Enhancing Convergence of Live VM Migration with Dynamic Workloads

Nadeesha Nethmini Epa
Index number: 20000499

Supervisor: Dr. Dinuni Fernando
Co-Supervisor: Dr. Jerome Dinal Herath

April 2025

Submitted in partial fulfillment of the requirements of the
B.Sc in Computer Science Final Year Project (SCS4224)

Declaration

I certify that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, to be made available for photocopying and interlibrary loans, and for the title and abstract to be made available to outside organizations.

Student Name : D.N.N. Epa
Registration Number : 2020/CS/049
Index Number : 20000499

Signature & Date

This is to certify that this dissertation is based on the work of Ms.D.N.N.Epa under my supervision. The thesis has been prepared according to the format stipulated and is of an acceptable standard.

Supervisor Name : Dr. Dinuni K.Fernando

Signature & Date

Co-Supervisor Name : Dr. Jerome Dinal Herath

Signature & Date

Abstract

Live Virtual Machine (VM) migration is a critical process in cloud computing, enabling workload balancing, resource optimization, and fault tolerance. However, selecting the most suitable migration technique is challenging due to the dynamic nature of workloads. Traditional migration methods—pre-copy, post-copy, and hybrid approaches—often struggle with increased Total Migration Time (TMT), downtime, and performance degradation, especially under varying workload conditions.

This research proposes ProMig, a novel migration decision framework that leverages Markov models to predict future workload behavior and select the optimal migration technique while adapting to the dynamic nature of workloads. CPU usage, memory usage, and network usage are used as key resource indicators to define system states and anticipate workload transitions. The evaluation reveals that ProMig consistently minimizes TMT by adapting to both stable and fluctuating workload patterns. It has 87% accuracy for the total migration time reduction. Furthermore, certain workloads, such as those with high dirty page rates but low memory usage, present additional migration challenges. Future work will explore integrating dirty page rate as a resource metric and extending the model to multi-VM migration scenarios for enhanced scalability and efficiency.

Through intelligent prediction and adaptive decision-making, ProMig improves migration efficiency, reduces downtime, and ensures optimal resource utilization in cloud environments.

Acknowledgment

First and foremost, I would like to express my heartfelt gratitude to my supervisors, Dr. Dinuni Fernando and Dr. Dinal Herath, for their invaluable guidance, unwavering support, continuous encouragement, and motivation throughout this research project. Their insightful feedback and expertise were instrumental in shaping the direction of this project and ensuring its successful completion.

I am also deeply grateful to WSO2 for providing the servers and to Dr. Dinuni Fernando for facilitating their collaboration, without which this project would not have been possible.

My sincere appreciation goes to the staff at Network Operating Center (NOC) for their dedication in setting up the servers and granting us access to the server room. I would also like to acknowledge all the servers I used for this project for their reliability, despite being pushed to their limits.

Finally, I am profoundly thankful to my parents, siblings, and colleagues for their unwavering support, encouragement, and understanding throughout this challenging journey. Their presence and belief in me served as a constant source of motivation throughout the project.

Contents

List of Figures

List of Tables

Acronyms

CC Cloud Computing. 1, 6

CRM Cloud Resource Manager. 14

CUA Capacity and Utility Agent. 11

IaaS Infrastructure as a service. 11

MAP Markovian Arrival Process. 13

ML Machine Learning. 3, 11

OS Operating Systems. 8, 19

ProMig Predictive Live Migration Algorithm. ii, 20, 29, 32, 35, 42, 45, 48, 53–55

RTM Real Time Monitoring. 11

SaaS Software as a service. 11

VM Virtual Machine. ii, 1, 4–7, 14, 19, 23, 28, 49

VMs Virtual Machines. 1, 2, 14, 16

1 Introduction

1.1 Introduction

As the technology grows cloud computing(CC) has become essential for developing reliable and efficient applications. One of the foundational concepts in cloud computing is virtualization. Through virtualization, users can access and utilize multiple hardware resources from a single physical machine, maximizing the machine's capability and efficiency. As the name suggests, Virtualization (Xing & Zhan 2012, Malhotra et al. 2014) defines the process of generating a virtual representation of actual hardware, allowing users to place multiple hardware resources inside one physical machine. This virtual representation is a Virtual Machine (VM).

