new velocity of local knowledge and global experience. They are normally set to 1. $B_i^n \ominus D_i^n$is used in the best positionB_i^n. $B_g^n \ominus D_i^n$ is used in the global best place to locate the special sub mutations B_g^n but not in the single D^n.An example is provided in Equations (3.2), (3.5), (3.6), (3.7), and (3.8). Let D_i^n be the matrix of representing the permutation of $(1,2,3,4,5)$.

$$D_i^n = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.9}$$

and B_i^n is a matrix representing the permutation of $(2,3,1,4,5)$.

$$B_i^n = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.10}$$

Then results show that the sub-permutations $(3,1)$ and $(1,4)$ are unique for B_i^n compared with D_i^n.

$$B_i^n \ominus D_i^n = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.11}$$

These are the random numbers represented in the range between 0 to 1 as matrix R_1.

$$\begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.5 & 0.4 \\ 0.8 & 0.6 & 0.4 & 0.6 & 0.6 \\ 0.6 & 0.4 & 0.6 & 0.3 & 0.5 \\ 0.9 & 0.5 & 0.6 & 0.8 & 0.9 \\ 0.6 & 0.3 & 0.9 & 0.4 & 0.3 \end{bmatrix} \tag{3.12}$$

In comparison to R_1, the binary matrix is transformed by $c_1 \otimes R_1 \times (B_i^n \ominus D_i^n)$ and

$$V_i^n = \begin{bmatrix} 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.13}$$

The elements are highlighted in the matrix of sub-permutations $(3,1)$ and $(1,4)$, increasing the probability of being selected in the new permutation. Probability coefficients produce new particles. The main PSO concept requires considering the current position and velocity to update the selection probability coefficients. Therefore,D_i^n transform to a matrix of probability coefficients. The BD^i is defined in Equation (3.6) as an D_i^n matrix extracted from the D_i^n by the $f(p)$ function. According to the following rules, $f(p)$ can be freely constructed by ensuring that the BD_i^n elements corresponding to the D_i^n elements have a strong probability and meaning the elements in BD_i^n that cannot be used to make a selection is 0.

Process 1: Pages shortlisted based on threshold

It corresponds to the same set of memory pages; the updation rate of pages is recorded using Algorithm 2 and represented in the form of the same size matrix as R_2.

$$\begin{bmatrix} 0.8148 & 0.4821 & 0.0561 & 0.5682 & 0.0708 \\ 0.2857 & 0.5572 & 0.3868 & 0.5357 & 0.9968 \\ 0.9176 & 0.5387 & 0.0902 & 0.5562 & 0.9342 \\ 0.7294 & 0.7655 & 0.0884 & 0.4895 & 0.2468 \\ 0.9704 & 0.2687 & 0.8507 & 0.8149 & 0.4830 \end{bmatrix} \tag{3.14}$$

If the page updation rate is more than the threshold value, then the page represented as the particle is replaced with value 0 in the BD_i^n and 1 in the corresponding matrix D_i^{n+1} based on the applied threshold value (0.8). It is tested on three threshold levels (0.6, 0.7, and 0.8).

$$BD_i^n = \begin{bmatrix} 0 & 0.4821 & 0.0561 & 0.5682 & 0.0708 \\ 0.2857 & 0.5572 & 0.3868 & 0.5357 & 0 \\ 0 & 0.5387 & 0.0902 & 0.5562 & 0 \\ 0.7294 & 0.7655 & 0.0884 & 0.4895 & 0.2468 \\ 0 & 0.2687 & 0 & 0 & 0.4830 \end{bmatrix} \tag{3.15}$$

$$D_i^{n+1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.16}$$

The D_i^{n+1} is used as the first set of shortlisted pages to be discarded from the pre-dump.

Process 2: Shortlist pages based on PSO

In addition to process 1, now the PSO-based selection method is discussed. Now D_i^n starts with a permutation of(1, 2, 3, 4, 5) in the corresponding matrix.

$$D_i^n = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.17}$$

And the BD_i^n is used with the same updation rate as in process 1. The outcome of process 1 as an input to this process 2.

$$\begin{bmatrix} 0.8148 & 0.4821 & 0.0561 & 0.5682 & 0.0708 \\ 0.2857 & 0.5572 & 0.3868 & 0.5357 & 0.9968 \\ 0.9176 & 0.5387 & 0.0902 & 0.5562 & 0.9342 \\ 0.7294 & 0.7655 & 0.0884 & 0.4895 & 0.2468 \\ 0.9704 & 0.2687 & 0.8507 & 0.8149 & 0.4830 \end{bmatrix} \tag{3.18}$$

Then, except for the items corresponding to an element in the D_i^n, and divide each element by 1000. The diagonal elements in D_i^n are set to 0 because they are not equal, and the corresponding elements in BD_i^n cannot be used.

While considering the current position and velocity, the probability coefficient matrix is modified as per the following.

$$V_i^{n+1} = \begin{bmatrix} 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.19}$$

then $BD_i^{n+1} = BD_i^n \oplus V_i^{n+1}$ is modified and represented as

$$BD_i^{n+1} = \begin{bmatrix} 0.0008 & 0.4821 & 0.0001 & 0.5006 & 0.0001 \\ 0.0003 & 0.0005 & 0.3868 & 0.0005 & 0.0008 \\ 0.6009 & 0.0005 & 0.0001 & 0.5562 & 0.0008 \\ 0.0007 & 0.0008 & 0.0001 & 0.0005 & 0.2468 \\ 0.0008 & 0.0003 & 0.0008 & 0.0008 & 0.0005 \end{bmatrix} \tag{3.20}$$

Here in this equation, (BD^{n+1}) this operation for the probability collection was defined, which selects and sets the right elements in one D_i^{n+1} according to the given probability coefficients in BD_i^{n+1}. If an element in BD_i^{n+1} is of a larger coefficient probability value, the selection procedure shall follow the rule that the element with the same index in D_i^{n+1} is more likely to be chosen and set to 1. so, for the new permutation, the adjacency matrix is inserted and decoded.

3.4.5 Probability Coefficients-based Selection Method

The selection of probability coefficients can also split optimal conditions to find better solutions. The selection process is outlined below:

- Step 1: Set D_i^{n+1} to 0.
- Step 2: choose a row based on the maximum number of elements of probability BD^{n+1}.
- Step 3: The relevant element in $BD^n + 1_i$ is used, and from a row, a single element is chosen randomly. selection based on elements with higher coefficients of chance is more likely to be picked.
- Step 4: The selection is set to 1 and BD^{n+1} id to 0 with the corresponding so-called contrasting elements. Three kinds of contrary elements exist:
    1. Elements from same row
    2. elements from same column
    3. elements without complete permutation
- Step 5: jump to step 2 if the exit condition is not satisfied.
- step 6: Union of Process 1 and Process 2 are represented together.

Let's start with Process 2:

$$BD_i^{n+1} = \begin{bmatrix} 0 & 0.4821 & 0.0001 & 0.5006 & 0.0001 \\ 0.0003 & 0 & 0.3868 & 0.0005 & 0.0008 \\ 0.6009 & 0.0005 & 0 & 0.5562 & 0.0008 \\ 0.0007 & 0.0008 & 0.0001 & 0 & 0.2468 \\ 0.0008 & 0.0003 & 0.0008 & 0.0008 & 0 \end{bmatrix} \tag{3.21}$$

$$D_i^{n+1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.22}$$

The components in a row of $(0.6009, 0.0005, 0, 0.5562, 0.0008)$ are used as coefficients according to Process 2 as it comprises the most significant discrete dynamics within the existence of the matrix of 5 values. Pick a column randomly depending on the coefficients for each element except for the third element whose coefficients are equal to zero, so it is decided by applying the selection policy. However, specific components with higher probability coefficients are more likely to be picked and the first element has been chosen. In this case, that particle is already set to 1, so there is no change in the corresponding matrix. It is demonstrated through the example given below:

$$D_i^{n+1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.23}$$

and

$$BD_i^{n+1} = \begin{bmatrix} 0 & 0.4821 & 0.0001 & 0.5006 & 0.0001 \\ 0 & 0 & 0.3868 & 0.0005 & 0.0010 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0008 & 0.0001 & 0 & 0.2468 \\ 0 & 0.0003 & 0.0009 & 0.0008 & 0 \end{bmatrix} \tag{3.24}$$

The element BD_i^{n+1} that is $(1,3)$ is set to 0, according to section 3.4.5, step4 with condition: iii). then, $(0, 0.4821, 0.0001, 0.5006, 0.0001)$ fourth element is selected from these coefficients. Then,

$$D_i^{n+1} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.25}$$

and

$$BD_i^{n+1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3868 & 0 & 0.0010 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0008 & 0.0001 & 0 & 0.2468 \\ 0 & 0.0003 & 0.0009 & 0 & 0 \end{bmatrix} \tag{3.26}$$

BD_i^{n+1}, $(4,3)$ = 0, in that case, the permutation $(3,1,4,3)$ would be created if the matching element in D_i^{n+1} is chosen to 1.

In the next step, the second row is picked and used as a weight to select a particle $((0,0,0.3868,0,0.00010))$ from the second row, and the third column is selected. Then,

$$D_i^{n+1} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.27}$$

and

$$BD_i^{n+1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0008 & 0 & 0 & 0.2468 \\ 0 & 0.0003 & 0 & 0 & 0 \end{bmatrix} \tag{3.28}$$

Next,$(0, 0.0008, 0, 0, 0.2468)$ and fourth row and fifth column is selected. Then,

$$D_i^{n+1} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.29}$$

and

$$BD_i^{n+1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0003 & 0 & 0 & 0 \end{bmatrix} \tag{3.30}$$

and the whole 0-1 adjacency matrix is generated, and the new permutation is (2, 3, 1, 4, 5) based on the adjacency matrix of (0-1).

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{3.31}$$

The probability-based PSO is used to identify the active set of pages to discard them from the pre-dump because such pages have a strong possibility of change soon. In matrix 18, this is an updation rate of pages. Now, the basis of threshold level applied to $(0.6, 0.7 \text{ and } 0.8)$ the pages will be discarded from pre-dump. All the pages represented with 1 will be discarded in the final matrix. Whatever the sub-permutation shown by probability-based PSO and the pages shortlisted by updation rate, the union of both will be used as the final result.

3.5 Experimental Evaluation

The proposed system is implemented and tested with container-cloudsim 4.0 and LSTM libraries in Eclipse. The implementation primarily focuses on the identification of memory pages related to the container and their read/write status. Pages are shortlisted to be migrated with the help of a probability-based PSO prediction scheme.

The performance depends on the total number of pages transferred. In the worst-case scenario, all the memory pages related to the container will be sent. The pre-dump time is the time taken to migrate the shortlisted pages in phase 1 depending on the size of the container, type of container application, and the kind of workload handled by the container. In such a case, the prediction method can be an extra overhead on CPU usage, but this overhead will not affect the cost over the network, which is our prime objective. Furthermore, this additional overhead will not affect the dump time because this is done in the preliminary phase of container migration, and the overhead impacts the processing unit only. Moreover, dirty page rate and network bandwidth consumption are some other parameters.

Before transferring the memory dump to the destination host during the migration process. To check the dirty bits or read/write operation status on the running container is

possible within the last few minutes of execution of the container file system. It provides the details like the number of pages currently active and the modification rate of pages.

3.5.1 Results and Discussions

The pre-dump checkpoint, as mentioned in Algorithm 3 for pre-dump migration, is implemented with different scenarios based on the threshold of updating rate. Then, three threshold levels are implemented (60%, 70%, and 80%), and each threshold level is tested on three different sets of containers (5 containers, 10 containers, and 15 containers). The number of pages in send pool varies with the threshold value. Considering a page in send pool only if it is modified upto the threshold level. The final set of pages are selected through Algorithm 3 and the probability-based PSO.

The purpose of constructing such a scenario with varying threshold values is to determine the average performance of approaches with variable update rates. This help to identify the actual performance with different test cases. If a page has been modified to the threshold level, and consider it inactive mode. The restriction is set to 60% at the threshold level of 0.6. Additionally, at threshold level 0.7, the limit for identifying the current page will be set to 70%, and similarly at threshold level 0.8.

In addition, each set of containers is tested with 5 test cases. The results of these migrations have four parameters (Number of Containers, Test Cases, standard Pre-dump Size, and Pre-Dump Size with probability-based PSO). In Figure 3.5 three sets of containers (5, 10 and 15 containers) are first implemented with threshold 0.6 (up to 60% update). Test cases 1 to 5 are tested with 5 containers, 6-10 with 10 containers, and 11-15 with 15 containers. In the first set of 5 containers, the actual pre-dump size is the same as 4367B in all the test cases because all the pages will be migrated to the destination host in the standard approach. With the proposed technique, the migrated data reduced by 34.74%. Furthermore, in 10 containers, the actual pre-dump size is 8176B, and with the proposed technique, the migrated data reduced by 33.89%. In 15 containers, the actual pre-dump size is 12387B, and with the proposed technique, the migrated data reduced by 35.27%. The proposed technique improves the performance with a huge difference and minimizes the requirement of resources over the network.

Table 3.2: The size of pre-dump using probability-based PSO with threshold level 60%

Containers	Test Cases	Pre-dump Size	Pre-dump Size with Proposed Technique
	1	4367	2848
	2	4367	2876
5	3	4367	2950
	4	4367	2742
	5	4367	2833
	1	8176	5340
	2	8176	5312
10	3	8176	5462
	4	8176	5575
	5	8176	5335
	1	12387	7818
	2	12387	8157
15	3	12387	8226
	4	12387	7998
	5	12387	7890

The average of all the test cases with a threshold level of 0.6 is mentioned in Table 3.3. The difference in data migrated in both the approaches is evident in Figure 3.6. Therefore the average of 5 containers is 2849.8B, 10 containers are 5404.8B, and 15 containers is 8017.8B.

Further in Figure 3.7 again, the same three sets of containers are implemented with a threshold of 0.7 (up to 70% update). In the first set of 5 containers, the probability-based PSO pre-dump size is (3251, 3175, 3133, 3215, 3163) with an average of 3187.4B, but

Table 3.3: The average size of different set of containers in pre-dump with threshold level 60%

Threshold 0.6	5 Containers	10 Containers	15 Containers
Data Migrated with Proposed Technique	2849.8	5404.8	8017.8
Standard Migration Technique	4367	8176	12387

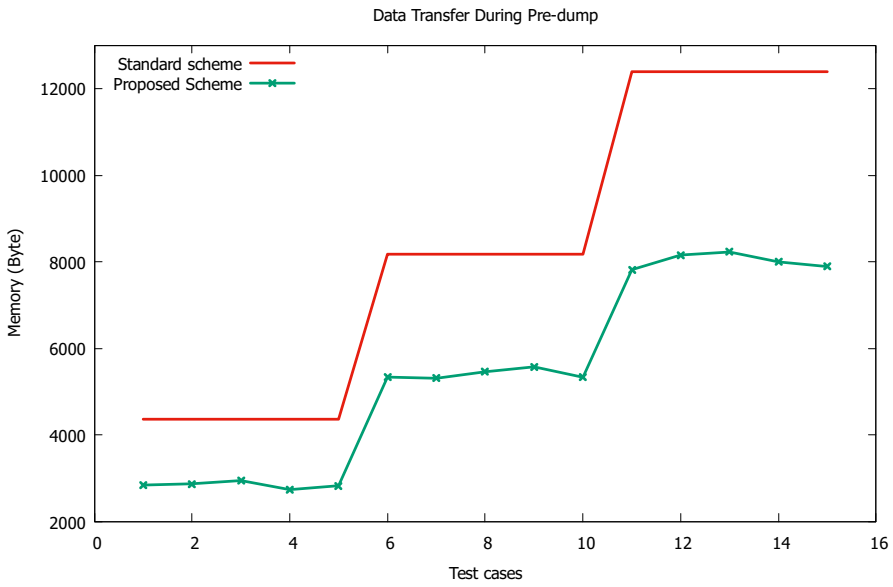

Figure 3.5: The size of pre-dump using probability-based PSO with threshold level 60%

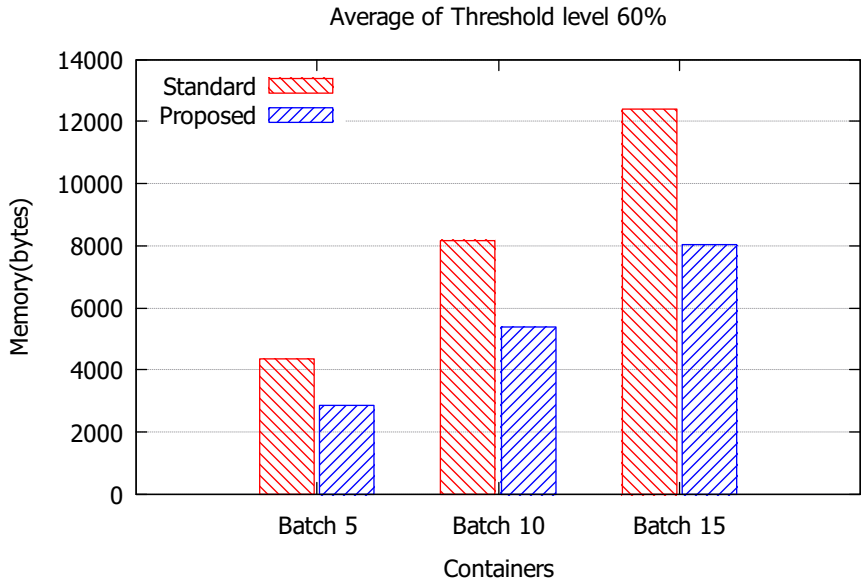

Figure 3.6: The average size of different set of containers in pre-dump with threshold level 60%

Table 3.4: The size of pre-dump using probability-based PSO with threshold level 70%

Containers	Test Cases	Pre-dump Size	Pre-dump Size with Proposed Technique
	1	4367	3251
	2	4367	3175
5	3	4367	3133
	4	4367	3215
	5	4367	3163
	1	8176	5991
	2	8176	5862
10	3	8176	6170
	4	8176	5936
	5	8176	6243
	1	12387	9079
	2	12387	8940
15	3	12387	8667
	4	12387	9118
	5	12387	8814

Table 3.5: The average size of different set of containers in pre-dump with threshold level 70%

Threshold 0.7	5 Containers	10 Containers	15 Containers
Data Migrated with Proposed Technique	3187.4	6040.4	8923.6
Standard Migration Technique	4367	8176	12387

in the standard approach, this size is 4367B. Now, with the proposed technique, the migrated data reduced by 27.01%. Therefore, in a set of 10 containers, the proposed pre-dump size is 6040.4B, and the actual was 8176B which is reduced by 26.12%. In the group of 15 containers, the actual pre-dump size is 12387B, and the proposed technique is 8923.6, which 27.9% reduces.

The average of all the test cases with a threshold level of 0.7 is mentioned in Table 3.5. The difference in data migrated in both the approaches is apparent in Figure 3.8. Therefore, the average of 5 containers is 2187.4B, 10 containers are 3978.4B, and 15 containers are 6534.2B.

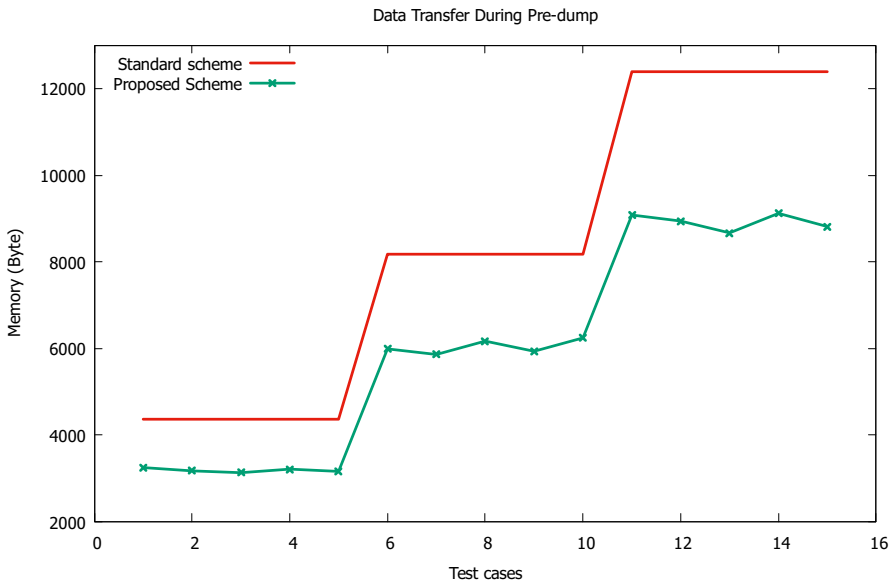

Figure 3.7: The size of pre-dump using probability-based PSO with threshold level 70%

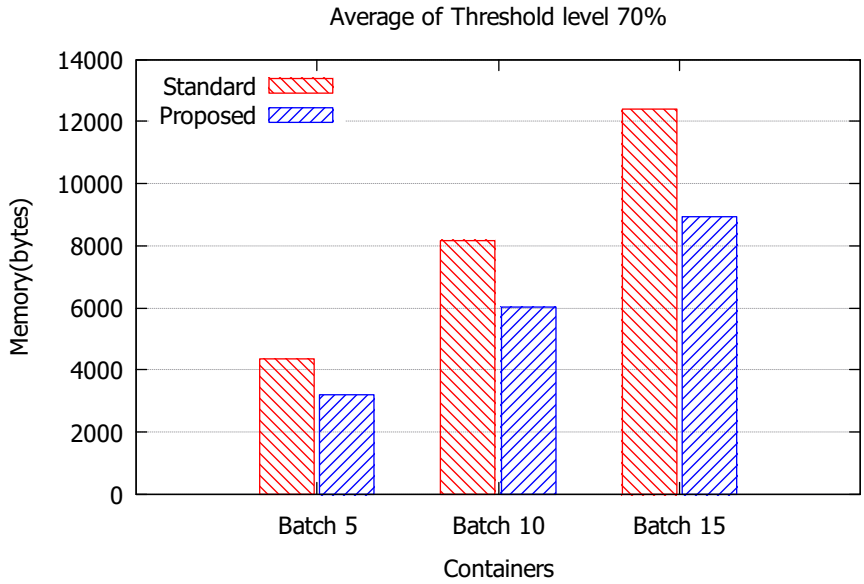

Figure 3.8: The average size of different set of containers in pre-dump with threshold level 70%

Table 3.6: The size of pre-dump using probability-based PSO with threshold level 80%

Containers	Test Cases	Pre-dump Size	Pre-dump Size with Proposed Technique
	1	4367	3654
	2	4367	3457
5	3	4367	3316
	4	4367	3723
	5	4367	3765
	1	8176	6573
	2	8176	6842
10	3	8176	6588
	4	8176	7146
	5	8176	6816
	1	12387	10063
	2	12387	9996
15	3	12387	10231
	4	12387	10128
	5	12387	10482

Table 3.7: The average size of different set of containers in pre-dump with threshold level 80%

Threshold 0.8	5 Containers	10 Containers	15 Containers
Data Migrated with Proposed Technique	3583	6793	10180
Standard Migration Technique	4367	8176	12387

Furthermore, in figure 3.9 again, the same three sets of containers are implemented with a threshold of 0.8. In the first set of 5 containers, the probability-based PSO pre-dump size is (3654, 3457, 3316, 3723, 3765) with an average of 3583B, but in the standard approach, it is 4367B. With the proposed technique, it is reduced by 17.95%. Therefore, in a set of 10 containers, the proposed pre-dump size is 6793B which originally was 8176B but got reduced by 16.91%. In the set of 15 containers, the actual pre-dump size is 12387B, and the proposed technique is 10180B which is reduced by 17.81%.

The average of all the test cases with a threshold level of 0.8 is mentioned in Ta-

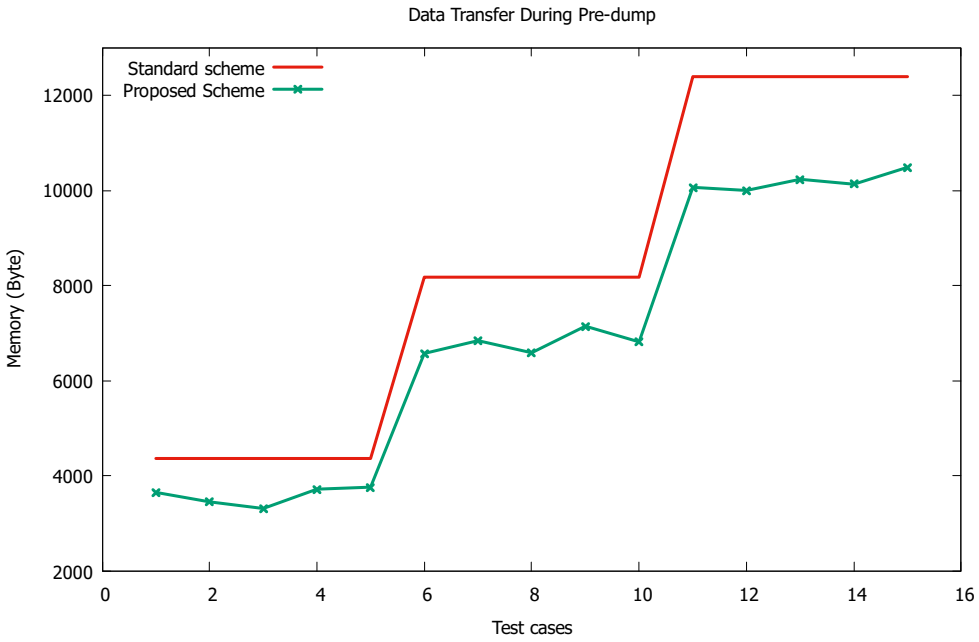

Figure 3.9: The size of pre-dump using probability-based PSO with threshold level 80%

Table 3.8: The average amount of data transfer in all set of container migration with various threshold levels

Average	5 Containers	10 Containers	15 Containers
Data Migrated after probability-based PSO	3206.7	6079.4	9040.5
Standard Migration Technique	4367	8176	12387

ble 3.7. The difference in data migrated in both the approaches is evident in Figure 3.10. Therefore, the average of 5 containers is 3206.7B, 6079.4B for 10 containers, and 9040.5B for 15 containers.

The overall performance of probability-based PSO is discussed in Table 3.8 and shown in Figure 3.11. To reiterate, it can be concluded that the present study started with the status of pages related to containers. Then, the observations were focused on two main parameters; whether a page is currently active or not and the update rate. The sending pool and non-sending pools are finalized based on these two parameters. The probability-based PSO is designed to improve further the results, which provides an accurate set of pages to be migrated. Further, each page is represented in the form of a particle, to use the updation rate of pages instead of random numbers with this algorithm. If the particle is 1, it means that the page will be migrated, and if the particle

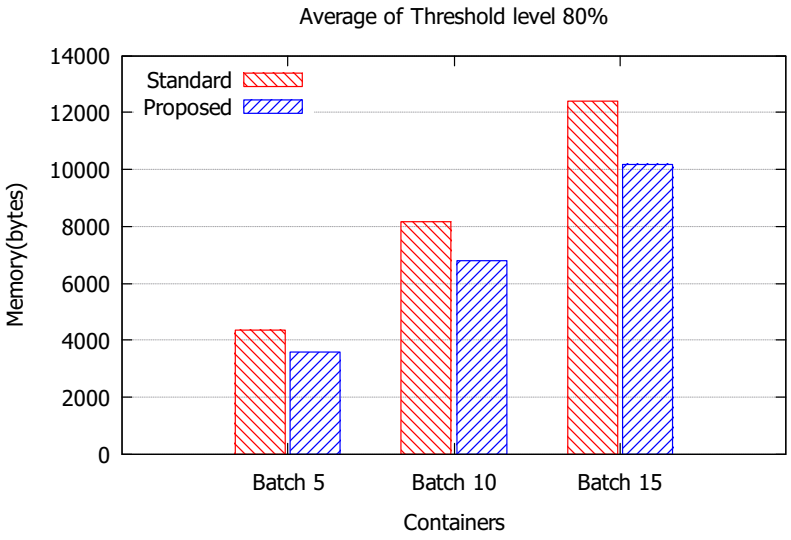

Figure 3.10: The average size of different set of containers in pre-dump with threshold level 80%

is 0, it will not be sent to the destination in the current phase. This process is repeated for every single container and to all its memory pages.

Now, compare the pre-dump size of all the sets of containers in various container migration techniques such as cold migration, pre-copy migration, post-copy migration, Hybrid migration, and VM migration technique with the proposed pre-copy migration technique. The result shows that in the case of cold migration and post-copy migration schemes, they are not participating in pre-dump because no data is dumped in the pre-dump phase. In post-copy methods, the dump will be started after the container is migrated to the host, and in cold migration, the container stops at the source, and then the complete dump is migrated to the destination. In Figure 3.12, a comparison of the amount of data transferred after the pre-dump phase of migration with four migration schemes where VM is tested with the batch of 5 and pre-copy (LXD/CR), hybrid technique, and proposed pre-copy technique with probability-based PSO is implemented in the batch of 5,10 and 15 containers. Compared to container migration techniques, VM is sending a large amount of data. Where in containers, the maximum data is migrated in the case of hybrid. The dump also migrates the container state. Moreover, the pre-copy is the second-largest pre-dump size because it transfers the complete dump. However, in the proposed pre-copy, the pre-dump size is smaller compared to other methods because it sends the shortlisted pages according to the probability-based PSO.

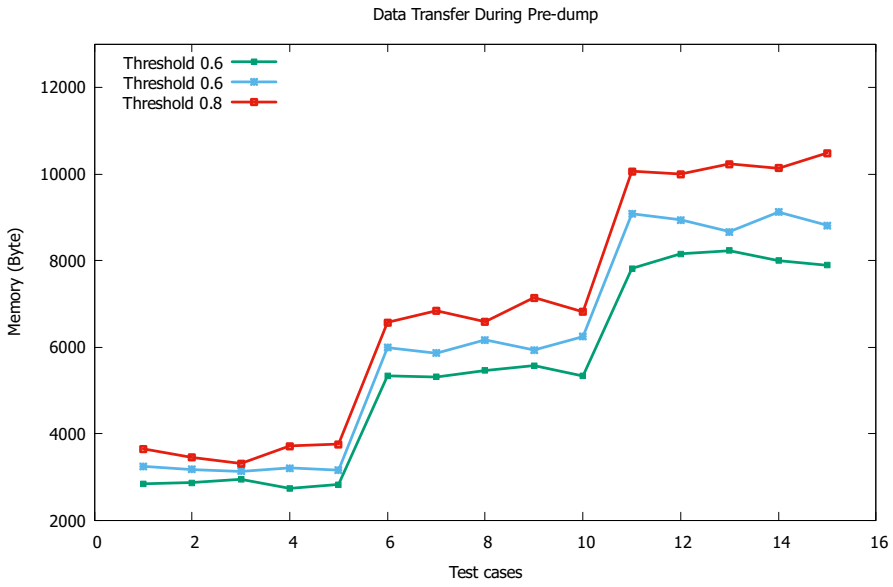

Figure 3.11: Comparison of pre-dump size with all the threshold level 60%, 70% and 80% with 5 test cases of each batch (5,10 and 15 containers).

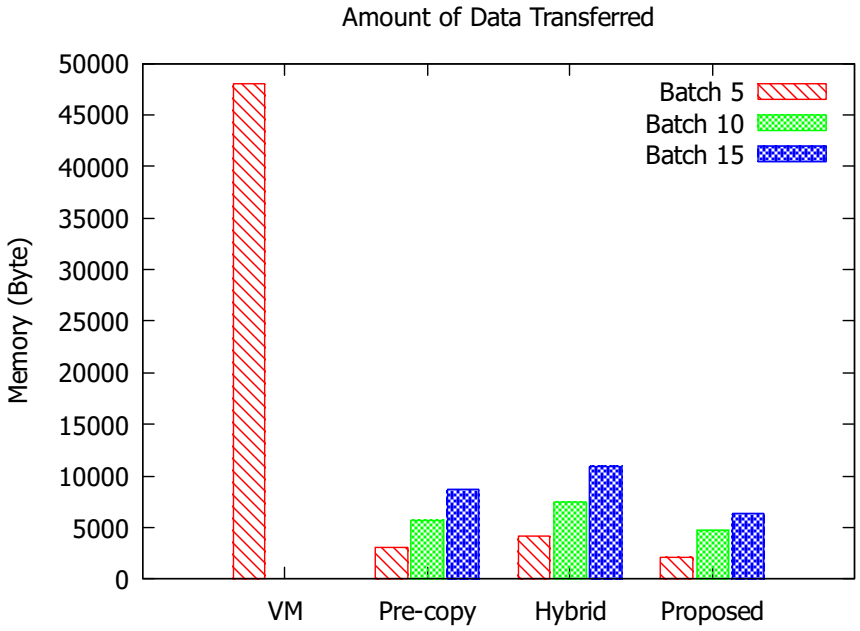

Figure 3.12: Comparison of amount of data transferred after the per-dump phase of migration with four migration schemes where VM is tested with the batch of 5 and pre-copy (LXD/CR) , hybrid technique and proposed pre-copy technique with PSO are implemented in the batch of 5,10 and 15 containers.

The remaining pages are migrated in the next phases before the container stops at the destination host.

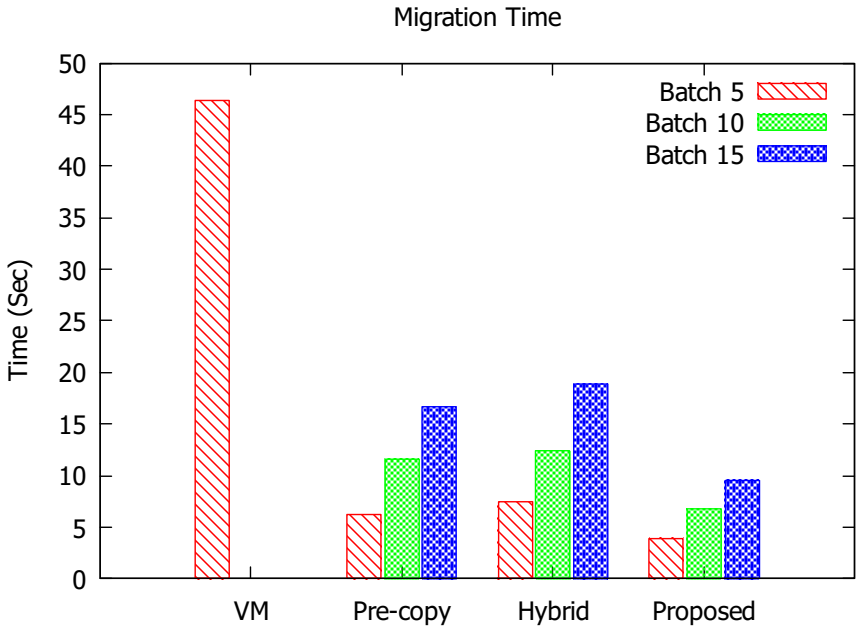

Figure 3.13: Time taken to migrate pre-dump is compared with four migration schemes where VM is tested with the batch of 5 and pre-copy (LXD/CR) , hybrid technique and proposed pre-copy technique with probability-based PSO are implemented in the batch of 5,10 and 15 containers.

The most important factor affecting the performance of the container migration process is the time to migrate to the destination host. Here, pre-dump time is discussed which further affects the overall time. In Figure 3.13, the time taken to migrate pre-dump is compared with four migration schemes where VM is tested with the batch of 5 and pre-copy (LXD/CR), hybrid technique and proposed pre-copy technique with probability-based PSO are implemented in the batch of 5,10 and 15 containers. In the case of cold migration and post-copy migration, no data is dumped in the pre-dump phase, and the time taken is zero. As in post-copy methods, the dump will start after the container is migrated to the host. The maximum time is taken in the case of VM and from container schemes the hybrid approach because with the complete dump, it also migrated the container state, which takes more time over the network. Additionally, the pre-copy is in third place in terms of pre-dump time because it transfers the complete dump. In contrast, in the proposed pre-copy, the pre-dump time is shorter than other

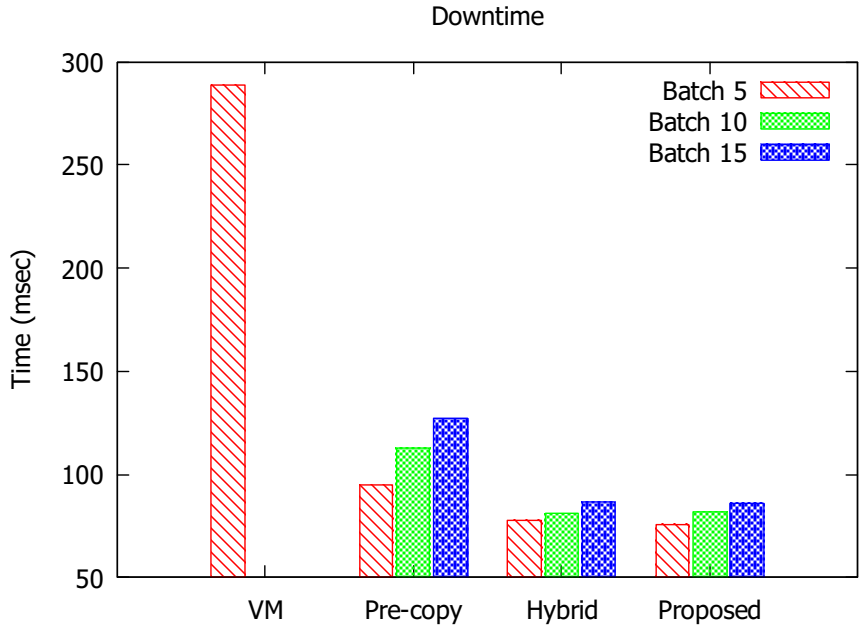

Figure 3.14: Downtime comparison of four migration schemes where VM is tested with the batch of 5 and pre-copy (LXD/CR) , hybrid technique and proposed pre-copy technique with probability-based PSO are implemented in the batch of 5,10 and 15 containers.

methods because it sends the shortlisted pages due to the small pre-dump size using probability-based PSO.

The probability-based proposed scheme outperforms in the case of migration time and size of pre-dump. As shown in Figure 3.14, the downtime of the overall migration process is reduced. Of course, the downtime will be more in the case of VM. But in container migration techniques, the proposed migrations techniques have less down-time, which is very close to the hybrid scheme.

3.6 Summary

The evolution of containers in cloud computing has changed the industry trends towards lightweight virtualization. There are various improved versions of migration techniques for containers. Yet, the migration phase is used in the same standard approach in the pre-dump phase. All the memory pages related to the container are sent to the destination host. This is a primary objective to reduce the data transmission during this phase. We can achieve a reduction in pre-dump size. In the standard approach, if we migrate

a set of 5 containers to the destination host, then 4367 B is transmitted, but in the proposed approach, it is reduced to 3206.7B approximately. In the same way, in the set of 10 containers, it was 8176B, which is now reduced to 6079.4B, and in the set of 15 containers where it was 12387B, now it is just 9040.5B.