Containerization(Watada et al. 2019) is another key technology that enhances resource utilization in cloud computing. Unlike VMs, which include a full operating system (OS) and virtualized hardware, containers share the host OS kernel while isolating applications and their dependencies. This lightweight architecture makes containers faster to deploy and more efficient for managing microservices and stateless applications. However, for workloads that require strong isolation, dedicated resources, and compatibility with legacy systems, VMs remain the preferred choice.

In cloud data centers, a single server can host multiple VMs. However, a failure in that server would render all VMs on it unavailable, leading to significant service disruptions. VM migration has been introduced as a solution to this problem. It defines the idea of transferring a VM running on one server(source host) to another server (destination host). Since the failures of a server can happen at any time migration is essential for low-level system maintenance, load balancing, and fault management (Choudhary et al. 2017).

A crucial factor in VM migration is identifying the convergence point, the stage where migration can be completed without ongoing changes disrupting the process. Convergence ensures that the source and destination VMs are synchronized, enabling a seamless switchover with minimal downtime and no data inconsistencies. Failure to achieve proper convergence may lead to service interruptions, data loss, or degraded application performance.

During migration, certain memory pages may be modified repeatedly over multiple iterations. The point at which the same set of pages continues to change is identified as the convergence point. Determining this point allows for predicting migration duration after a triggering event. This prediction helps in planning and optimizing migration processes, ensuring efficient resource allocation and reduced downtime. However, the workload's nature plays a significant role in how the convergence point is determined, influencing migration efficiency and overall performance.

Workloads are generally classified into two main categories: static and dynamic.

A static workload maintains a consistent demand for computing resources over an extended period while dynamic workloads frequently change their behavior, leading to varying demands on computing resources. This variability makes dynamic workloads more complex to predict and manage, as it becomes challenging to determine the workload's next state accurately.

For static workloads, identifying the convergence point is relatively straightforward. During migration, there comes a point where the same set of memory pages (known as the working set) stabilizes and no longer changes. This stable state marks the convergence point, allowing the migration to complete smoothly.

In contrast, dynamic workloads require additional effort due to their unpredictable nature. The working set in dynamic workloads can change at any time, making it challenging to identify a stable convergence point. To address this, machine learning and statistical techniques are commonly employed. These methods help predict and manage the variability of dynamic workloads, providing more reliable solutions for achieving a stable convergence point during migration.

1.2 Background

1.2.1 Workload Categorization

VMs need to be migrated to prevent interruptions caused by server node failures. However, the methods used for migration depend heavily on the workload type. Workloads are generally classified into two main categories: static and dynamic.

A static workload maintains a consistent demand for computing resources over an extended period. Due to its stable nature, the behavior of a static workload is relatively easy to predict based on the current usage patterns, making the migration process more straightforward.

Within the category of dynamic workloads, there are several specific types, each with unique characteristics:

1. **Persistent Workloads**: These workloads are continuous and stable in resource demand, though they can experience occasional spikes. Applications with long-running tasks, such as databases, often generate persistent workloads (YCSB 2024). **Database Management Systems (DBMS) like MySQL, PostgreSQL, or MongoDB** can be given as examples of persistent workloads.

2. **Real-Time Workloads**: Real-time workloads require immediate processing with minimal latency, often found in applications like video streaming or IoT sensors. These workloads are highly sensitive to delays, making migration techniques for real-time workloads particularly complex, as they must minimize any disruption during the migration process. **Video Streaming Platforms like YouTube** and **Augmented Reality (AR) Applications** are few examples of real-time workloads.

3. **Serverless Workloads**: Serverless workloads(AWS 2024, Oracle 2024) are characterized by rapid, unpredictable demand changes, as they only consume resources when triggered by specific events. Serverless applications, such as cloud-based functions, make workload prediction challenging since they can vary significantly over short periods. **AWS Lambda, Google Cloud Functions, and Oracle** can be given as examples of serverless workloads.

4. **Interactive Workloads**: These workloads involve direct user interactions, such as online gaming, virtual desktops, and collaborative applications. Interactive workloads require low latency and high responsiveness to ensure a seamless user experience. **Google Docs, Microsoft Teams, and Slack** are some of the interactive workloads.

5. **Burstable Workloads**: Burstable workloads experience sudden and intense spikes in resource demand for short durations, followed by periods of low or no activity. Examples include web servers handling unexpected traffic surges or financial trading platforms during market opens. **Social Media Platforms and Websites like Twitter or Instagram** are a few examples of burstable workloads.