Probability-based PSO reduced the size of the pre-dump phase, which led to less migration time, minimized the data transfer over the network, and minimized the resource requirement. Compared to the standard pre-dump migration technique of container migration, the proposed pre-dump technique outperforms with a 26.48% reduction in pre-dump size. In the future, this technique can be used in the next phases of container migration to minimize the overall dump size.

CHAPTER 4

A PREDICTIVE CHECKPOINT TECHNIQUE FOR ITERATIVE PHASE OF CONTAINER MIGRATION

Containers are lightweight virtual environments that share the host operating system's kernel. Containerized services are essential for reducing data transmission, cost, and time, among other things. The primary factor affecting the performance is the amount of data transferred over the network. It has a direct impact on migration time, downtime, and cost. This chapter presenting a predictive iterative-dump approach using LSTM to anticipate which memory pages will be moved by limiting data transmission during the iterative phase. In each loop, the pages are shortlisted to be migrated to the destination host based on predictive analysis of memory alterations. Dirty pages will be predicted and discarded using a prediction technique based on the alteration rate. The results show that the suggested approach surpasses existing alternatives in overall migration time and amount of data transmitted. There is 49.42% decrease in migrations time and 31.04% reduction in the amount of data transferred during the iterative phase[1].

[1]This chapter is derived from:

Gursharan Singh, Parminder Singh, Mustapha Hedabou, Mehedi Masud, and Sultan S. Alshamrani. "A Predictive Checkpoint Technique for Iterative Phase of Container Migration." Sustainability 14, no. 11 (2022): 6538.

4.1 Introduction

Cloud computing is a cost-effective method of delivering numerous services in Industry 4.0. The demand for dynamic cloud services is rising day by day, and because of this, data transit across the network is extensive. Virtualization is a significant component, and the cloud servers might be physical or virtual. The majority of businesses are transitioning from virtual machines to containers [91].

Since software emulates components of a system, virtual machines have long been the primary means of delivering virtualization in the industrial Internet of Things (IIoT). Virtualization permits activities to be performed in isolated environments, allowing for greater consistency because an emulation abstracts the entire underlying system [6]. Containers have emerged as a viable alternative to virtual machines in the past few years [92]. Containers existed previously, but they witnessed a significant surge in popularity when the container framework Docker was introduced in 2013. Docker introduced capabilities that allowed users to quickly construct, distribute, and build upon each other's containers, which helped their growth since users could utilize pre-existing containers [93].

Containers are frequently viewed as lightweight virtual computers with short boot times and low resource consumption [8]. One significant reason for this is that containers, unlike virtual machines, run on the host machine's kernel, as shown in Figure 1.1. This is advantageous in multi-tenant cloud providers and data centers since each bare-metal system is likely to run more instances, and instances may be launched or restarted more effectively, resulting in a much higher quality of service [94]. Migration is a critical technique in the context of virtual machines and containers [9]. The process of moving an instance of a container that is running across hosts is known as migration [10]. Depending upon the nature of the task or according to the customer's demands, a container migration can be live or non-live. While moving an instance, live migration means the user is unaware of this migration. The state of the container is migrated before the container migration [12]. This technology is critical in various virtualized settings because it allows instances of virtual machines or containers to be transferred across hosts while retaining state, enabling effective load balancing and more

(https://www.mdpi.com/2071-1050/14/11/6538)

straightforward maintenance with minimum impact [95].

subsectionLong Short-Term Memory

Long short-term memory networks are used to learn order dependence in sequence prediction tasks [96]. This has been shown to overcome the Recurrent Neural Network (RNN)'s vanishing gradient constraints [97]. LSTM is ideally suited to forecasting time series data because of its capability. The LSTM network comprises a series of interconnected LSTM units/cells.

The cell state, shown as dotted lines in Figure 4.1, is the most significant component of the LSTM because the data from the gates is retained. The LSTM is divided into three layers known as gates: forget, input and output. The forget gate decides whether the data coming from a previous timestamp is relevant to remember or appropriate to forget. The input gate tries to learn and provide new data to the cell. The third is the output gate, which passes the data for the current cycle to the next timestamp.

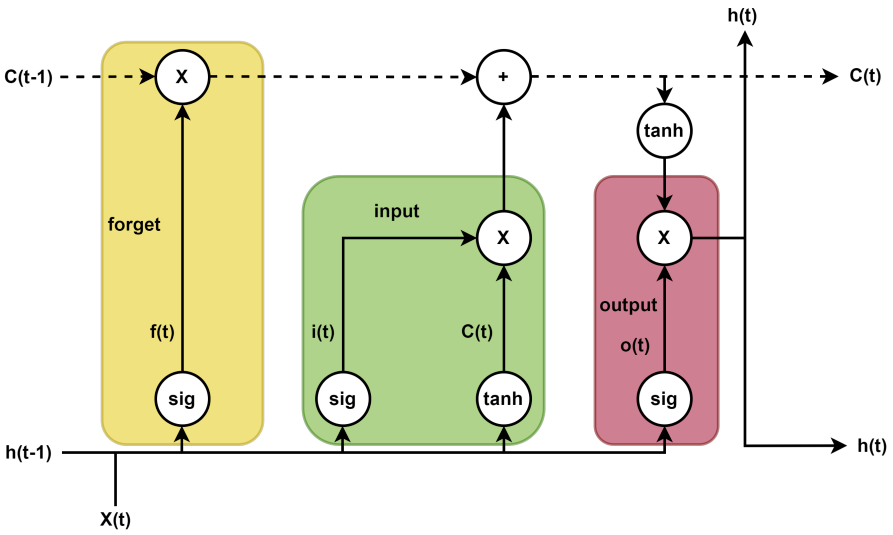

Figure 4.1: Architecture of long short-term memory [3].

LSTM has a hidden state as well, where $h(t-1)$ represents the hidden state of the previous iteration and $h(t)$ provides the hidden state of the current iteration. Both these handle "short-term memory". In the same way, the cell state of the previous iteration is $C(t-1)$ and for the current iteration is $C(t)$. The cell states handle "long-term memory".The unit processes the input data for each input vector to the LSTM network as follows:

1. A new vector will be created by adding the h_{t-1} (hidden state vector) and the x_t

(input vector). The newly created vector will be input to the *tanh* function and the three gates.

2. The flow of previously stored cell states is regulated by the forget gate:
   $f_t = sig(W_{hf} * h_{t-1} + x_t * W_{xf} + b_f)$
   where W_{hf} is the weight of the previous hidden state h_{t-1} and W_{xf} is the weight of the input. x_t is the input at timestamp t and b_f is the bias parameter. Further, by applying a sigmoid function, the f_t ranges between 0 and 1. If f_t = 0, then forget everything and if f_t = 1 that means forget nothing.

3. The input gate quantifies the data from input:
   $i_t = sig(W_{hi} * h_{t-1} + x_t * W_{xi} * b_i)$
   where, W_{hi} is the weight of input for the hidden state h_{t-1} and W_{xi} is the weight of the input. x_t is the input at timestamp t. The sigmoid function manages the input value between 0 and 1.

4. The new data needs to be passed to the cell state:
   $N_t = tahn(W_{hc} * h_{t-1} + x_t * W_{xc} + b_c)$
   Here, the activation function is *tahn*, which manage the value of N_t between -1 to 1. If it is positive, then data will be added to the cell state, and if negative, the data is discarded from C_t.

5. The final calculated C_t is:
   $C_t = f_t * C_{t-1} + i_t * N_t$

6. The output gate determines how much C_t is passed to the next cell. The hidden state h_t is calculated as follows:
   $o_t = sig(W_o * h_{t-1} + x_t * b_o))$
   $h_t = o_t * tanh(C_t)$

The output gate is further subdivided. The sigmoid function is applied to the filter $h(t-1)$, $h(t)$, and these are used to scale the values of the vector generated from the tanh cell, which manages the values from -1 to 1. Then, the product of the filter mentioned above and the vector is the output state for the next cell state.

Various migration techniques has been surveyed and pointed out their advantages and disadvantages [82].

1. The pre-copy migration technique is chosen to perform live migration. It is preferred for live container migration, but the factor affecting its performance is the amount of data transferred in the iterative dump. The proposed work is carried out in the iterative phase only. The pre-dump phase is discussed in our previous research paper.

2. A predictive container migration technique is proposed for the iterative dump to minimize the data transfer over the network.

3. A prediction model is designed and implemented with LSTM.

4. Consideration of experimental results implemented in different scenarios and compared with other migration techniques shows that the proposed system outperforms other techniques.

The rest of the chapter is organized as follows: Section 4.2 describes the literature review and the research gap. The problem identification, along with the main objective of the study, is discussed in Section 4.3. The tools and techniques used in the proposed system and the prediction model using LSTM are discussed in Section 4.4. In Section 4.5, the evaluation of the system model is elaborated in detail and concluded in Section 4.6.

4.2 Related work

The live migration of containers is prevalent, and challenges also arise with the increasing popularity and broad adoption. The container's size is already smaller than virtual machines, but containers' performance can still be enhanced further. The most influential factor is the data migration over the network, which directly affects the cost and performance of container migrations in a lightweight environment. There are several techniques proposed to minimize the dump size.

C Puliafito et al. [15] have carried out a detailed evaluation of various migration techniques under four parameters like total migration time, downtime, dump-time, and amount of data transferred data. They recognized several situations and suggested which strategy would be most suited to them. The findings demonstrate that the cold migration suffers from significant downtime, whereas hybrid migration suffers from a

longer overall migration time. Pre-copy and post-copy migrations may thus be the best alternatives under specific scenarios.

A successful migration is needed to restore immense data structures following a disaster. Several standard memory compression methods are mentioned and studied from a memory standpoint, including RLE, Huffman Coding, and a novel strategy for minimizing migration time. There are numerous difficulties to consider regarding memory compression, and an effective scheduling strategy is suggested [12], but the compression overhead affects migration time. To provide load balancing, the container is scheduled on the relevant server. The core implementation of dynamic migration scenarios has been developed for IoT [35], the main motive is resource provisioning. It can be more effective if it is manageable to reduce memory transfer. Moreover, to reduce the data transfer, prediction methods can be applied [47]. It can be reduced further with LSTM as suggested by [53]. The reuse distance concept is used, and the changed memory pages are traced back during the copy process [54]. A model includes clusters, containers, and micro resources with four optimization goals. The experimental results show that this strategy is a solution to the issue of container allocation and flexibility, attaining higher ethical standards than Kubernetes container management regulations. Each method's implementation scenarios are examined in [55]. For containers, precopying is the predominant way of migration.

Elghamrawy et al. [30] described the fact that there is a significant discrepancy between distinct prediction systems in the behavior of current memory pages. That is the hole they want to fill. They characterize the behavior of memory pages using a prediction approach for relatively stable memory pages and using the memory page characterization to prioritize specific pages with live migration since these pages will be updated gradually in subsequent cycles. The genetic algorithm technique employing the non-dominated Sorting Genetic Algorithm is recommended to maximize container assignment and management elasticity to the degree that this algorithm has produced good results on other cloud management challenges [98].

Mirkin et al. [56] The OpenVZ container Checkpoint and Resume (CR) mechanism were provided. This function allows the container to scan and restart programs and network connections. Checkpoints and restarts work with the kernel directly to reduce service delays and the size of the dump file.Dump size can be further reduced for memory-

intensive applications. Molto et al. [34] designed a hybrid distributed computing for VMs and containers for a better synchronization among hosts. In memory-intensive applications, the repetition of memory may increase [99].

The live container just-in-time migration service was designed following OCI standards introduced by Nadgowda et al. [46]. Containers run on the server with the shared file system, and some are with the local file system. CRIU helps perform a live migration of containers and ensures the destination host's restart. A union mount file system and CRIU helps to reduce migration time and data transfer.

Luo et al. [44] developed a technique based on data compression and de-duplication. The authors employed the RLE technique and the runtime storage image identity to reduce duplicate memory data. Hash-based fingerprints were employed for page similarity calculations. LRU, Hash tables FNHash and FPHash were used for implementation. The efficiency of migration has increased in terms of space with the overhead of CPU resources.

During the incremental copying procedure, the primary approach of pre-copy transfer memory pages are duplicated repeatedly at high replacement rates is provided by Ansar et al. [45]. Thus, the article developed an optimized pre-copy method (OPCA) with a Gray-Markov model. They shortlist the pages based on modification rate, which decreases the number of iterations and other parameters. This increases the resource utilization, which is managed by using hot and warm working sets to categorize the pages. The pages only from the HW set will go through the process. Chronopoulos et al. [57] show the effective use of machine learning to build an artificial neural network for speech-language therapy.

There are other techniques as well, suitable for predicting memory. Sobia Pervaiz et al. [73] conducted a detailed review of various Variants of PSO and highlighted the key features of every variant. It helps to identify the best-suited variant of PSO depending on the nature of the problem. Waqas Haider Bangyal et al. [74] have used a unique quasirandom sequence termed the WELL sequence to initiate the PSO particles. The velocity and position vectors of particles are changed in random order. In [75] presented three sequence strategies Torus, Knuth, and WELL. All these techniques are tested with low-discrepancy sequences. The results show that the WE-PSO strategy outperforms the PSO, S-PSO, and H-PSO approaches. The result shows that the proposed techniques

outperform standard PSO and other variants.

Moving a container application can be accomplished using various container migration techniques, and to increase the performance, various alternatives are there, as shown in Table 4.1. Because of the container's limited lifespan, they used a pre-copy strategy to facilitate the migration procedure. [27].

Table 4.1: A prediction based comparison on various container migration techniques

Ref.	**Migration Technique**	**Prediction Method**	**Achieved**	**Outcome**
[12]	pre-copy	RLE, Huffman Coding	memory compression	Compression overhead effects migration time
[35]	CloudIoT	LXC	vertical offloading	main motive is resource provisioning
[47]	pre-copy	ARIMA	Predict dirty pages, reduce and compress	can be reduced further with LSTM [53]
[54]	pre-copy	ARIMA	Memory forecast	can be reduced further using containers with LSTM
[55]	pre-copy	LXD	Container resource management	Resource provisioning
[56]	pre-copy	OpenVZ	Incremental Checkpoint	dirty pages transmission can be minimized
[27]	pre-copy	Gray-Markov prediction model	reduce iterative cycle	it shortlist the active pages
[53]	pre-copy	LSTM and ARIMA	dirty page prediction	600 times faster prediction time than ARIMA
[76]	Pre-copy	LSTM	reduce the size of iterative-dump	amount of data transfer is reduced by 31.04%

4.3 Motivation and Research Gap

Containers provide a robust environment for virtualized computing. The choice of container migration mechanisms significantly impacts container migration performance. The comparison demonstrates that pre-copy and hybrid copy migration approaches outperform post-copy migration techniques. This work is separated into three steps based on the pre-copy live migration technique: pre-dump, iterative dump, and final dump. During these phases, the container is running to accomplish live migration in Industry 4.0 applications. All of these strategies were covered in [82].

When it is determined to migrate a container using the existing technique of pre-copy container migration, then all the associated memory pages and their configuration are sent to the destination host in the pre-dump phase. Further, in the iterative phase, all the updated pages will be migrated in every iteration, and there are chances that a single page may be migrated several times. This overhead directly impacts several parameters, including the performance and the operating cost. In an iterative phase, the data transfer is hugely dependent upon the size of the container and the kind of operation it is handling. Sometimes containers send data more than the actual data size due to the re-transmission of updated pages. This is the primary factor affecting the performance. Our main objective is to lower the data transfer size across the network during the iterative phase.

An algorithm designed for the iterative phase of pre-copy container migration to minimize page transfer for cross-domain and cross-border Industry Internet of Things () applications. In this algorithm, instead of sending updated pages in every iteration, predicts the chances of modification in the subsequent few iterations to minimize re-transmission. If sending the active pages in the last few iterations is possible, the data transmission can be minimized.

If we focus on container migration for any reason, then the main concern is about container image and runtime, as highlighted in Figure 1.2. CRI handles all the container activities such as start, manage and stop. When there is a requirement to start a container, CRI takes control, and the CRI daemon and Runs work together to initiate a container. There is a total of three phases in pre-copy container migration.

To understand the scope of improvement, let us discuss the iterative phase at the

memory level. When a container runs on a machine, some memory is allotted to that container. These memory pages are in use by the container and modified frequently. When it is decided to migrate a container to a target host, the whole container memory set is transferred to the destination host at the initial state of migration. The same set of memory pages will be migrated repeatedly to synchronize the destination host memory with current changes on the source host, as illustrated in Figure 4.2.

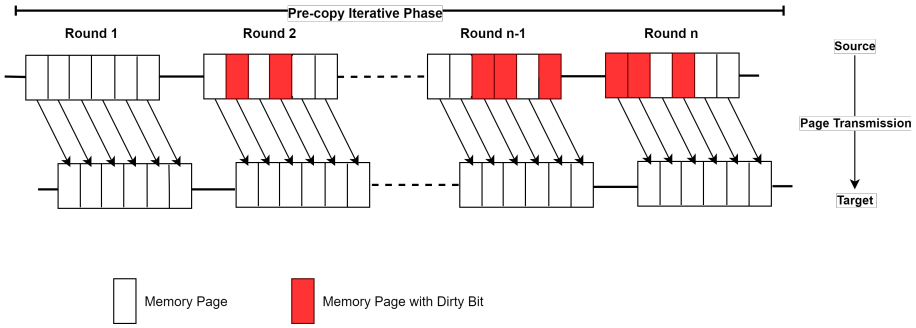

Figure 4.2: Process of transferring pages in an iterative phase of existing approach.

Nowadays, Industry widely adopts containers, and researchers are also working on the same. If we talk about the latest version of pre-copy migration, then there is no need to retransfer the same memory page in the iterative phase. Depending upon the data updated in the last iteration, it can be decided whether a page will be migrated to its destination or not, as shown in Figure 4.3. So in this Figure, the iterative phase is explained with the number of rounds. Each round handles a set of memory pages represented with a rectangular box and subdivided into individual pages.

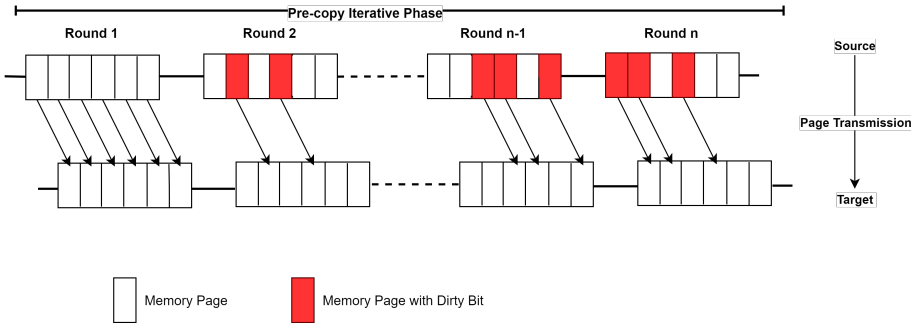

Figure 4.3: Process of transferring pages in an iterative phase according to dirty bits in the existing approach

Furthermore, the pages have categorizations as follows:

1. Pages with no color denotes no updation compared to the previous iteration.

2. Pages with red color denotes that they are updated after the previous iteration.

In Figure 4.3 you can see that in round 1, all the pages are transferred from source one to the target host because there was no updation in the memory pages. If we check round 2, two pages have dirty bits from this chunk of pages. Only these updated pages will be migrated to the destination host, whereas other pages will be discarded from this retransfer. The exact process will be followed in alliterative rounds, and this helps to minimize the data transfer during this iterative phase of pre-copy container migration.