Understanding these dynamic workload categories is essential for designing effective migration strategies that can maintain performance and availability across diverse application types. By tailoring migration techniques to the specific characteristics of each workload type, cloud data centers can enhance reliability, optimize resource usage, and ensure seamless application performance even in the face of varying demands.

Handling the dynamic nature is a challenging task because the frequently changing behavior of dynamic workloads doesn't lead to straightway convergence or smooth transitions during the migration. Because of that dynamic workloads require specialized techniques to handle their variability, including Profiling (Ye et al. 2014, Wu & Wolf 2008, Han et al. 2020) which involves recording past behaviors to help predict future workload patterns. In addition, statistical and machine learning (ML) methods (Wei et al. 2018, Naik 2022) are commonly used to handle the unpredictable nature of dynamic workloads. These techniques contribute to more accurate forecasting of workload changes, thereby optimizing the migration process. Further details on these methods are discussed in section 2.1.

1.2.2 Live VM Migration

Live VM Migration is a process designed to maintain application continuity when failures occur mid-execution. During Live VM Migration, the Virtual Machine (VM) is transferred from a failing server (source) to another server (destination) without interrupting the applications running inside the VM. This ensures minimal downtime and preserves the seamless operation of applications despite the underlying server issues. Christopher (2005) has shown several advantages of live migration as

- In live migration the entire operation system and all of its applications are moved at once. This will avoid many challenges encountered by process-level migration methodologies. (Christopher 2005)
- After the migration the host machine does not depend on the source and this solves the problem of residual dependencies.
- Live VM migration has the ability to complete the migration process without providing an interruption to the users.

Live VM migration can be categorized broadly into three(3) main techniques as **Pre-Copy migration, Post-Copy migration, and Hybrid migration**(He 2021, Christopher 2005, Fernando et al. 2020, Hines et al. 2009).

1. **Pre Copy Migration**
   In Pre-Copy(Christopher 2005, Fernando et al. 2020)migration, the memory transferring will happen via two(2) main steps as **push phase and stop and copy phase.**
   (a) Push phase
   Memory pages are iteratively sent from the source to the destination while the source VM continues to run. In the first iteration, all pages are transferred, and in subsequent iterations, only modified pages (dirty pages) are sent. For static workloads, the dirty page count eventually stabilizes, marking a convergence point. However, for dynamic workloads, the dirty pages may change continuously, making it hard to find a convergence point.
   (b) Stop and copy phase
   After identifying the convergence point the source VM holds its execution and quickly sends the remaining pages, CPU state, and the I/O state to the destination VM. From this point onwards the destination VM starts to be executed. The duration that source VM paused its execution and started resuming in the destination is known as the **downtime** of the VM migration.

2. **Post-Copy Migration**
   The execution nature of the Pre-Copy technique is used to send the dirty pages to the destination through multiple iterations. But in Post-Copy (Hines et al. 2009, Fernando et al. 2020) migration it uses a mechanism that ensures no duplicates of the same page will be sent to the destination. The steps of this technique are as follows.

   (a) First, the VM at the source server paused its execution.Then, a minimal set of memory pages required for execution of the destination VM will be moved to the destination server.

   (b) Then the destination VM starts its execution and the source quickly starts sending the remaining memory pages to the destination. This process is known as **pre-paging**.

   (c) If the destination VM may notice that some pages have not been successfully sent to the destination, a page fault will occur and the source quickly starts sending the missing pages to the destination. This is known as the **Demand paging**

3. **Hybrid Migration**
   Hybrid migration(Sahni & Varma 2012, Altahat et al. 2020) can be known as a combination of both Pre-Copy and Post-Copy techniques. Hybrid migration aims to increase the performance for both read-intensive and write-intensive workloads by minimizing downtime and avoiding redundant data transfers. Hybrid algorithm stages can be described as given below.

   (a) Migration begins with Pre-Copy, where memory pages are sent from the source to the destination VM while the source VM remains active. Unlike traditional Pre-Copy, Hybrid Migration does not wait for a convergence point.

   (b) After a short Pre-Copy period, the source VM is paused, and its CPU and I/O states are transferred to the destination. The VM resumes on the destination immediately, without waiting for all pages to transfer.

   (c) As the final step the Post-Copy algorithm starts the execution and the remaining dirty pages will copy from the source to the destination.

1.3 Motivation

With advancements in technology, cloud computing(CC) has become essential for users globally. It offers the ability to create and customize applications via the Internet, providing cost-effectiveness and flexibility in accessing resources. Leading cloud service providers, such as Cloud(GCP 2024), Amazon Web Services (AWS)(AWS 2024), Microsoft Azure(Microsoft 2024), and Oracle(Oracle 2024) utilize virtual machines (VMs) to deliver scalable, on-demand computing resources.

Given the potential for server failures at any moment, it is vital to implement effective strategies for managing these disruptions. VM migration has emerged as a critical solution to ensure continuous application availability. Over the years, various optimizations have been developed to reduce the total migration time and minimize the downtime of the migration process.

However, many applications today operate under dynamic workloads, characterized by variability and complexity. These dynamic environments present several challenges (Zhang & Boutaba 2014, Hossain & Song 2016, Cerotti et al. 2012) that complicate the migration process. A significant challenge is the difficulty in predicting future workload behavior, which hinders the selection of the most suitable migration strategy. Not selecting the best migration technique may lead to a longer Total Migration Time (TMT), increasing the risk that a server may fail before the migration process is completed. Additionally, once migration has started, determining the convergence point, when the process should ideally be completed, becomes challenging due to fluctuating workload conditions. This unpredictability can lead to resource inefficiencies, highlighting the need to improve the performance of live migration techniques tailored for dynamic workloads.

1.4 Research Questions

1. How to identify the convergence point of dynamic workloads in live migration with workload-specific behavior?
   This question examines using Markov Chains to detect the convergence point, considering how workloads like persistent, real-time, and serverless uniquely impact convergence.

2. How can future workload behavior prediction be used to select the most optimal VM migration technique for dynamic workloads?
   The main idea is to select the best migration technique by considering the future behavior of a dynamic workload. This focuses on improving the performance of the applications by reducing the total migration time, and downtime with minimal impact on the performance.

1.5 Aims and Objectives

1.5.1 Aim

The primary goal of this research (ProMig) is to develop a method for selecting the optimal migration strategy for dynamic workloads and determining the optimal convergence point during migration based on the chosen technique. ProMig aims to enhance migration efficiency while reducing total migration time and downtime compared to traditional approaches. Instead of making decisions solely based on current workload behavior, ProMig predicts future workload trends, analyzes these predictions, and selects the most suitable migration strategy accordingly.

1.5.2 Objectives

The main objectives of the research are as follows:

1. Design and develop a method to predict the future behavior of a dynamic workload.
2. Design and develop an algorithm to select the best migration strategy to migrate a dynamic workload.
3. Identify the convergence point of a dynamic workload based on the selected migration strategy.
4. Evaluate the performance of migration using total migration time, downtime, and application performance metrics by incorporating industrial accepted benchmarks such as Sysbench, YCSB (Yahoo cloud serving benchmark), Stress, Memcached, etc. These benchmarks support generating workloads of different natures such as **persistent workloads, real-time workloads, CPU-intensive workloads, and memory-intensive workloads.**

1.6 Scope

- **Live VM migration with dynamic workloads**: The research will focus on developing a strategy to migrate a dynamic workload using Live VM migration.

- **Single VM migration**: The research focused on migrating a dynamic workload in a single VM.

- **Implementing a working prototype**: At the end of the research, a working prototype will be developed to find the convergence point and migrate a dynamic workload with minimal total migration time and downtime.

- **QEMU-KVM Hypervisor**: The research uses QEMU-KVM as the hypervisor for implementing and evaluating the working prototype.

- **Ubuntu Host OS**: The research will focus on Ubuntu Servers as the host OS.

- **Linux guest OS**: Linux will be considered as the guest OS for this research.

1.7 Dissertation Outline

The rest of this dissertation is organized as follows: Section 2 reviews the related work for this research. Section 3 discusses the challenges that occurred when time series forecasting was used for future workload prediction. Section 4 presents the design and methodology details of ProMig. Section 5 provides the implementation details of ProMig, and section 6 evaluates the performance of ProMig using various metrics and compares it with the traditional migration techniques. Finally section 7 provides a discussion and conclusion of the results and the section 8 provides the limitations and future works.

2 Background

A literature review was conducted for the topic **Live VM Migration Convergences in a Dynamic Workload** and the report consists of a detailed explanation of live migration techniques, an explanation about the dynamic workload, and the different types of dynamic workload handling methods with their advantages and limitations.