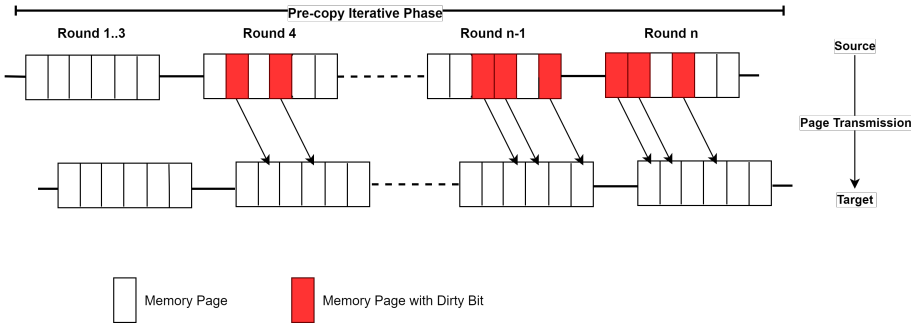

Figure 4.4: Process of transferring pages in an iterative phase according to dirty bits prediction in the proposed approach

A detailed review has been conducted for container migration techniques and identified the scope of enhancement in the iterative phase. We can further reduce the data transmission during the iterative phase. This method works on the concept of dirty bit and is based on the updation rate of pages. As you can see in Figure 4.4, in rounds 1 to 3, there is no page transferred from source to target host. It will wait for the first three iterations to analyze which page will be migrated to the source host and which page will be discarded.

To identify the pages that need to migrate will be shortlisted using the record of the last three iterations. If a particular page is in the update, discard that page from the dump, and if a page is not updated during the previous three iterations, that will migrate to the destination host. It has a significant impact on memory-intensive application containers. It will minimize the cost over the network by reducing the data transfer to the destination host.

4.4 Methodology

A new algorithm is designed, which helps to minimize the dump size. In this Algorithm 4, the first three iterations will only record the activities of every page. Starting from the fourth iteration, a decision will be taken according to the activity status of the last three iterations. If a page is not modified in the previous three iterations, that page will be added to S_{pool}; otherwise, the page remains inactive set. This process will be repeated for all remaining iterations. Then the status of pages will be provided to the proposed prediction scheme for the final decision of iterative-dump.

Algorithm 4 Iterative-dump/checkpoint

Result: Prediction of memory pages to be migrated in iterative phase

Access the memory pool of the container. Get the read-write status of every memory page in every iteration

while M_{pool} **do**

  if $DB(P_i) == 1$ **then**

    $P_i.active+ = 1;$ ▷ Page will be added to active pool

  else

    if $P_i.active \geq 1$ **then**

      $P_i.active = 0;$

    else

      $P_i.active- = 1;$

    end

  end

  if $P_i.active \leq -2$ **then**

    $S_{Pool}.append(Pi)$ ▷ Page will be added to send pool

  end

end

4.4.1 Machine learning-based predictive checkpoint

The LSTM is used in a network model to predict the memory load to be migrated. In this network model, the input cells layer is used as 3, LSTM layer units are 10, and 1 output layer cell as shown in Figure 4.5. The input layer of this network model receives the modification data of the previous three iterations to predict the chances of modification in the next iteration. This model aids in generalizing the training data, allowing us to adjust the input size.

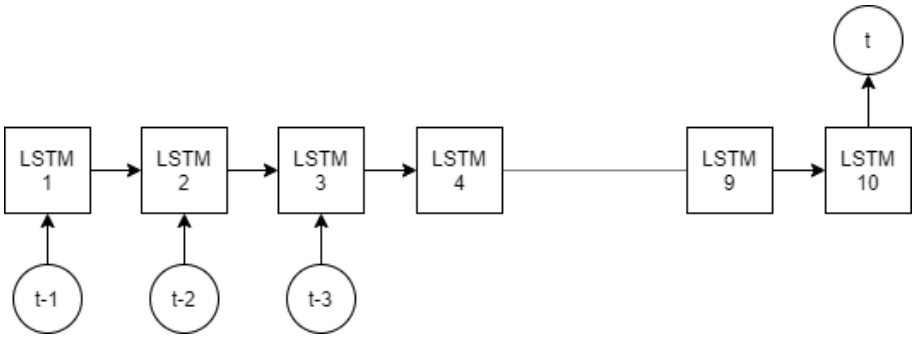

Figure 4.5: LSTM network architecture [4]

The architecture of the prediction model is designed using LSTM, that works on-page history generated by a prediction model. This model works with three inputs that interact with the hidden layer with a size of 10 units and finalizes the result with a single output layer.

Table 4.2: LSTM model configuration parameters

Parameters	Range
LSTM units	10
Kernel initializer	Random uniform
Loss function	MSE
Batch size	64
Size of input	3
Layers of LSTM	1
Size of Output	1
epochs	10

Parameter's description is mentioned in Table 4.2. This works in a serial order after the process of Figure 4.6. The loss function for training LSTM prediction models is mean square error. The sample input is divided into a batch size of 64, which works with a maximum of 10 iterations. Then these iterations with the weighted input will be further processed in each epoch. To set the initial random weight, the initializer used is "RandomUniform" and the bias initializer is "zeros".

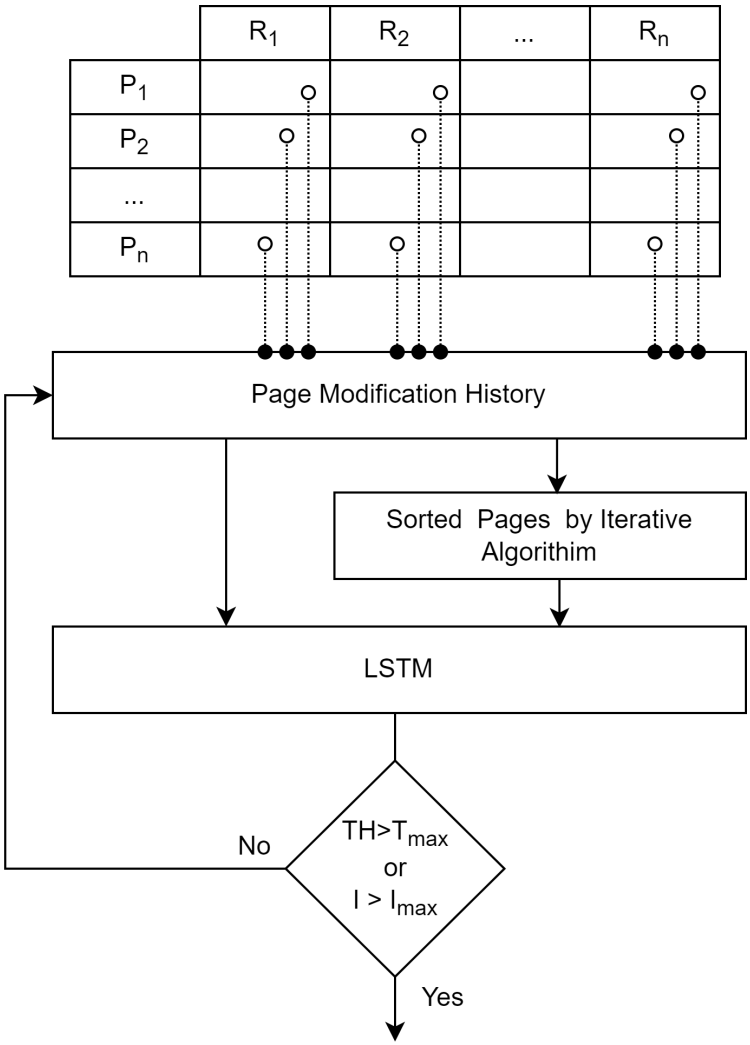

Figure 4.6: The proposed prediction model to shortlist the pages to be transferred

The suggested approach is significantly minimizing the size of the checkpoint. In this approach, the active set of pages minimizes the re-transmission of pages because the incremental checkpoint ignores dirty pages, frequent copying, and transmission. It has a direct impact on application downtime. As a result, this research proposes a delayed checkpoint approach based on the prediction of dirty pages, which can re-

duce application downtime by reducing the time spent copying and transmitting dirty pages regularly. Following is a detailed explanation of the proposed prediction model as shown in Figure4.6.

1. In every iteration of this prediction model, the activities related to the memory pages are recorded and stored in "Page modification History."

2. based on memory status, Algorithm 4 will shortlist the pages in $send_{pool}$.

3. These pages will be further passed to the input gate of the LSTM module, and it works as mentioned in Figure 4.1. It works on the provided input according to the mentioned epochs and generates an output with a timestamp.

4. Depending upon the condition of iterative-dump, if the stopping criteria do not match, the next iteration is initiated.

5. Now, the iterative algorithm again fetches memory pages' latest status and passes it to the LSTM module. Along with that, the previous timestamp output is also provided to the current state of LSTM.

6. In every iteration, the sorted pages after the LSTM module are migrated to the destination host.

7. When stopping criteria match, then the control shift to the final dump of the pre-copy migration technique.

The parameter $(P_1, P_2 \ldots P_n)$ represent pages related to a container, and $(R_1, R_2 \ldots R_n)$ represents the iteration number. During this iterative phase, the dirty bit status of each page in every iteration is stored in "Page History." The same input will be provided to an iterative algorithm to sort the pages. Then both the inputs will be provided to the proposed LSTM module to predict the final set of pages to be migrated in the next iteration. The total number of iteration depends on the dynamic conditions, such as if the updation rate of memory pages reach the set threshold level or When the iteration count reaches the specified maximum limit, then the iterative phase ends immediately. The maximum threshold value "T_{max} is set as 70%, and the maximum iteration "I_{max}" is 10.

4.5 Results and Discussion

This prediction model was developed using a direct multi-step process to forecast the number of dynamic pages migrated in the following round. Cloudsim 4.0 libraries are used to simulate the environment for container migration, and the Keras library manages the implementation of these prediction models with the help of DL4J. LSTM is used to predict short-term memory changes. The Tensor Processing Unit () was used to train the model. The Mean Squared Error () is used to measure the degree of errors. If there is no error, the MSE generates a value of 0. Calculating the average time to predict after ten attempts, the model will be examined using the mean square error.

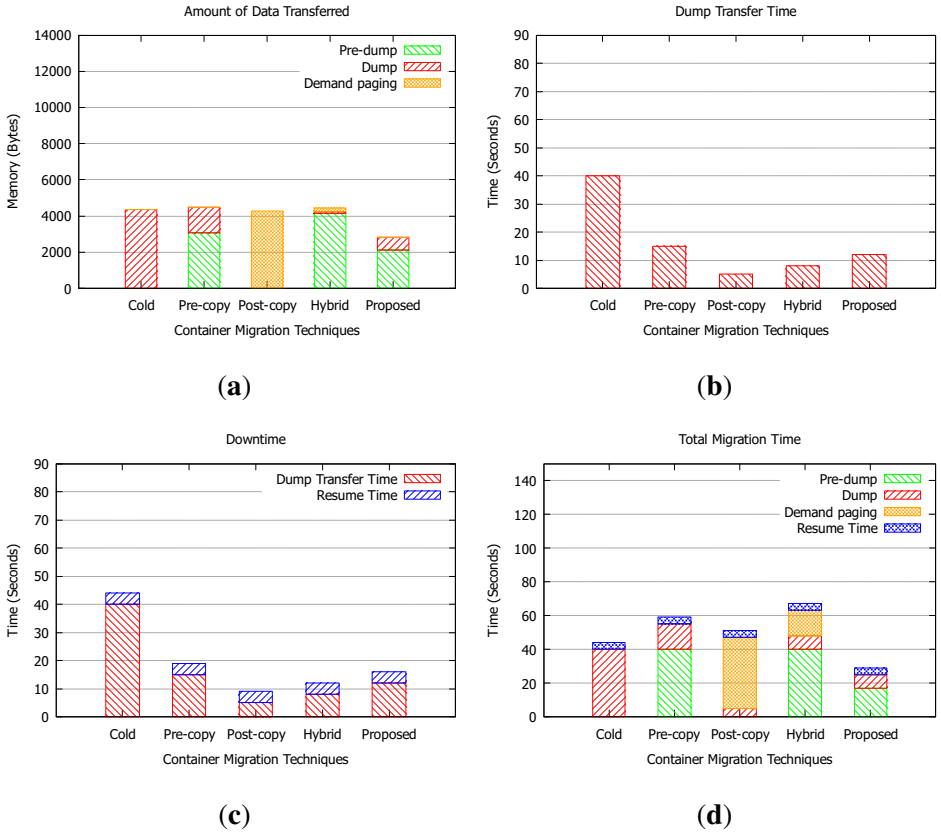

Figure 4.7: Comparison of various container migration techniques with batch of 5 containers where a). shows the total amount of data transfer during iterative-dump, b). shows the time to transfer the iterative-dump, c). indicates the overall downtime of a container including resume time on destination host and d). shows the total time taken during the migration process.

There are some container migration techniques and Figure 4.7 shows the compar-

ison of various container migration techniques with a batch of 5 containers where a). displays the total amount of data transfer during the iterative dump, b). shows the time to transfer the iterative dump, c). indicates the overall downtime, including resume time on the destination host, and d). shows the total time taken during the migration process.

The proposed container migration technique is implemented as a pre-copy. Compared to the existing pre-copy, the amount of data transfer is reduced in the proposed approach, dump-time, downtime and total migration time are also minimized.

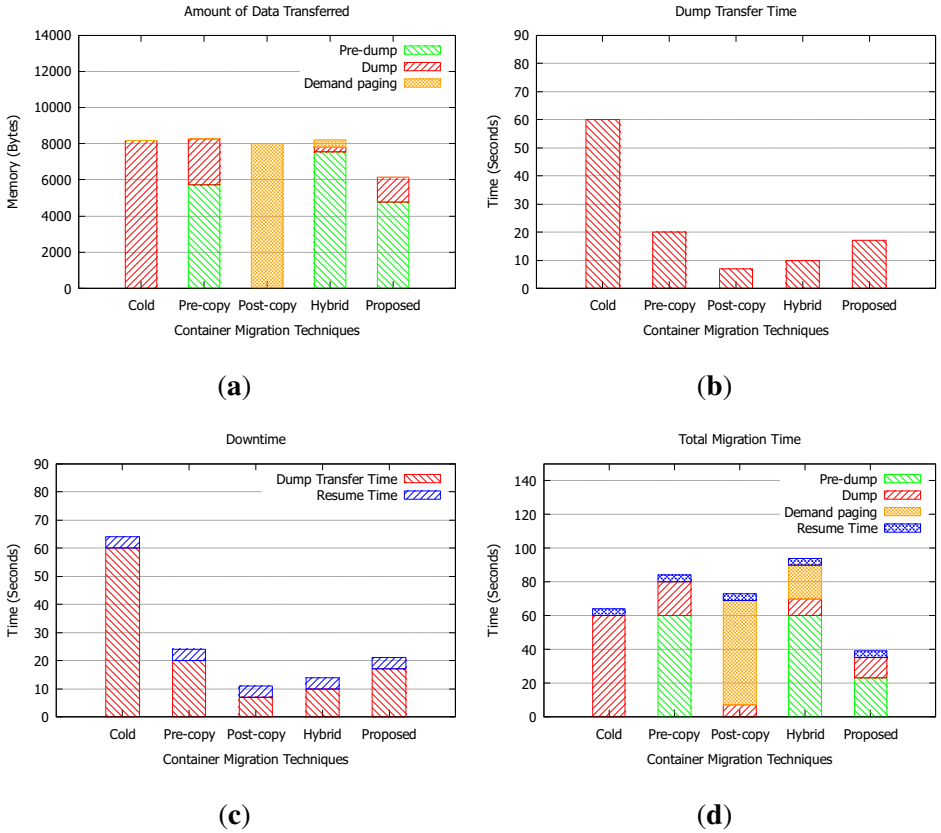

Figure 4.8: Comparison of various container migration techniques with batch of 10 containers where a). shows the total amount of data transfer during iterative-dump, b). shows the time to transfer the iterative-dump, c). indicates the overall downtime of a container including resume time on destination host and d). shows the total time taken during the migration process.

The proposed container migration technique is implemented as a pre-copy compared to the existing pre-copy; the amount of data transfer is reduced in the proposed approach, while the dump time downtime and the total migration time are also minimized.

We are implementing the proposed container migration technique as a pre-copy. In Figure 4.8 the comparison of various container migration techniques with a batch of 5 containers where a). shows the total amount of data transfer during the iterative dump, b). shows the time to transfer the iterative dump, c). indicates the overall downtime, including resume time on the destination host, and d). shows the total time taken during the migration process. Compared to the existing pre-copy, the amount of data transfer is reduced in the proposed approach, and dump-time downtime and total migration time are also minimized.

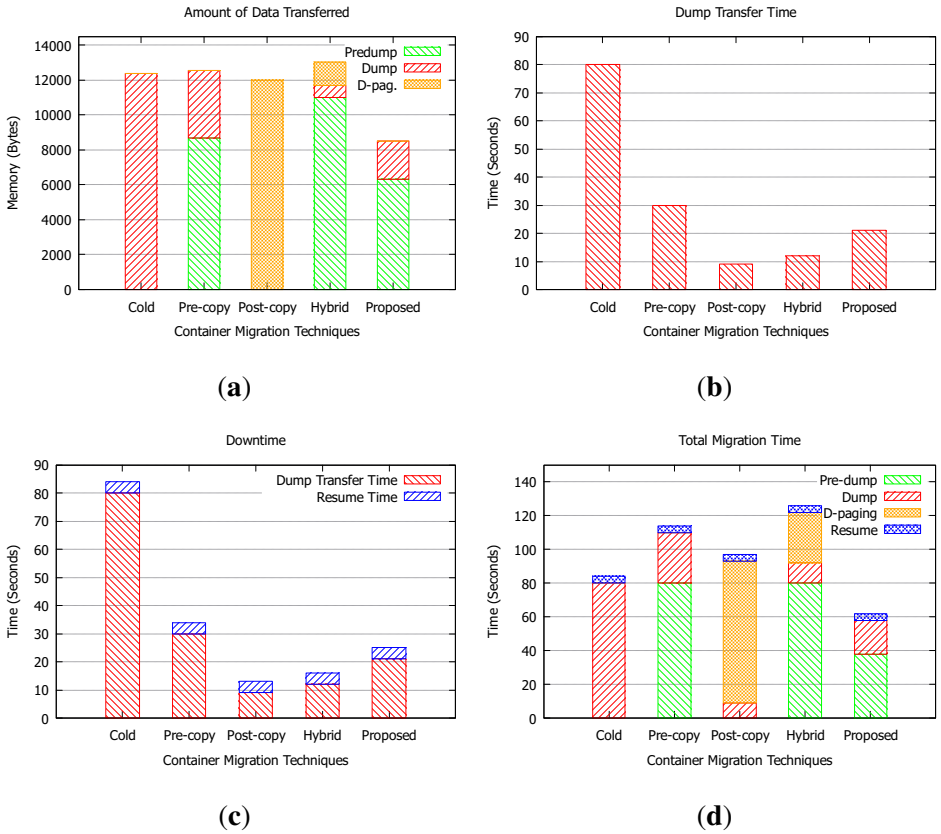

Figure 4.9: Comparison of various container migration techniques with batch of 15 containers where a). shows the total amount of data transfer during iterative-dump, b). shows the time to transfer the iterative-dump, c). indicates the overall downtime of a container including resume time on destination host and d). shows the total time taken during the migration process.