2.1 Dynamic workload

A dynamic workload (Hossain & Song 2016, Cerotti et al. 2012) consists of several computational tasks that change the behavior of tasks very frequently over time. Because of its changing behavior, it is hard to predict workload patterns for dynamic workloads. This makes dynamic workload handling a challenging and complex task. Some of the key challenges of dynamic workloads are given below.

1. Dynamic workloads consist of different types of applications that consist of their own and different kinds of resource requirements, which is known as the **Heterogeneity** (Zhang & Boutaba 2014).
2. Dynamic workloads have the problem of over-provisioning and under provisioning. Because of the dynamic nature, it is hard to predict the resource demand hence, most of the applications provide resources more than enough to execute the operations. This may lead to a waste of resources. In similar cases, the resources provided may not be enough for the applications to execute, which leads to under-provisioning and failures in the application process. This is known as the applications do not meet their **peak demands**. In both cases, part of or the whole cost used for the application process has become a waste (Hossain & Song 2016).

Dynamic workloads can be handled based on the following approaches:

- Mathematical/Machine Learning Techniques
- Statistical Techniques

To deal with dynamic workloads, these techniques require capturing and collecting records of the current behavior of the workload to predict future behaviors. For that process, a well-known technique called **profiling** has been used by several researchers(Wu & Wolf 2008, Ye et al. 2014, Han et al. 2020).

2.2 Profiling

Profiling is a concept used to capture the behavior of workloads based on their characteristics. There are two profiling techniques runtime profiling and offline profiling (Wu & Wolf 2008). Runtime profiling tries to capture the characteristics of an application when the application starts its execution. Because of that, runtime profiling (Wu & Wolf 2008) is used to deal with dynamic workloads. The captured data will be used to increase application efficiency. CPU usage, memory usage, and network consumption are some of the runtime profiling information.

The studies conducted by Wu & Wolf (2008),Ye et al. (2014), and Han et al. (2020) have used profiling techniques to identify the behavior of a workload. The main idea behind the concept used by these research works is to gather runtime profiling information such as CPU time, memory usage, and service time and understand the workload behavior using the collected information. The study followed by Wu & Wolf (2008) collects real-time information such as task service times, edge utilizations, and task utilizations and then uses a duplication and mapping algorithm that can capture the changes in the workload by using the collected profiling information. So the tasks with high computational demands will be duplicated across multiple cores by the task duplication algorithm. The task mapping algorithm will assign tasks to the processors to minimize the overhead. Also, the study of Ye et al. (2014) collects profile information for various types of workloads such as CPU-intensive, memory-intensive, and network-intensive. Then by analyzing these details, the characteristics and resource demands of different types of workloads will be identified. Finally, with the use of these details, they implemented two(2) models as a consolidation planning module and a migration planning module to complete the migration process with minimal effect on the performance. Han et al. (2020) has introduced a combination of profiling systems and refinement frameworks to identify micro-services placement with dynamic workloads.

One of the most important points highlighted by the above studies (Wu & Wolf 2008, Ye et al. 2014, Han et al. 2020) is profiling techniques has the ability to measure the system performance for various workload conditions(CPU intensive, memory intensive, network intensive, etc). So the workload changes can be identified in real time. Also profiling incurs low overhead because profiling can directly monitor specific metrics and uses simple analytical techniques which do not require high computational power.

2.3 Workload Prediction

1. **Mathematical/Machine Learning Techniques**
   Various research efforts have addressed the challenges of managing dynamic workloads, utilizing methods such as dynamic thresholds (Lin et al. 2011), reinforcement learning (Wei et al. 2018), use of autonomic computing (Sah & Joshi 2014), etc.

   A study conducted by Lin et al. (2011) proposed a dynamic resource allocation schema that adjusts resources according to the application's changing workload. Given that dynamic workloads can fluctuate throughout the application's lifecycle, this schema uses a threshold value to determine optimal instances for resource allocation. Based on this threshold, resources are allocated dynamically, adapting to the application's immediate needs.

Similar systems can be found in the solution provided by Sah & Joshi (2014), a solution based on autonomic computing. Autonomic computing refers to the term that applications can change themselves according to the changes of the application environment without any external help(Without notifying the user). The algorithm created using autonomic computing, consists of a capacity and utility agent (CUA) and a resource controller. CUA is used to collect data about the resources and the capacity of the VMs. According to the collected data, an initial schedule was implemented, and if the workload changes then the resource controller will change the schedule to allocate resources in a better way.

Enhanced Dynamic Johnson Sequencing is another technique proposed by Banerjee et al. (2023), that uses a combination of ML and RTM to handle the nature of a dynamic workload. It is also known as OptiDJS+. The algorithm provides a schedule based on the dynamic quantum value (minimum execution time + maximum execution time)/2 of the tasks.

Also the framework proposed by Wei et al. (2018) claims a workflow scheduling algorithm using a sequential decision-making approach based on reinforcement learning. The ML approach used in this proposed algorithm is called the Q-learning. This algorithm aims to identify suitable Infrastructure as a Service (IaaS) providers for Software as a Service (SaaS) applications, optimizing resource allocation and minimizing costs. The Q-learning approach continually learns and adapts based on workload changes, making it particularly suited to dynamic environments.

As explained, several instances of using machine learning to handle dynamic workloads exist. However, there are a few important points to be considered when using ML techniques. To provide an accurate prediction, ML models need to be trained properly (Wei et al. 2018). This required extensive and accurate training data to make reliable predictions. But in this instance acquiring data from a dynamic workload can be challenging since its workload behavior will change frequently(Khelghatdoust et al. 2016). Additionally, dynamic environments increase the risk of overfitting, where a model becomes too specialized to the training data, resulting in reduced effectiveness for real-world applications(Wei et al. 2018). Also implementing a proper model can be resource-intensive because it requires a high computational power to train and run complex models.

2. **Statistical Techniques** Statistical techniques have gained popularity for predicting future workloads due to their simplicity, lower data requirements, and transparency in operations (Devi & Valli 2023, Calheiros et al. 2014). Many studies in this area focus on time series forecasting, which leverages historical data to identify patterns over time, providing insights into future workload trends. Another common approach is the use of Markov chains, which model workload transitions between states, allowing predictions based on probable sequences. These statistical methods are especially valuable for dynamic workloads, offering an accessible and computationally efficient alternative to more complex machine learning models.

    (a) **Time Series Forecasting**

    Calheiros et al. (2014) has conducted a study about workload prediction using **ARIMA** model. ARIMA stands for the Auto-Regressive Integrated Moving Average and it is used to predict the future workload. This paper has introduced an analyzer implemented using the ARIMA model. First, the data that needs to predict the future workload such as CPU usage, memory usage, etc will be fed into the ARIMA model. An important point considered here is these data must be stationary. For that process, a transformation method called differencing has been used. ARIMA model needs three(3) parameters to be fed the number of differences used, the auto-correlation value, and the partial auto-correlation value. The analyzer used in this paper will be responsible for updating the model with new data to increase the prediction accuracy of future workload behaviors. These predicted results have been used to ensure that enough resources are available for users to complete their tasks. With that Calheiros et al. (2014) has been able to increase the quality of service(QOS) by predicting future behavior using the ARIMA model.

    The study conducted by Devi & Valli (2023) has used a combination of the ARIMA model and ANN(Artificial Neural Network) to predict the behavior of future workloads. So this hybrid model can be introduced as an enhanced version of Calheiros et al. (2014). ARIMA model was used to deal with the linear components and ANN was used to model the non-linear components. With that, the accuracy of the future workload prediction has been increased. Finally, the study has evaluated the performance of the new method using metrics such as mean absolute error and root mean squared error and the results have proven that the proposed solution works efficiently in a dynamic cloud environment to predict future workloads.

    Ganapathi et al. (2010) has proposed a statistic-driven working model that can be used to predict the future behavior of a workload. Mainly the paper tries to predict the resource demands for applications that use MapReduce(Hashem et al. 2016). Mapreduce tasks consist of several jobs which are known as Hadoop jobs. The proposed solution named as **Kernel Canonical Correlation Analysis (KCCA)** will predict the execution times, and resource demands for the Handoop jobs. KCCA consists of a special feature called **Feature Vectors** which are constructed using job configuration parameters and input data characteristics. With the use of feature vectors, KCCA was able to provide accurate predictions about the future workloads.

(b) **Markov Chains**

The research conducted by Kim et al. (2018) introduced a predictive framework called **CloudInsight** that addresses the limitations of reactive autoscaling by adapting to both regular and irregular workload patterns. A key component of this framework is the use of **Markov chains**, which model the transitions between different workload states to capture short-term, bursty fluctuations. This allows the system to react quickly to sudden changes in workload behavior—something traditional predictors often struggle with. In combination with **Fast Fourier Transform (FFT)**, which identifies periodic workload trends, the Markov model enhances CloudInsight's ability to make accurate and timely predictions.These predictors are integrated into an ensemble-based approach, where each model acts as an expert in a "council of predictors." The ensemble weights are updated in real time using multi-class regression based on the predictors' accuracy under current conditions. The use of Markov chains thus plays a central role in improving responsiveness and accuracy in dynamic cloud environments.