The results shown in Figure 4.9 indicates almost the same ratio as discussed in the previous two scenarios with batch size 5 and 10 containers. The pooled results of all three batches show that the proposed technique outperforms in comparison to existing

pre-copy. The post copy and hybrid technique show that downtime and dump-time are less than the proposed technique. In these techniques, some of the processes are carried out after migration, and that increases the amount of data transfer and the total migration time as shown in Figure 4.9(a) and Figure 4.9(d).

4.6 Summary

In this chapter, our approach minimizes the data transfer over the network. We have implemented the proposed approach as pre-copy migration. The result shows that the proposed approach outperforms existing approaches regarding the amount of data transferred and total migration time. In case of time taken to transfer the dump, the proposed technique takes less time than cold and pre-copy migration. However, the time taken by post-copy and a hybrid approach is lesser than the proposed. Same in case of downtime, the cold and pre-copy approaches downtime is more than the proposed, but post-copy and hybrid have lesser downtime. Because in both post-copy and hybrid, most of the data is transferred after the container is initiated on the destination host. The proposed technique outperforms for "total migration time" and amount of data migrated," but post-copy gives better results in the remaining two parameters(downtime and dump-time) as shown in Table 4.3.

Table 4.3: Comparison of various container migration techniques on the basis of migration time, downtime, dump-time and amount of data transfer

	Total Migration Time	Downtime	dump-time	Data Migrated
Cold	64.00	64.00	181.00	8310.00
Pre-copy	85.67	25.67	65.00	8427.67
post-copy	73.67	11.00	21.00	8091.00
Hybrid	95.67	14.00	28.00	8556.67
"Proposed Pre-copy"	43.33	20.67	51.00	5811.33

Nevertheless, compared to pre-copy only:

1. Total migration time is decreased by 49.42%
2. The downtime is decreased by 19.47%.

3. The dump-time is minimized by 21.53%.

4. The amount of data transfer is reduced by 31.04%

Because of the increasing demand for containers, the amount of data transfer is a crucial factor for overall performance. In future scope, the memory prediction in live container migration can be evaluated at a large scale on real scenarios for detailed insights. It can be tested with other alternative prediction schemes, such as ANN, ARIMA, etc., to establish future directions. Depending upon the type of application running in the container may affect the prediction results. We plan to extend our study to different types of applications.

CHAPTER 5

A DUMP REUSING TECHNIQUE FOR CONTAINER MIGRATION ALONG WITH A PAGE RECOVERY MECHANISM

In this chapter, we describe the proposed approach for container migration to migrate back to the same host. In some cases, the container will migrate back to the same host. The original image kept on the source host can be reused in such cases. The memory pages similar to the source image will not be sent back; only the updated pages will be transferred. This approach helps in reducing the amount of data transmission over the network. Furthermore, if the container image is kept on the source host, it will provide demand paging and help recover from failure at the destination host. The result shows the average rate of reduction in the data transfer over the network by 60.68% compared to standard pre-copy and 52.30% compared to advanced pre-copy[1].

[1]This chapter is derived from:

Gursharan Singh, Parminder Singh. "A Container Migration Technique to Minimize the Network Overhead with Reusable Memory State."IJCNA Volume 9, Issue 3, June(2022).

(`https://www.ijcna.org/abstract.php?id=1848`)

5.1 Introduction

Cloud computing is a new computing technique for massive data centers that keeps computational resources online rather than on local machines. As cloud computing grows in popularity, so does the need for cloud resources [100]. Container placements on physical hosts in Infrastructure-as-a-Service (IaaS) data centers are constantly tuned in response to the usage of host resources. When migrating a container to another host, the container image will be removed from the source host after the successful migration [20]. Container migration enables load balancing, system maintenance, and fault tolerance, among other things [101]. The need for cloud computing has eased the dynamic deployment of computing, networking, and storage resources to provide on-demand services [102]. Traditionally, the process of directly executing on the operating systems has been the core piece for hosting the service by employing the resources [103]. With the advancement of virtualization, one of the vital virtualization technologies used to host cloud services, virtual computers (containers) may share processing, networking, and storage resources from actual machines. Due to its flexibility and tiny footprint, the container, on the other hand, is the growing virtualization instance to offer a more elastic services architecture [6]. Under various Service Level Agreements (SLA's), application providers can lease virtualized instances (containers) from cloud providers of several types. The instances are then initialized by the container or container administrators, and the cloud broker or orchestrator chooses the best possible placement based on the available resources and the allocation rules [104].

The live migration of processes, virtual machines, containers, and storage is a critical component in cloud computing for supporting dynamic resource management. It can migrate and synchronize the operating state of an instance of the container from one host to another without affecting services. [105]. Live migration provides a general solution that does not require application-specific configuration or administration. Many studies have been conducted on resource utilization through live migration, such as fault tolerance, load balancing, system upgrade, hardware and software maintenance, etc. AWS, Azure, Google, IBM, RedHat, and other cloud service providers have begun to integrate live container migration.

5.1.1 Background

Migration management must reduce migration costs and maximize migration performance while meeting resource utilization goals. Live migration and virtualization are the leading performance factors of cloud computing. The virtualization is achieved with the live migration of container instances, where the type of migration and the amount of data transfer are the two main factors [106]. We highlight solutions for migration management and problems since dynamic resource management necessitates several instances of migrations to fulfill the objectives. In addition, we examine relevant state-of-the-art works and identify future research opportunities.

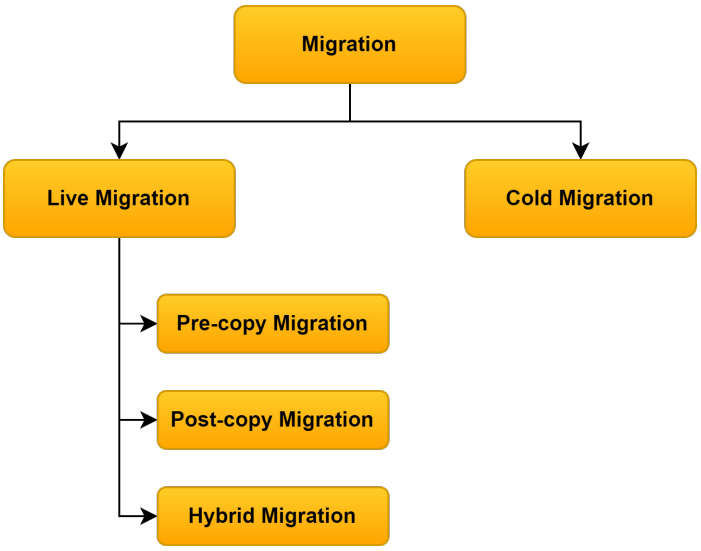

Figure 5.1: Categories of container migration

Virtualization

The virtual machine and the container are the two industry-standard solutions for virtualization used in live migration. This part introduces the container runtimes and the memory tracking method that enables isolation and virtualization of resources [107].

The container runtime is software that generates and maintains containers on a computing node. Apart from Docker, there are many others, including containers, CRI-O, and runs. The standard for live container migration is CRIU [108]. It uses ptrace to capture processes and inject parasite code that dumps the process's memory pages into the image file using its address space. Additionally, the state of the task assigned, registry entries, linked files, and the credentials are captured and preserved in the container's

dump files [22]. While a process tree is being checked, CRIU produces checkpoint files for each linked child process. CRIU uses information from the dump files created during checkpointing to restore processes on the destination host [109].

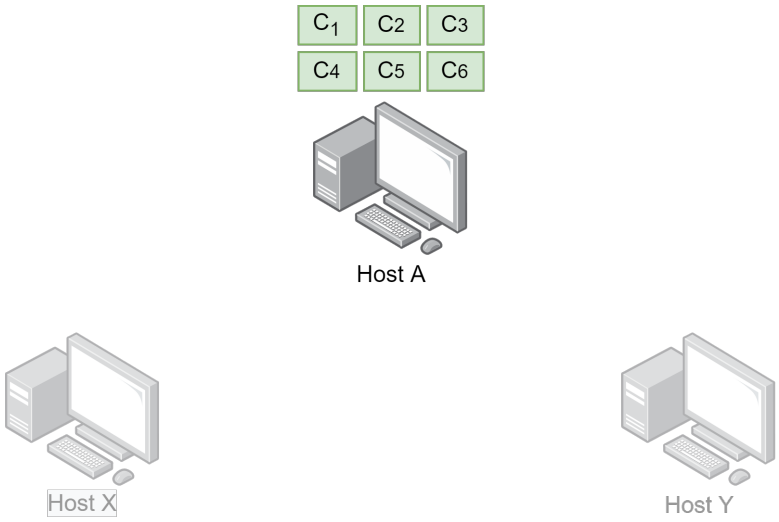

Figure 5.2: System configuration and container placement when source host is idle [5]

If a system is in the ideal situation, then it can handle the containers. When a system is overloaded, it will initiate the migration process of some instances (containers) to transfer them to another available host. This can happen in other situations, like fault tolerance, load balancing, system upgrade, etc.

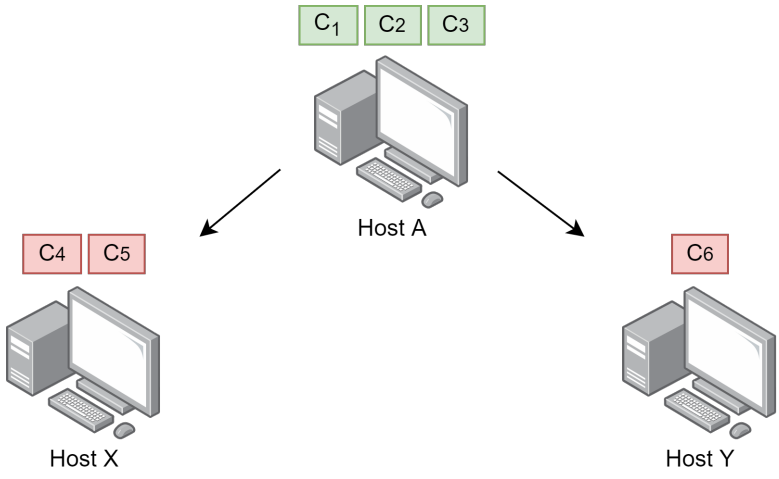

Figure 5.3: System configuration and container placement when source host is overloaded [5]

As shown in Figure 5.2, "Host A" is managing all the containers from C1 to C6.

But when the host system is overloaded, it may start migrating the containers to other available hosts. The scenario after the migration is shown in Figure 5.3.

Migration Types

Migration allows accessing the resources, processes, and containers virtually. The migration of instances and storage may be classified as "cold" and "live". There are three types of live migration: pre-copy, post-copy, and hybrid migration.

The system design of live migration and its continual adjustment and refinement aims to reduce total migration time and the amount of data transmission. The migration time for a single instance is the time between the start of the pre-dump phase and the completion of the post-migration phase during which the instance is operating on the destination host [77]. The time when the container service is unavailable due to synchronization requirements is counted as downtime. Memory transfer of container may be classified into three phases to do a performance trade-off analysis:

1. The push phase, during which the instance continues to execute on the source host and the related memory and data are pushed to the destination host across the network.

2. A stop-and-copy phase in which the instance is first stopped, then memory and data are transferred over the network to the destination. The instance will continue at the destination after the conclusion of the phase.

3. The pull phase, during which the new instance operates while retrieving faulty memory pages.

Cold Migration: Compared to live migrations, it includes transferring a single copy of memory and disc or a dump file for a single container checkpoint from the source to the destination host. In other words, this category includes the stop-and-copy approach. In comparison, cold migration is more straightforward than live migration. The amount of data transfer and the migration time directly depends on the nature of the task and the amount of data assigned.

Pre-copy migration: It transfers the container memory in multiple iterations. A single page is migrated many times depending o the number of iterations. Optimized

pre-copy on transferring the modified pages in these iterations. During this process, the container is running [2]. It is classified into the following phases:

1. Initialization: The first step is to select the target host to expedite subsequent migrations.

2. Reservation: configures the shared file server (optional) and initializes an instance container on the destination host for the reserved resources.

3. Iterative pre-copy migration: It transfers the modified pages to the destination host. In the initial round, the initial memory states are duplicated.

4. Stop-and-Copy: When this process meets the threshold value in terms of a number of iterations or amounts of data transfer, that is the last iteration.

5. Commitment: The sync of the source host will get the destination host's commitment to the successfully cloned instance.

6. Activation: The new instance is allocated reserved resources.

Post-copy container migration technique suspends the instance at the source and restarts the same instance on the destination host by transferring a container execution state and the remaining pages. If any page is unavailable at the destination host, the page fault occurs, and the same page will be accessed from the source host. Compared to pre-copy migration, a post-copy technique can significantly reduce the time taken by the migration process. The service will also break if the running instance fails since the originating host does not have a running example with its memory set. The main reason for the decreased performance is the steady demand for memory pages from the source copy.

To strike a balance between the three phases of live migration, Hybrid post copy [110] uses a hybrid approach. Pre-copy and post-copy migrations can also be used as an optimization approach. It all starts with the pre-copy approach, which copies dirty pages repeatedly. If the memory copy iteration fails to attain a specific percentage increase over the previous iteration, the post-copy migration will be activated. The migration time will be reduced in some cases, but the downtime will be somewhat higher. There

are certain downsides to post-copy migration, such as slower processing speeds and the possibility of container reboots if the network is unstable when extracting faulty pages.

This chapter is organized as follows: Section 5.2 describes the literature review and the research gap. The problem identification, along with the objective of the study, is discussed in Section 5.3. The tools and techniques used in the proposed system and the memory reusing model are discussed in Section 5.4. The experimental setup is discussed in Section 5.5. In Section 5.6, the evaluation of the system model is elaborated in detail and concluded in Section 5.7.

5.2 Related Work

Virtualization in cloud computing has dramatically improved due to the development of containers. When compared to virtual machines, container migration is much more efficient [111]. Cloud services are now migrating to a container environment, necessitating new research to improve this technique. Karhula et al. [112] Examined the notion of function as a service concerning IoT edge devices. We used the checkpointing method to conserve resources on resource-constrained devices to halt long-running blocking functions. Additionally, we successfully demonstrated live container migration using CRIU [76]. The assessment indicates good results for cloud device checkpoints capability. cloud systems can use these building pieces for fault-tolerant, offload resources, and boost efficiency and availability at the IoT edge. To freeze a running container, Jiaxin Feng et al. [113] describe the checkpoint/restore in userspace. Compared to the re-deployment technique, the restored container takes advantage of stateful migration to retain its state after being frozen. The CSS approach selects and migrates the container with the capacity(512MB). The container migration time is 48 seconds, which is significantly faster than the time required for standard minute-level wireless reconfiguration.

Janaina Schwarzrock et al. [114] highlighted the concept of virtual memory, like page faults and TLB, which increases control switching between hosts and affects the overall performance. Load balancing is presented using the migration method based on the two options. Compared to the conventional wireless reconfiguration technique, the migration mechanism may preserve the DU/state CU's while requiring minimal service interruption to handle memory adequately. The transition overhead establishes

a reliance between various local setups and should be considered in online techniques [114].

Ranjan Sarpangala et al. [115] described the development and operation of VAS CRIU, a novel technique for reducing memory snapshot and restoring time by utilizing task address space. After their first launch phase, applications may be snapshotted into a VAS and then immediately restored into fresh container instances. Microservices and NFVs are two use cases that we consider. Additionally, the snapshot might contain information about popular pages, which could be used for page pre-copying. A system that monitors containers' performance and the hosts on which they run. Analyzing, describing, and developing forecast models can benefit from this data. They have fine-tuned resource provisioning techniques by analyzing data from the monitoring system to establish a cloud provisioning platform that enhances container workload utilization and implementation through live migration. All the details of our lightweight resource monitoring tool, which allows for the offline and real-time examination of active migration workloads, along with the impact on their hosts, are described [116].

Florian Hofer et al. [117] propose a migration architecture and demonstrate, via the use of a custom-built orchestration tool, that containerized apps may run on shared resources without jeopardizing planned execution within specified time restrictions. They investigate the boundaries of three system configurations using latency and computational performance studies and write a summary. They were using Layrub, a data placement approach for GPU-accelerated deep learning. DNN models of all shapes and sizes may be trained using Layrub's extraordinary memory optimization. Experimentation has shown that Layrub can reuse a consistent amount of memory space no matter how deep the network is. The authors further highlight the advantages of Layrub by comparing it favorably with GeePS, vDNN, MXNet, and TensorFlow on many DNN models and datasets. Using Layrub might help you keep your memory as efficient as possible [118].

Evangelos Vasilakis et al. [119] Provides a novel data migration technique for hybrid memory systems that accounts for the overheads as mentioned earlier and significantly increases migration efficiency and effectiveness. It is based on discovering that migrating memory segments stored in the last level cache reduce migration load. Their

solution is based on the current status of the previous level cache to forecast reuse and prioritize memory segments for transfer [120, 121]. Thus, when segments are present in the last level cache, they are transferred at a lower cost. The results demonstrate that our technique beats existing state-of-the-art migration designs by 12.1% in terms of system performance and 13.2% in memory system dynamic energy reduction.

5.3 Problem Formulation

With the increasing popularity of containers in cloud computing, the data transmission also increases over the network, which leads to high network traffic.

Although the container helps to minimize the migration size compared to VMs [14], it can still be reduced further. The primary disadvantage of existing live migration approaches is that they need extensive data transfer to relocate a container. Transferring a massive volume of data introduces two complications:

1. The migration process results in memory accesses that decrease the performance of containerized apps.

2. Consolidating several containers onto a single host congests the host's network and slows the consolidation.

We suggest a memory reuse approach to minimize the quantity of data exchanged during live migration. A container may migrate back to the host on which it was previously operated. When the container migrates away from the host, the memory image is retained on the host, and the image is reused when the container migrates back to the host later. The reduced data volume results in a faster migration time and enhanced optimization via container placement algorithms. The container migration technique used in this method is pre-copy, and in the case of the pre-copy migration technique, the process is divided into three phases. These phases are called pre-dump, iterative dump, and final dump. As mentioned earlier, the method we will discuss comes after all three of these three phases. We will reuse memory in the event of container migration to reduce the data transmission over the network.