Pacheco-Sanchez et al. (2011) uses Markovian Arrival Processes (MAP) to model and predict the time-varying characteristics of cloud workloads, specifically for web traffic. This approach allows for performance prediction, including metrics such as server utilization, queue length, and response times. MAPs capture not only the distribution of workload arrival rates but also the temporal dependencies in these rates, accommodating heavy-tail distributions common in HTTP traffic. By fitting MAP parameters to real HTTP log data, the framework enables efficient, queueing-based performance analysis without needing extensive simulation. This model is then used to predict Quality of Service (QoS) metrics, supporting resource allocation decisions in cloud environments by ensuring the provision of sufficient resources for varying workload intensities

Also, the research conducted by Gambs et al. (2012) uses a n-Mobility Markov Chain (n-MMC) to predict future locations by modeling an individual's movement patterns based on the last **n** locations visited. Each location or "point of interest" (POI) represents a state, with transitions indicating the probability of moving from one POI to another. By considering sequences of locations rather than just the current location, the n-MMC model improves prediction accuracy, capturing more complex mobility patterns.

In recent years, application providers have increasingly favored statistical approaches to predict future workload behaviors due to their simplicity and ease of interpretation. Unlike machine learning techniques, statistical methods require smaller datasets to produce effective models (Calheiros et al. 2014, Devi & Valli 2023), making them more accessible and efficient for real-time predictions. Additionally, statistical models demand less computational power (Hashem et al. 2016) during both training and execution, making them a practical choice for environments with limited resources or tight processing requirements. This combination of advantages makes statistical approaches highly suitable for many predictive applications in workload management.

2.4 Existing Dynamic Workload Migration Strategies

Naik (2022) has claimed a solution for dynamic workload migration using a technique called **Adaptive Push-Pull**. This algorithm consists of a combination of a push function and a pull function. In the concept of Adaptive Push-Pull the push function is used to distribute the workload among the resources, and when a VM is overloaded . The pull function is invoked when the VM is underloaded . For this process, a VM manager and the CRM are used. VM manager will be responsible for managing local workload distribution, while a CRM (Cloud Resource Manager) manages global resource balancing. This adaptive approach ensures efficient load sharing, reducing migration overhead and improving resource utilization.

Also similar systems can be found in Lu et al. (2015), an optimal scheduling algorithm called **VHaul** to solve the problem of migrating multi-tier applications. These multi-tier applications consist of a group of co-related VMs that makes the application inherently dynamic. So to create the scheduling algorithm the VMs will be categorized according to their relationships. VMs belonging to the same application will be assigned to the same group. Then the production of resource utilization and migration time are used to decide the impact from the current VM to the next VM to be migrated. Then the migration schedule will be created as the smaller value for the production of resource utilization and migration time means less impact for the next VM to be migrated.

Khelghatdoust et al. (2016) came up with a solution **GLAP**, which is a combination of a gossip-based learning algorithm and Q-learning. GLAP uses continuous monitoring of the resource demands of the VMs to predict the behavior of future workloads. The table 1 shows the comparison of the existing methods.

	Pros	Cons
Adaptive Push-Pull	• Provides an adaptive push-pull system to dynamically switch between push and pull functions to react to the changes in the workload. • Solves the problem of overloading and underloading in the VMs.	• With a highly dynamic workload CRM and VMM will not be able to handle the problem of overloading.
vHaul	• Provide an efficient scheduling algorithm by considering the resource utilization and the migration time.	• Have to calculate the resource utilization and migration time whenever the workload changes.
GLAP	• Provides a threshold-free approach to detect future behaviors and migrate a workload without overloading.	• Have to use a lot of computation power and resources to train an accurate model

Table 1: Comparison of dynamic workload migration techniques

Among the existing approaches, Adaptive Push-Pull handles overloading and underloading using the VM Manager and Cloud Resource Manager, but lacks predictive capabilities. VHaul focuses on multi-tier application migration by estimating resource utilization and migration time, but does not directly address dynamic workload migration. GLAP applies machine learning for VM placement but introduces high computational overhead. Compared with these techniques, the solution of this research aims to offer a novel and lightweight solution for dynamic workload migration. By anticipating future workload states, it enables smarter and more efficient migration decisions, making it a more adaptive and resource-aware alternative to the above methods.