As you can see in Figure 5.4(a) shows the migration from source to destination, where the first three phases of pre-copy are applicable. In the case of standard pre-copy migration, all the memory pages or the related configuration are migrated to the

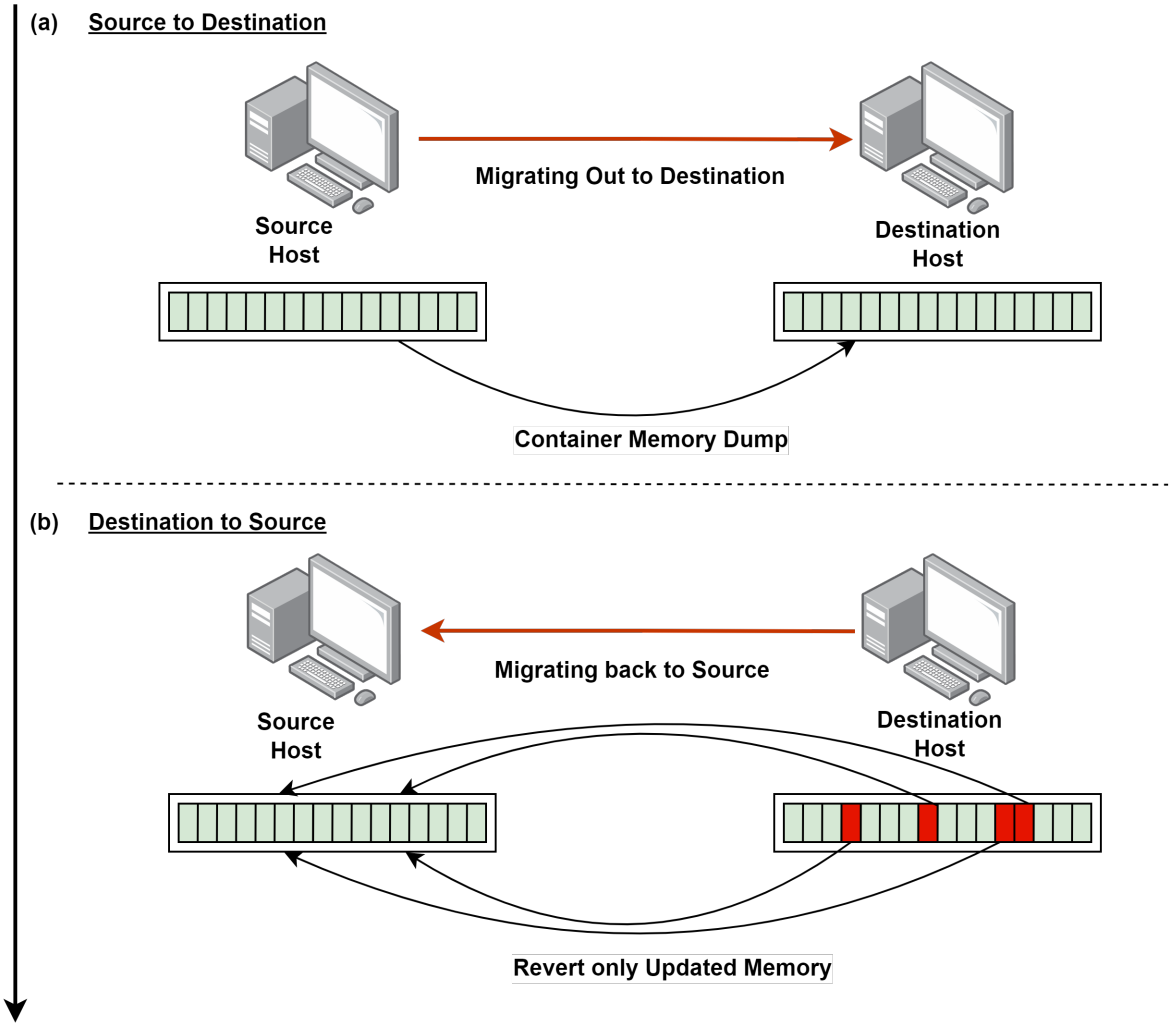

Figure 5.4: (a) The process of transferring the complete set of memory pages from source to destination host during container migration and (b) shows the process of transferring highlighted pages to the source host. Only modified pages will be migrated when migrating back to the same host.

destination host. Now we are going to extend this migration process further. Suppose a container is migrated to another host due to any of the following reasons: fault tolerance, system update, load balancing, or any other reason. In that case, there is a chance that the container will come back to the same host once the purpose of migration is achieved. In the case of standard techniques, while migrating back to the source host, We have to follow the same migration process of copying memory pages from the destination host back to the source host.

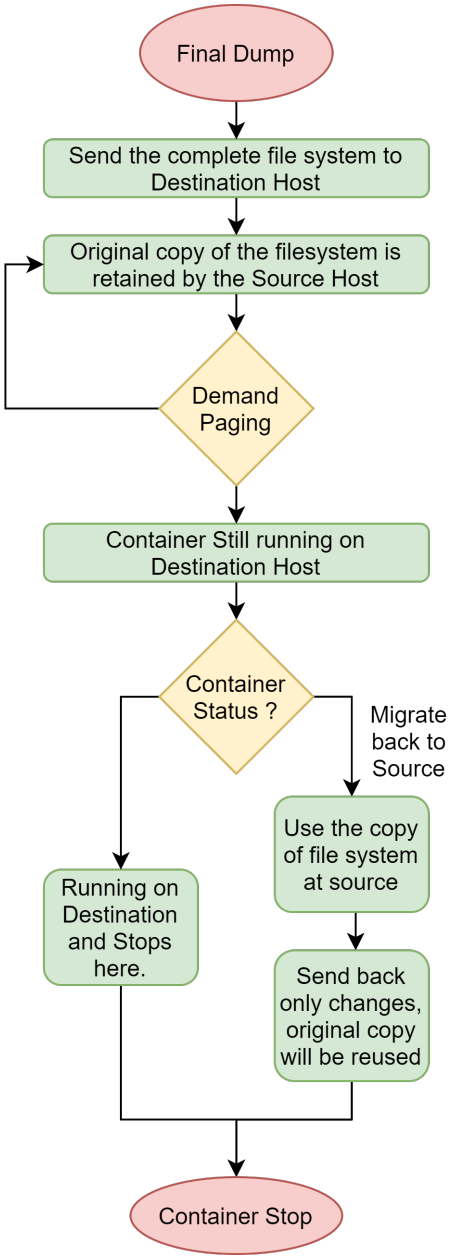

Figure 5.5: The detailed process of proposed methodology

In the proposed technique, when it is decided to migrate back to the same host, we will not send back all the memory pages on the destination host. As you can see in Figure 5.4(b), the pages indicated in red are those that the destination host has modified. Only the modified pages will be transmitted to the source host when migrating back.

5.4 System Model

The proposed migration model is working in two different phases. Phase 1 is the migration of containers from source to destination using the predictive pre-copy approach.

LSTM predicts the set of pages to be migrated in an iterative dump in this approach. Memory migration is predicted using the LSTM in a network model.

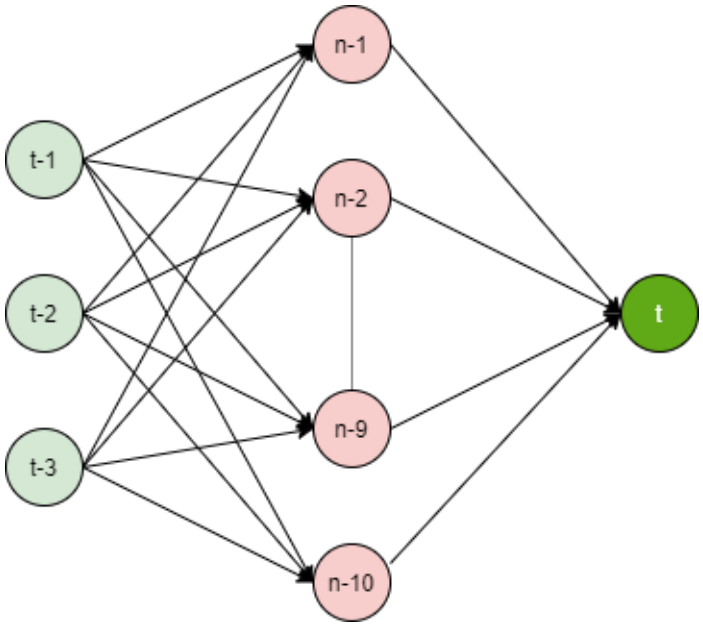

Figure 5.6: The ANN prediction model architecture used in proposed system model

An ANN is used to create the prediction model's architecture. This model utilizes three input layers that interact with a hidden layer of ten cells (LSTM cells) to produce a single output layer, as shown in Figure5.6. Where t_1, t_2, t_3 are representing the input layers, $n_1, n_2 ... n_3$ are denoting the LSTM cells and t in the final single layer output. Each LSTM cell used in Figure5.6 as $n_1, n_2 ... n_3$ is represented as show in Figure5.7. The LSTM module comprises three gates: forget gate, input gate, and output gate. The quantity of data that can pass via these gates is limited. A sigmoid function and an operation are used in each gate. The dotted line handles the data generated by these gates. The union of $h(t-1)$ and $x(t)$ determines the value of the Sigmoid function, which is between 0 and 1.

The output is obtained by multiplying the sigmoid result by the gate's input. For example, if the sigmoid result is 1, the gate output will be the same as the input since it is multiplied by one. The unit processes the input data for each input vector to the LSTM network as follows:

1. A new vector will be created by adding the $h(t-1)$ (hidden state vector) and $x(t)$ (input vector). The newly created vector will be input to the $tanh$ function and the three gates.

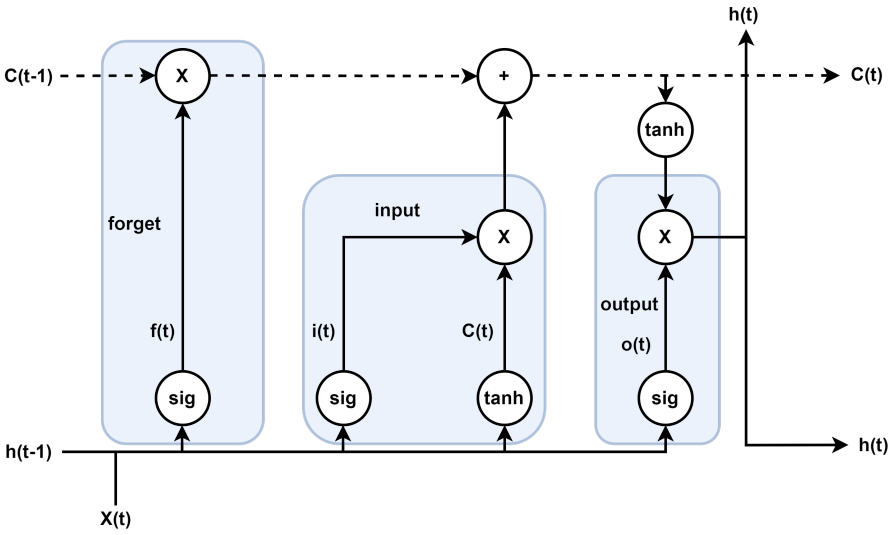

Figure 5.7: LSTM cell architecture

2. The flow of previously stored cell states regulated by the forget gate:
   $f(t) = sig(Wf * [h(t-1), x(t)] + bf)$

3. The C(t) candidate value for the present cell state is calculated as follows:
   $C(t) = tahn(W_c * [h(t-1), x(t)] + b_c))$

4. The amount of $C(t)$ to be added to current cell is fixed by input gate and then $C(t)$ multiplies with $i(t)$.
   $i(t) = sig(W_i * [h(t-1), x(t)] + b_i))$

5. The final calculated C(t) is as:
   $C(t) = f(t) * C(t-1) + i(t) * C(t)$

6. The output gate determines how much C(t) is passed to the next cell. The hidden state h(t) is calculated as follows:
   $O(t) = sig(W_O * [h(t-1), x(t)] + b_0))$
   $h(t) = O(t) * tanh(C(t))$

5.5 Experimental setup

Using the time series observation of the prior three iterations, this network design can anticipate the next batch of pages that will be transferred. There are two possible circumstances in which this model can be used. The first approach is called single-step

prediction, and it makes a single forecast based on several time-series inputs. Direct and recursive prediction are the other two methods of multi-step prediction.

Table 5.1: LSTM model configuration

Parameters	Range
Input size	3
Output size	1
Number of LSTM Layers	1
Number of LSTM Units	1
Kernel initializer	Lecun uniform
Loss function	MSE
Optimizer	adam
Batch size	64
Number of epochs	10

The mean square error is the loss function used to train LSTM and ANN prediction models. This prediction model is trained using a multi-step direct technique to forecast the number of active pages moved in the subsequent round. The model was trained using a Tensor Processing Unit (TPU) supplied by colab.

Algorithm 5 To identify modified pages in pre-copy container migration process

Result: Set of Modified Pages

d_b, Pg_i, r, R_{max}

while $r \leq R_{max}$ **do**
    while (mem_{pool}) **do**
        if $Pg_i.db\ TRUE$ **then**
            $Mod_{pool}.append(pg_i)$
        end
    end
    $r+=1$
end

return $(Mod_{pool}[Pg_i])$

The parameters used in this Algorithm 5 are as: P_{id} is representing pages of memory pool with page id, R_{max} is the maximum number of iterations and d_b is a Boolean data member to store dirty bit status, r is set to 1 and it is used for rounds/iterations. According to the selected memory pool $1 \leq r \leq R_{max}$ and $R_{max} = 10$. Algorithm **??** will return the set of modified memory pages.

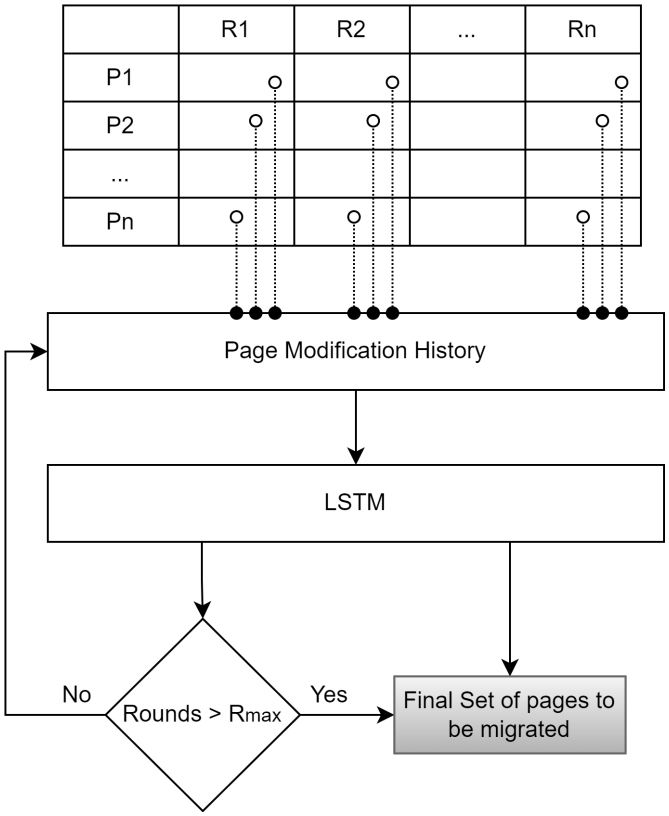

Figure 5.8: System architecture to identify updated pages using LSTM

The cell structure of LSTM cells is used in the ANN network. Now we will integrate this method with the proposed prediction system. As shown in Figure5.8, the array represents the memory pages in each round. Where P_1,P_2,P_n are representing the memory pages and R_1,R_2,R_n are denoting number of iterations. The updated status of each page will be stored in "Page Modification History". This will be provided as input to the LSTM module.

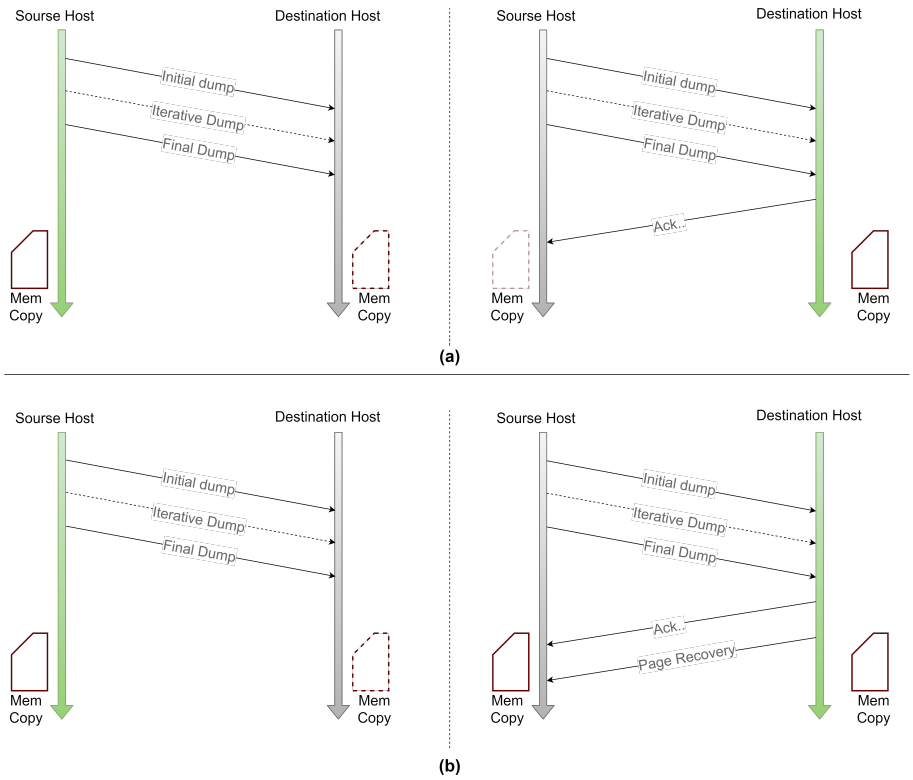

Figure 5.9: The process of transferring the container memory, where a). depicts the existing approach, in which the container's memory is deleted from the source host when it is migrated to the destination host and b). is a proposed approach for providing page recovery by keeping memory on the source host following a successful migration.

In this module, the cells mentioned in the ANN network are used to predict the updated pages. This process will be repeated upto the maximum iterations. This module will produce the final set of memory pages to be migrated to the destination host.

When a container is migrated from any source host to a destination, all its memory and system configuration is sent to the destination host. Everything related to that container is available on the destination host only. Once the receiving host acknowledges it, then the memory dump is removed from the source host as shown in Figure 5.9(a). In this standard pre-copy technique, there is no option to recover the pages from the source host.

The proposed technique provides the page recovery and reuses the memory while migrating back to the same host. The LSTM is used to migrate from source to destination only. When it is decided to migrate back to the same host, then we will migrate back only the modified pages, and the rest of the pages will be recovered from the copy

of dump at the host as shown in Figure 5.9(b). The set of pages identical to the pages at the source host will be discarded. This complete scenario is implemented with container cloudsim 4.0 and LSTM with dl2j used to predict memory pages to be migrated.

5.6 Performance and Evaluation

Migration performance and cost modeling is an essential component of migration management to assess and forecast migration requests' overall cost and performance. Single migration performance indicators have been the subject of several studies. We categorize these indicators according to time and data quantity. Data transmission size is the critical element for determining the network overhead associated with network migration. It is substantially positively associated with the migration time for pre-copy migration. The overall quantity of data transmitted equals the sum of the amounts transferred on each occasion. It consists of two components: memory data and storage data.

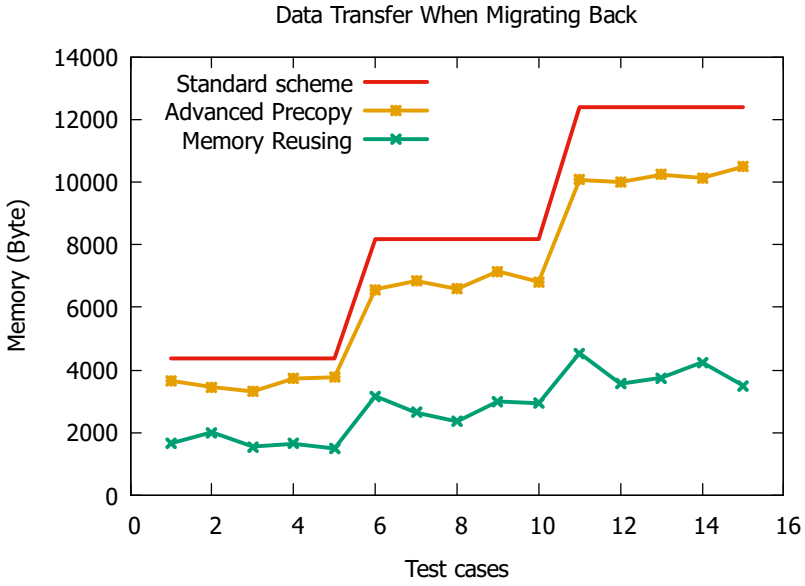

Figure 5.10: The amount of data transferred when migrating back to the same host with 15 test cases and test cases 1 to 5 implemented with 5 containers, 6 to 10 with 10 containers and 11 to 15 with 15 containers by using the standard pre-copy, advanced pre-copy and the proposed technique which revert only updated pages.

The proposed memory reusing approach is tested on different set of containers for better understanding and outcomes. As mentioned in Figure 5.10, the number of bytes

transferred during the migration of containers is specified. There are 15 test cases, where in the first 5 test cases, the set of 5 containers is implemented. The memory transfer with the existing approach [55] is represented in red color.

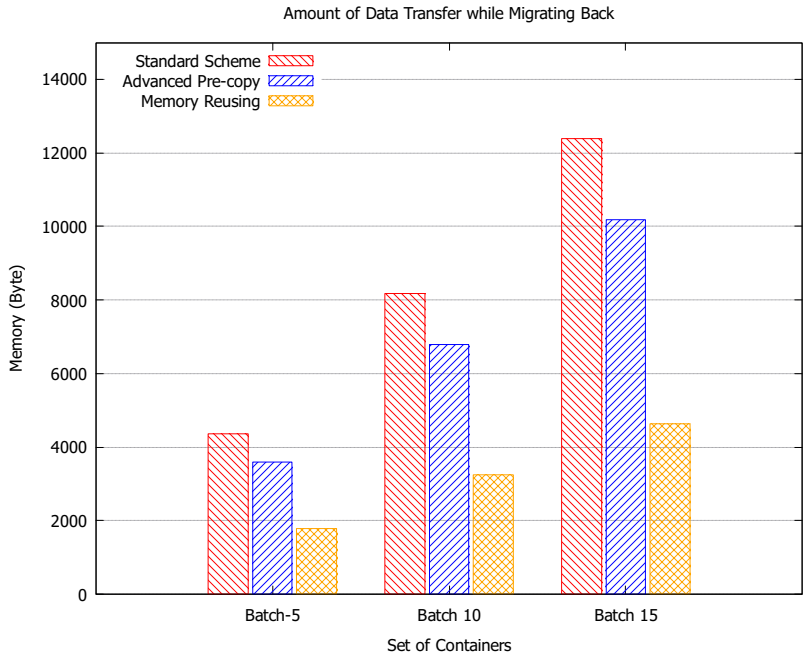

Figure 5.11: The amount of data transferred with the batch of 5 containers, 10 containers and 15 containers by using the standard pre-copy, advanced pre-copy and the proposed technique.

You can see the difference in data transferred in the proposed technique as illustrated in red-green color. The yellow color represents the migration from source to destination with the proposed scheme of our previous research. And according to the proposed approach, when it is migrating back to the same host, we are reverting only the updated pages.

The memory dump copies of the containers migrated to the destination host are stored at source host. Because in some of the cases when containers are migrated due to load balancing, fault tolerance, system upgrade, etc., the container migrates back to the same host. As we mentioned in the process of memory reusing in Section 5.3 when the container is migrated back to the same host, then we send only the modified memory pages, and these copies of memory are dumped at the source will be used to initiate the containers on the source host. In the case of the proposed technique, there is an additional space overhead on the source host. This additional space overhead will

occupy memory from the host system only. But with this small overhead, we can reduce the costly data transmission over the network. The result shows that the data transfer during migrating back is reduced.

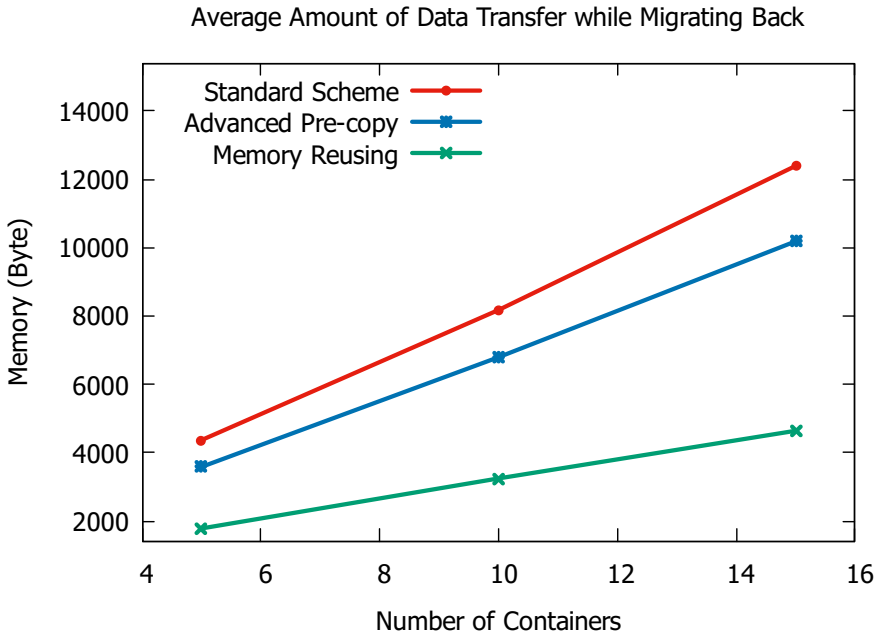

Figure 5.12: The average rate of data transferred when migrating back to the same host with the batch size of 5 containers, 10 containers and 15 containers by using the standard pre-copy, advanced pre-copy and the proposed technique.

Results show that if the container number increases in a batch of migration, the percentage of data transmission over the network improves. In the same way, we have run containers in three batches as a set of 5,10, and 15 containers. Further, in Figure 5.11, the amount of data transferred during the existing approach of container migration and the proposed technique are shown. Here we can identify the difference in the transmission.

The average rate of data transferred when migrating back to the same host with the batch size of 5 containers, 10 containers and 15 containers is specified in Figure 5.12. The rate of data transfer in the standard pre-copy approach [55], advanced pre-copy and the proposed pre-copy approach are discussed in detail. The number of bytes transferred in the proposed technique is much lesser than the existing techniques. It shows the average rate of reduction in the data transfer over the network by 60.68% compared to standard pre-copy and 52.30% compared to advanced pre-copy.

5.7 Summary

The migration process is divided into three phases in the pre-copy container migration technique. The first two phases are pre-dump and iterative dump. The pre-dump was implemented with the PSO algorithm and the iterative dump is implemented with LSTM. After that in the final dump, the memory related to the container will be removed from the source host. We recognized a few cases (fault tolerance, system upgrade, load balancing, etc.), where containers go back to the same host. In such cases, we have implemented the proposed migration technique that helps to reduce the data transfer over the network and it outperforms compared to the existing system. As a future direction, this technique can be implemented in various cloud environments like VMs used for various services, fog, edge computing, etc., where the instances are moving rapidly. The other alternative is to use the centralized instance image to reduce data transfer between source and destination host.

CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

This chapter concludes the thesis, including a summary of work and significant contributions. It then suggests and analyses several potential future research topics for improving container migration paradigms.

6.1 Summary of Contributions

Chapter 1: An introduction to related topics has been explored in detail in chapter 1 of this thesis. Containers, including how they vary from VMs, types of containers, container migration, and numerous container migration methodologies, are all thoroughly covered.

Chapter 2: The key component of pre-copy container migration is memory change prediction. It should be carefully chosen based on pre-copy migration's pre-dump, iterative, and final dump phases. To improve the prediction mechanism, a prediction scheme or collection of multiple approaches should be applied to the memory pages to be migrated to the destination host. PSO-based prediction schemes, LSTM-based prediction schemes, and ANN-based prediction schemes are all common options. Because of its prominence in migration procedures, it has been discovered that pre-copy has to be properly described. Pre-migration guarantees that resources are available between

the source and destination. Source hosts transmit information such as memory size and running application data with the destination host to select a suitable destination host and confirm that. To begin the migration for resource reservation, first check that the target host has enough resources and recognize the source host. To decrease the size of the memory dump during the pre-dump phase, a probabilistic PSO migration strategy is developed and implemented. Furthermore, the source host initiates the iterative copying of memory pages in iterative pre-copy. The next iteration sends all of the memory pages that were changed in the previous iteration. To limit the quantity of data transmitted, a predictive checkpoint for the iterative dump has been proposed. Finally, the "Stop-and-Copy" container will shut down and copy the remaining dirty index memory pages to their new location. The container migrates back to the same host in various instances, such as failure tolerance, load balancing, etc. A memory reuse technique is provided to reduce the total migration size, which also supports page faults.

Chapter 3: The rise of containers in cloud computing has shifted industry trends away from heavy virtualization and lightweight virtualization. There are several updated variants of container migration algorithms. Nonetheless, the migration step follows the same basic procedure as the pre-dump phase. The destination host receives all of the memory pages associated with the container. The reduction of data transmission during this period is a primary goal. We can reduce the size of the pre-dump. If we migrate a group of 5 containers to the destination host using the standard method, we will send 4367 B. However, using the proposed method, we will send about 3206.7B. Similarly, it was 8176B in the ten containers, but it is now 6079.4B, and it was 12387B in the fifteen containers, but it is now just 9040.5B.

Chapter 4: Virtual machines are more efficient than native servers for application deployment, and moving to containers can provide even more value to the IIoT ecosystem. Containers are becoming increasingly popular in IT businesses due to rising demand. Container migration has been deployed in several batches of containers to improve performance. These batches are divided into five, ten, and fifteen containers. This chapter shows how we can reduce data transfer across the network. The recommended method has been implemented as a pre-copy migration. The suggested technique exceeds previous alternatives in terms of the volume of data moved and overall migration time. When it comes to transferring the dump, the recommended method is faster than cold

and pre-copy migration. However, post-copy and a hybrid technique require less time than the suggested method. In terms of downtime, the cold and pre-copy procedures have more than the recommended, while the post-copy and hybrid options have less. Because the majority of data is sent after the container is started on the destination host in both post-copy and hybrid scenarios. In terms of "total migration time" and "amount of data moved," the suggested approach outperforms, but post-copy excels in the remaining two criteria, downtime and dump-time.

Chapter 5: The pre-copy container migration approach divides the migration procedure into three steps. Pre-dump and iterative dump are the first two processes that have previously been addressed in our past study. The PSO algorithm was used to create the pre-dump phase, while the LSTM method was used to perform the iterative dump phase. The memory associated with the container will then be deleted from the source host in the final dump. We identified a few instances where the container returns to the same host (fault tolerance, system upgrade, load balancing, and so on). In such cases, we've applied the recommended migration approach, which helps to decrease network data transmission. They can be improved even further by determining whether or not the container will be moved back. As a result, the migration strategy will be determined.

Multiple migration approaches to minimize dump size in various phases of pre-copy container migration are shown in the chapters above, which is a relevant contribution to the state-of-the-art.

6.2 Future Research Directions

This thesis addressed several memory-related concerns in the container migration paradigm. However, container migration approaches can be enhanced much further by addressing a few critical difficulties that need to be investigated further.

6.2.1 Container applications

Container success in the deployment of cloud applications stems from their loosely connected architecture, modularity, and flexibility. To address the obstacles, containers placement approaches should consider the context of applications and data before migrating them. Containers loosely connected and lightweight architecture also facilitates

the transfer of containerized applications. As a result, different migration models (e.g., pre-copy, post-copy, hybrid) should be thoroughly investigated in real cloud environments for a smooth migration, considering the type and structure of cloud applications.

6.2.2 Pre-dump

The pre-copy container migration is divided into three phases, as discussed in chapter 3. The size of the pre-dump phase was reduced by using probability-based PSO, which resulted in shorter migration time, less data transmission over the network, and reduced resource use. The suggested pre-dump approach surpasses the usual pre-dump migration technique of container migration, with a 26.48 percent reduction in pre-dump size. This strategy can be utilized in the future to reduce the overall dump size during container migration in cloud and edge computing.

6.2.3 Iterative dump

Because of the increasing demand for containers, the amount of data transfer is a crucial factor for the overall performance. In future research, the memory prediction in live container migration can be evaluated at a large scale in real scenarios to obtain detailed insights. The type of application running in the container may affect the prediction results. We plan to extend our study to different applications and test these with alternative prediction schemes, such as ANN, ARIMA, etc., to determine future directions.

6.2.4 Final dump

The final dump is the last phase of pre-copy container migration and is discussed in chapter 5. The complete dump is transferred to the destination host in the standard approach. The data transmission in the final dump is dependent on the memory migrated in the previous two phases. With the proper synchronization in these phases, the final dump size can also be reduced.

6.2.5 Recovery Technique

The successful migration of containers and resumes to the destination host is an essential factor affecting the migration. There are chances that the container resume fails on the destination host due to data loss during data transmission. Some effective methods

to handle page faults can be discussed further. The online shared dump is one of the options to address the issue.

6.2.6 Memory Reusing

As stated in Chapter 5, the container will most likely migrate back to its original host in some instances. We can minimize the amount of data transmission in such cases by recycling the dump from the source host. Pages that are identical to the source material will not be returned. It would be a huge benefit if it could be expanded further in a cloud setting.

6.3 Final Remarks

The cloud computing paradigm has evolved into the backbone of today's digital world, allowing numerous solutions to be implemented for various use cases, including computing clouds, data centers, science, and entertainment, to name a few. To fully realize the potential of cloud computing, effective container migration is a critical challenge, influencing container application execution costs, user QoE, and operational expenses.

This thesis examined how to migrate containers efficiently for the smooth execution of processes on resource-constrained systems. The techniques and system designs suggested in this thesis reduce the time it takes to migrate containerized processes, the amount of data transferred, and downtime while also improving user QoE. The limitation of the system is the extra overhead on the host system. The cause of this extra overhead is the time and resources taken by the probability calculation. Still, this overhead is comparatively less than the overhead over the network. Container migration research, such as those provided in this thesis, will allow cloud service providers to migrate containers in cloud computing environments at scale successfully and efficiently. Furthermore, the findings of this research can help promote container migration innovations and advances.

REFERENCES

[1] R. Synytsky. Containers live migration: Behind the scenes. [Online]. Available: https://www.infoq.com/articles/container-live-migration/

[2] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, "Container migration in the fog: A performance evaluation," *Sensors*, vol. 19, no. 7, p. 1488, 2019.

[3] J. Tang, Y. Li, M. Ding, H. Liu, D. Yang, and X. Wu, "An ionospheric tec forecasting model based on a cnn-lstm-attention mechanism neural network," *Remote Sensing*, vol. 14, no. 10, 2022. [Online]. Available: https://www.mdpi.com/2072-4292/14/10/2433

[4] P.-P. Phyo and C. Jeenanunta, "Advanced ml-based ensemble and deep learning models for short-term load forecasting: Comparative analysis using feature engineering," *Applied Sciences*, vol. 12, no. 10, p. 4882, 2022.

[5] A. F. Leite, A. Boukerche, A. C. Magalhaes Alves de Melo, C. Eisenbeis, C. Tadonki, and C. G. Ralha, "Power-aware server consolidation for federated clouds," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 12, pp. 3427–3444, 2016.

[6] D. Merkel *et al.*, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[7] Q. Huang, L. Sun, F. Jia, J. Yuan, Y. Wu, and J. Pan, "Automatic scaling mechanism of intermodal edi system under green cloud computing," *Journal of Advanced Transportation*, vol. 2022, 2022.

[8] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, "A comparative study of containers and virtual machines in big data environment," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 178–185.

[9] J. Mellado and F. Núñez, "Design of an iot-plc: A containerized programmable logical controller for the industry 4.0," *Journal of Industrial Information Integration*, vol. 25, p. 100250, 2022.

[10] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete container state migration," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2137–2142.

[11] H. Korala, D. Georgakopoulos, P. P. Jayaraman, and A. Yavari, "A survey of techniques for fulfilling the time-bound requirements of time-sensitive iot applications," *ACM Computing Surveys*, 2022.

[12] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Autonomic vertical elasticity of docker containers with elasticdocker," in *2017 IEEE 10th international conference on cloud computing (CLOUD)*. IEEE, 2017, pp. 472–479.

[13] P. Chaurasia, S. B. Nath, S. K. Addya, and S. K. Ghosh, "Automating the selection of container orchestrators for service deployment," in *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2022, pp. 739–743.

[14] C. Prakash, D. Mishra, P. Kulkarni, and U. Bellur, "Portkey: hypervisor-assisted container migration in nested cloud environments," in *Proceedings of the 18th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2022, pp. 3–17.

[15] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, and F. Longo, "Design and evaluation of a fog platform supporting device mobility through container migration," *Pervasive and Mobile Computing*, p. 101415, 2021.

[16] S. Hu, W. Shi, and G. Li, "Cec: A containerized edge computing framework for dynamic resource provisioning," *IEEE Transactions on Mobile Computing*, 2022.

[17] S.-H. Choi and K.-W. Park, "Icontainer: Consecutive checkpoint with a rapid resilience for immortal container-based services," *Available at SSRN 4034464*.

[18] M. P. Yadav, H. A. Akarte, and D. K. Yadav, "Container elasticity: Based on response time using docker," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, vol. 15, no. 5, pp. 773–785, 2022.

[19] A. K. Gupta, A. Sharma, A. Pandey, P. Kaustubh, and S. Sharma, "Live migration in hpc," in *Cybersecurity and High-Performance Computing Environments*. Chapman and Hall/CRC, 2022, pp. 191–227.

[20] C. Pu, H. Xu, H. Jiang, D. Chen, and P. Han, "An environment-aware and dynamic compression-based approach for edge computing service migration," in *2022 2nd International Conference on Consumer Electronics and Computer Engineering (ICCECE)*. IEEE, 2022, pp. 292–297.

[21] C. Puliafito, L. Conforti, A. Virdis, and E. Mingozzi, "Server-side quic connection migration to support microservice deployment at the edge," *Pervasive and Mobile Computing*, vol. 83, p. 101580, 2022.

[22] B. Shi, H. Shen, B. Dong, and Q. Zheng, "Memory/disk operation aware lightweight vm live migration," *IEEE/ACM Transactions on Networking*, 2022.

[23] A. K. Dash, "Understanding migration mechanisms of containers using criu," 2022.

[24] R. Hemalatha, D. Akila, D. Balaganesh, and A. Paul, *The Internet of Medical Things (IoMT): Healthcare Transformation*. John Wiley & Sons, 2022.

[25] Y. S. Abdulsalam and M. Hedabou, "Decentralized data integrity scheme for preserving privacy in cloud computing," in *2021 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*. IEEE, 2021, pp. 607–612.

[26] A. Bentajer and M. Hedabou, "Cryptographic key management issues in cloud computing," *Adv. Eng. Res*, vol. 34, pp. 78–112, 2020.

[27] H. Nie, P. Li, H. Xu, L. Dong, J. Song, and R. Wang, "Research on optimized pre-copy algorithm of live container migration in cloud environment," in *International Symposium on Parallel Architecture, Algorithm and Programming*. Springer, 2017, pp. 554–565.

[28] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, "Containers checkpointing and live migration," in *Proceedings of the Linux Symposium*, vol. 2, 2018, pp. 85–90.

[29] B. R. Raghunath and B. Annappa, "Prediction based dynamic resource provisioning in virtualized environments," in *2017 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2017, pp. 100–105.

[30] K. Elghamrawy, D. Franklin, and F. T. Chong, "Predicting memory page stability and its application to memory deduplication and live migration," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2017, pp. 125–126.

[31] Y. Qiu, C.-H. Lung, S. Ajila, and P. Srivastava, "Experimental evaluation of lxc container migration for cloudlets using multipath tcp," *Computer Networks*, vol. 164, p. 106900, 2019.

[32] D. Kim, H. Muhammad, E. Kim, S. Helal, and C. Lee, "Tosca-based and federation-aware cloud orchestration for kubernetes container platform," *Applied Sciences*, vol. 9, no. 1, p. 191, 2019.

[33] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

[34] G. Moltó, M. Caballer, A. Pérez, C. de Alfonso, and I. Blanquer, "Coherent application delivery on hybrid distributed computing infrastructures of virtual

machines and docker containers," in *2017 25Th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2017, pp. 486–490.

[35] C. Dupont, R. Giaffreda, and L. Capra, "Edge computing in iot context: Horizontal and vertical linux container migration," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, 2017, pp. 1–4.

[36] T. Wu, N. Guizani, and J. Huang, "Related dirty memory prediction mechanism for live migration enhancement in cloud computing environments," *Journal of Network and Computer Applications*, vol. 3, pp. 1–14, 2017.

[37] X. Guan, X. Wan, B.-Y. Choi, S. Song, and J. Zhu, "Application oriented dynamic resource allocation for data centers using docker containers," *IEEE Communications Letters*, vol. 21, no. 3, pp. 504–507, 2016.

[38] C. Kan, "Docloud: An elastic cloud platform for web applications based on docker," in *2016 18th international conference on advanced communication technology (ICACT)*. IEEE, 2016, pp. 478–483.

[39] C. Yu and F. Huan, "Live migration of docker containers through logging and replay," in *2015 3rd International Conference on Mechatronics and Industrial Informatics (ICMII 2015)*. Atlantis Press, 2015.

[40] P. Jain and R. Agrawal, "An improved pre-copy approach for transferring the vm data during the virtual machine migration for the cloud environment," *International journal of engineering and manufacturing*, vol. 6, no. 6, pp. 51–60, 2016.

[41] Y. Fang, Y. Chen, and J. Ge, "Improvement of live migration mechanism for virtual machine based on pre-copy," in *2016 3rd International Conference on Materials Engineering, Manufacturing Technology and Control*. Atlantis Press, 2016.

[42] E. Mostajeran, M. F. Khalid, M. N. M. Mydin, B. I. Ismail, and H. Ong, "Multifaceted trust assessment framework for container based edge computing platform," in *Fifth InternationalConference On Advances in Computing, Control and Networking-ACCN 2016*, 2016.

[43] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys european conference on computer systems 2007*, 2007, pp. 275–287.

[44] S. Luo, G. Zhang, C. Wu, S. Khan, and K. Li, "Boafft: distributed deduplication for big data storage in the cloud," *IEEE transactions on cloud computing*, 2015.

[45] M. Ansar, M. W. Ashraf, M. Fatima *et al.*, "Data migration in cloud: A systematic review," *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, vol. 48, no. 1, pp. 73–89, 2018.

[46] M. Patel, S. Chaudhary, and S. Garg, "vmeasure: Performance modeling for live vm migration measuring," in *Advances in Data and Information Sciences*. Springer, 2019, pp. 185–195.

[47] M. Patel and S. Chaudhary, "Improved pre-copy algorithm using statistical prediction and compression model for efficient live memory migration," *International Journal of High Performance Computing and Networking*, vol. 11, pp. 55–65, 2018.

[48] S. Sharma and M. Chawla, "A three phase optimization method for precopy based vm live migration," *SpringerPlus*, vol. 5, no. 1, p. 1022, 2016.

[49] R. de Jesus Martins, C. B. Both, J. A. Wickboldt, and L. Z. Granville, "Virtual network functions migration cost: from identification to prediction," *Computer Networks*, vol. 181, p. 107429, 2020.

[50] X.-Z. Xie, C.-C. Chang, and K. Chen, "A high-embedding efficiency rdh in encrypted image combining msb prediction and matrix encoding for non-volatile memory-based cloud service," *IEEE Access*, vol. 8, pp. 52 028–52 040, 2020.

[51] J. Kumar and A. K. Singh, "Decomposition based cloud resource demand prediction using extreme learning machines," *Journal of Network and Systems Management*, vol. 28, no. 4, pp. 1775–1793, 2020.

[52] X. Wang, C. Xu, K. Wang, F. Yan, and D. Zhao, "Memory scaling of cloud-based big data systems: A hybrid approach," *IEEE Transactions on Big Data*, 2020.

[53] M. Imdoukh, I. Ahmad, and M. G. Alfailakawi, "Machine learning-based auto-scaling for containerized applications," *Neural Computing and Applications*, vol. 32, no. 13, pp. 9745–9760, 2020.

[54] M. Masdari and H. Khezri, "Efficient vm migrations using forecasting techniques in cloud computing: a comprehensive review," *Cluster Computing*, pp. 1–30, 2020.

[55] A. Bhardwaj and C. R. Krishna, "A container-based technique to improve virtual machine migration in cloud computing," *IETE Journal of Research*, pp. 1–16, 2019.

[56] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, "Containers checkpointing and live migration," in *Proceedings of the Linux Symposium*, vol. 2, 2018, pp. 85–90.

[57] S. K. Chronopoulos, E. I. Kosma, K. P. Peppas, D. Tafiadis, K. Drosos, N. Ziavra, and E. I. Toki, "Exploring the speech language therapy through information communication technologies, machine learning and neural networks," in *2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2021, pp. 193–198.

[58] A. Slominski, V. Muthusamy, and R. Khalaf, "Building a multi-tenant cloud service from legacy code with docker containers," in *2015 IEEE International Conference on Cloud Engineering*. IEEE, 2015, pp. 394–396.

[59] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *Journal of Grid Computing*, vol. 16, no. 1, pp. 113–135, 2018.

[60] Y. Du, H. Yu, G. Shi, J. Chen, and W. Zheng, "Microwiper: efficient memory propagation in live migration of virtual machines," in *2010 39th International Conference on Parallel Processing*. IEEE, 2010, pp. 141–149.

[61] Y. Zhong, J. Xu, Q. Li, H. Zhang, and F. Liu, "Memory state transfer optimization for pre-copy based live vm migration," in *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*. IEEE, 2014, pp. 290–293.

[62] B. Das, K. K. Mandal, and S. Das, "Improving total migration time in live virtual machine migration," in *Proceedings of the Sixth International Conference on Computer and Communication Technology 2015*, 2015, pp. 57–61.

[63] W. Zhang, L. Chen, J. Luo, and J. Liu, "A two-stage container management in the cloud for optimizing the load balancing and migration cost," *Future Generation Computer Systems*, 2022.

[64] W. Moussa, M. Nashaat, W. Saber, and R. Rizk, "Comprehensive study on machine learning-based container scheduling in cloud," in *International Conference on Advanced Machine Learning Technologies and Applications*. Springer, 2022, pp. 581–592.

[65] M. Gundall, J. Stegmann, M. Reichardt, and H. D. Schotten, "Downtime optimized live migration of industrial real-time control services," *arXiv preprint arXiv:2203.12935*, 2022.

[66] M. Terneborg, "Enabling container failover by extending current container migration techniques," 2021.

[67] M. Terneborg, J. K. Rönnberg, and O. Schelén, "Application agnostic container migration and failover," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE, 2021, pp. 565–572.

[68] Z. Zhi, Z. Zhuofeng, and L. Han, "Static layout and dynamic migration method of a large-scale container," in *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 5. IEEE, 2021, pp. 1897–1901.

[69] S. Zheng, F. Huang, C. Li, and H. Wang, "A cloud resource prediction and migration method for container scheduling," in *2021 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS)*. IEEE, 2021, pp. 76–80.

[70] R. Yang, H. He, and W. Zhang, "Multitier service migration framework based on mobility prediction in mobile edge computing," *Wireless Communications and Mobile Computing*, vol. 2021, 2021.

[71] L. Chen and W. Zhang, "A deep learning-based approach with pso for workload prediction of containers in the cloud," in *2021 13th International Conference on Advanced Infocomm Technology (ICAIT)*. IEEE, 2021, pp. 204–208.

[72] D. Dai Vu, X. T. Vu, and Y. Kim, "Deep learning-based fault prediction in cloud system," in *2021 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2021, pp. 1826–1829.

[73] S. Pervaiz, Z. Ul-Qayyum, W. H. Bangyal, L. Gao, and J. Ahmad, "A systematic literature review on particle swarm optimization techniques for medical diseases detection," *Computational and Mathematical Methods in Medicine*, vol. 2021, 2021.

[74] W. H. Bangyal, A. Hameed, W. Alosaimi, and H. Alyami, "A new initialization approach in particle swarm optimization for global optimization problems," *Computational Intelligence and Neuroscience*, vol. 2021, 2021.

[75] W. H. Bangyal, K. Nisar, A. Ibrahim, A. A. Bin, M. R. Haque, J. J. Rodrigues, and D. B. Rawat, "Comparative analysis of low discrepancy sequence-based initialization approaches using population-based algorithms for solving the global optimization problems," *Applied Sciences*, vol. 11, no. 16, p. 7591, 2021.

[76] G. Singh, P. Singh, M. Hedabou, M. Masud, and S. S. Alshamrani, "A predictive checkpoint technique for iterative phase of container migration," *Sustainability*, vol. 14, no. 11, p. 6538, 2022.

[77] R. Stoyanov and M. J. Kollingbaum, "Efficient live migration of linux containers," in *International Conference on High Performance Computing*. Springer, 2018, pp. 184–193.

[78] A. Widjajarto, D. W. Jacob, and M. Lubis, "Live migration using checkpoint and restore in userspace (criu): Usage analysis of network, memory and cpu," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 2, pp. 837–847, 2021.

[79] S. Oh and J. Kim, "Stateful container migration employing checkpoint-based restoration for orchestrated container clusters," in *2018 International Conference*

on Information and Communication Technology Convergence (ICTC). IEEE, 2018, pp. 25–30.

[80] R. Torre, E. Urbano, H. Salah, G. T. Nguyen, and F. H. Fitzek, "Towards a better understanding of live migration performance with docker containers," in *European Wireless 2019; 25th European Wireless Conference*. VDE, 2019, pp. 1–6.

[81] Y. Al-Dhuraibi, F. Zalila, N. Djarallah, and P. Merle, "Coordinating vertical elasticity of both containers and virtual machines," 2018.

[82] G. Singh and P. Singh, "A taxonomy and survey on container migration techniques in cloud computing," in *Sustainable Development Through Engineering Innovations: Select Proceedings of SDEI 2020*. Springer Singapore, 2021, pp. 419–429.

[83] M. A. Haghighi, M. Maeen, and M. Haghparast, "An energy-efficient dynamic resource management approach based on clustering and meta-heuristic algorithms in cloud computing iaas platforms," *Wireless Personal Communications*, vol. 104, no. 4, pp. 1367–1391, 2019.

[84] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "A pso model with vm migration and transmission power control for low service delay in the multiple cloudlets ecc scenario," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[85] M. S. Sheela and C. Arun, "Hybrid pso–svm algorithm for covid-19 screening and quantification," *International Journal of Information Technology*, pp. 1–8, 2022.

[86] P. Rawat and S. Chauhan, "Particle swarm optimization based sleep scheduling and clustering protocol in wireless sensor network," *Peer-to-Peer Networking and Applications*, vol. 15, no. 3, pp. 1417–1436, 2022.

[87] U. S. Bist and N. Singh, "A novel chaotic kernel framework for support vector machines using probability-based feature extraction method," in *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2022, pp. 885–890.

[88] Y. Liu and M. Zhao, "An obsolescence forecasting method based on improved radial basis function neural network," *Ain Shams Engineering Journal*, vol. 13, no. 6, p. 101775, 2022.

[89] X. Zhu, N. Li, and Y. Pan, "Optimization performance comparison of three different group intelligence algorithms on a svm for hyperspectral imagery classification," *Remote Sensing*, vol. 11, no. 6, p. 734, 2019.

[90] F. Wihartiko, H. Wijayanti, and F. Virgantari, "Performance comparison of genetic algorithms and particle swarm optimization for model integer programming bus timetabling problem," in *IOP Conference Series: Materials Science and Engineering*, vol. 332, no. 1. IOP Publishing, 2018, p. 012020.

[91] Y. Hu, H. Wang, Y. Zhang, and B. Wen, "Frequency prediction model combining isfr model and lstm network," *International Journal of Electrical Power & Energy Systems*, vol. 139, p. 108001, 2022.

[92] Y. Xu, C. Hu, Q. Wu, S. Jian, Z. Li, Y. Chen, G. Zhang, Z. Zhang, and S. Wang, "Research on particle swarm optimization in lstm neural networks for rainfall-runoff simulation," *Journal of Hydrology*, vol. 608, p. 127553, 2022.

[93] N. Elizabeth Michael, M. Mishra, S. Hasan, and A. Al-Durra, "Short-term solar power predicting model based on multi-step cnn stacked lstm technique," *Energies*, vol. 15, no. 6, p. 2150, 2022.

[94] L. Du, R. Gao, P. N. Suganthan, and D. Z. Wang, "Bayesian optimization based dynamic ensemble for time series forecasting," *Information Sciences*, vol. 591, pp. 155–175, 2022.

[95] M. Xu, J. Du, Z. Xue, Z. Guan, F. Kou, and L. Shi, "A scientific research topic trend prediction model based on multi-lstm and graph convolutional network," *International Journal of Intelligent Systems*, 2022.

[96] W. Jiang, "Internet traffic matrix prediction with convolutional lstm neural network," *Internet Technology Letters*, vol. 5, no. 2, p. e322, 2022.

[97] "Understanding lstm networks," https://colah.github.io/posts/2015-08-Understanding-LSTMs/, accessed: 2021-11-22.

[98] K. Cho and Y. Kim, "Improving streamflow prediction in the wrf-hydro model with lstm networks," *Journal of Hydrology*, vol. 605, p. 127297, 2022.

[99] J. Chen, X. Wang, and X. Xu, "Gc-lstm: Graph convolution embedded lstm for dynamic network link prediction," *Applied Intelligence*, vol. 52, no. 7, pp. 7513–7528, 2022.

[100] O. Osanaiye and S. Adeshina, "Service availability of virtual machines in cloud computing," in *Cloud and Fog Computing Platforms for Internet of Things*. Chapman and Hall/CRC, 2022, pp. 129–141.

[101] D. N. Nirmala and K. S. Vengatesh, "Research challenges in pre-copy virtual machine migration in cloud environment," *The Internet of Medical Things (IoMT) Healthcare Transformation*, pp. 45–72, 2022.

[102] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[103] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing—the business perspective," *Decision support systems*, vol. 51, no. 1, pp. 176–189, 2011.

[104] A. M. Joy, "Performance comparison between linux containers and virtual machines," in *2015 International Conference on Advances in Computer Engineering and Applications*. IEEE, 2015, pp. 342–346.

[105] Y. Mao, Y. Fu, S. Gu, S. Vhaduri, L. Cheng, and Q. Liu, "Resource management schemes for cloud-native platforms with computing containers of docker and kubernetes," *arXiv preprint arXiv:2010.10350*, 2020.

[106] K. Govindaraj and A. Artemenko, "Container live migration for latency critical industrial applications on edge computing," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 83–90.

[107] W. Liang, L. Cui, and F. P. Tso, "Low-latency service function chain migration in edge-core networks based on open jackson networks," *Journal of Systems Architecture*, p. 102405, 2022.

[108] CRIU. Live migration. [Online]. Available: https://criu.org/Live_migration

[109] V. Kulkarni, S. S. Aldi, M. M. Mulla, D. Narayan, and P. Hiremath, "Dynamic live vm migration mechanism in openstack-based cloud," in *2022 International Conference on Computer Communication and Informatics (ICCCI)*. IEEE, 2022, pp. 1–6.

[110] T. He, A. N. Toosi, and R. Buyya, "Performance evaluation of live virtual machine migration in sdn-enabled cloud data centers," *Journal of Parallel and Distributed Computing*, vol. 131, pp. 55–68, 2019.

[111] C. Sunil, K. Raghunandan, K. Ranjit, H. Chethan, and G. H. Kumar, "An innovative approach for cloud-based web dev app migration," in *ICT with Intelligent Applications*. Springer, 2022, pp. 807–817.

[112] P. Karhula, J. Janak, and H. Schulzrinne, "Checkpointing and migration of iot edge functions," in *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking*, 2019, pp. 60–65.

[113] J. Feng, J. Zhang, Y. Xiao, and Y. Ji, "Demonstration of containerized vdu/vcu migration in wdm metro optical networks," in *2020 Optical Fiber Communications Conference and Exhibition (OFC)*. IEEE, 2020, pp. 1–3.

[114] J. Schwarzrock, M. G. Jordan, G. Korol, C. C. de Oliveira, A. F. Lorenzon, M. B. Rutzig, and A. C. S. Beck, "Dynamic concurrency throttling on numa systems and data migration impacts," *Design Automation for Embedded Systems*, vol. 25, no. 2, pp. 135–160, 2021.

[115] R. S. Venkatesh, T. Smejkal, D. S. Milojicic, and A. Gavrilovska, "Fast in-memory criu for docker containers," in *Proceedings of the International Symposium on Memory Systems*, 2019, pp. 53–65.

[116] A. E. González and E. Arzuaga, "Herdmonitor: Monitoring live migrating containers in cloud environments," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 2180–2189.

[117] F. Hofer, M. Sehr, A. Sangiovanni-Vincentelli, and B. Russo, "Industrial control via application containers: Maintaining determinism in iaas," *Systems Engineering*, vol. 24, no. 5, pp. 352–368, 2021.

[118] H. Jin, B. Liu, W. Jiang, Y. Ma, X. Shi, B. He, and S. Zhao, "Layer-centric memory reuse and data migration for extreme-scale deep learning on many-core architectures," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 3, pp. 1–26, 2018.

[119] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Llc-guided data migration in hybrid memory systems," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 932–942.

[120] K. Kaur, F. Guillemin, and F. Sailhan, "Container placement and migration strategies for cloud, fog and edge data centers: A survey," 2022.

[121] R. M. Haris, K. M. Khan, and A. Nhlabatsi, "Live migration of virtual machine memory content in networked systems: A review," *Computer Networks*, p. 108898, 2022.

LIST OF PUBLICATIONS

– **Gursharan Singh,** Pooja Gupta, "A review on migration techniques and challenges in live virtual machine migration", *5th International Conference on Reliability, Infocom Technologies and Optimization* (Published), 2016. (Scopus)

– **Gursharan Singh,** Parminder Singh, "A Taxonomy and Survey on Container Migration Techniques in Cloud Computing", *Sustainable Development Through Engineering Innovations* (Published), 2021. (Scopus)

– **Gursharan Singh,** Parminder Singh, Babar Shah, Farman Ali and Daehan Kwak, "A Lightweight Migration Technique for Uninterrupted Health Care System in Cloud", *Computational Intelligence and Neuroscience* (Under Revision), 2022. (Scopus, SCIE 3.63 IF)

– **Gursharan Singh,** Parminder Singh, Mustapha Hedabou, Mehedi Masud and Sultan S. Alshamrani, "A Predictive Checkpoint Technique for Iterative Phase of Container Migration", *Sustainability* (Published), 2022. (Scopus, SCIE 3.25 IF)

– **Gursharan Singh,** Parminder Singh, Container Migration Technique to Minimize the Network Overhead with Reusable Memory State, *International Journal of Computer Networks and Applications (IJCNA)* (Published), 2022. (Scopus)