2.5 Research Gap

Migration operations are often triggered by sudden resource or power failures, maintenance needs, or load-balancing requirements. However, deciding the future behavior of a workload for migration is challenging, especially with dynamic workloads where CPU, memory, and storage demands fluctuate rapidly. This unpredictability complicates decisions on identifying the convergence point. Additionally, the variable behavior of workloads, whether they become CPU-intensive, memory-intensive, or network-intensive, can influence the success of the migration process. Selecting an incorrect migration strategy can further increase the total migration time (TMT) and elevate the risk of server failure during migration. This highlights the importance of making timely and accurate predictions.

Existing research on dynamic workload migration (Section 2.3) has primarily focused on developing complex mechanisms to adapt migration schedules based on workload changes. Many of these solutions rely on data such as resource utilization metrics, migration durations, and the load status of VMs (whether they are overloaded or underloaded). This information, however, requires significant processing time to analyze and interpret, which can delay response to real-time changes in workload demands.

Machine learning and statistical techniques provide potential solutions for forecasting workload behavior before migration. Machine learning methods, while effective, require extensive training periods, large datasets, and considerable computational resources (Khelghatdoust et al. 2016, Wei et al. 2018). By contrast, statistical approaches demand smaller datasets (Calheiros et al. 2014, Devi & Valli 2023) and lower computational power (Hashem et al. 2016) for both training and execution. Studies by Calheiros et al. (2014), Hashem et al. (2016), and Devi & Valli (2023) demonstrate that even with limited computational resources, statistical techniques can reliably predict future workload behaviors.

In VM migration, the full state of the VM must be transferred to the target machine, which demands substantial bandwidth and is recognized as a resource-intensive process. Employing techniques with high computational overhead to handle workload dynamics can further strain the system, negatively impacting applications running on the VMs. Therefore, this research seeks to bridge the gap by developing a lightweight, performance-efficient solution for dynamic workload migration. By using profiling information (e.g., CPU, memory usage, and network usage), this solution(**ProMig**) will employ statistical models to predict future workload behaviors and schedule migrations accordingly. The research aims to minimize both the total migration time and downtime, reducing the performance impact on applications during migration and mitigating the risk of server failure before completing the migration process.

3 Challenges in Time Series Forecasting

Time series forecasting was explored as a starting point to predict future workload behavior. There are two(2) main categories to be considered as univariate and multivariate time series models.

- **Univariate Models:** Models such as ARIMA, ETS, and FB Prophet can predict a single variable at a time. For this research, using univariate models would require four separate models, one each for CPU, memory, incoming network, and outgoing network.
- **Multivariate Models:** VAR (Vector AutoRegression) is a multivariate model capable of predicting multiple variables simultaneously, making it potentially more suitable for predicting all four resources together.

To determine the most resource-efficient approach, experiments were conducted to compare the resource usage of these models during predictions. These tests provided insights into the computational demands of each model type, guiding the selection of an optimal prediction method for dynamic workloads in cloud environments. Based on the results, multivariate forecasting was identified as the less resource-intensive prediction model. Results will be shown in figure 1.

However, several challenges arise when using time series forecasting. A significant issue was that even within the same workload type, different workloads required distinct parameter settings for the models. Consequently, the algorithm needed extra time to analyze past data and adjust parameters to fit each workload accurately, making it difficult to develop a generalizable solution.

Additionally, prediction accuracy varied across resources. Some resources were accurately forecasted, while others showed substantial prediction errors. For instance, as shown in figure 2, the model accurately predicted memory usage but struggled to forecast incoming network usage for the same workload.

To address these limitations and to develop a more generalized solution, the approach was shifted to using **Markov chains**(Kim et al. 2018, Pacheco-Sanchez et al. 2011). Unlike time series models, which require tuning specific parameters for each workload, Markov chains offer a state-based approach that can adapt more easily to varying workload patterns without extensive reconfiguration. By modeling resource usage levels as states with probabilistic transitions, Markov chains enable a more flexible and scalable framework for predicting dynamic workloads across multiple resources, mitigating the need for constant parameter adjustments. This approach simplifies workload prediction and enhances consistency across different types of resources.

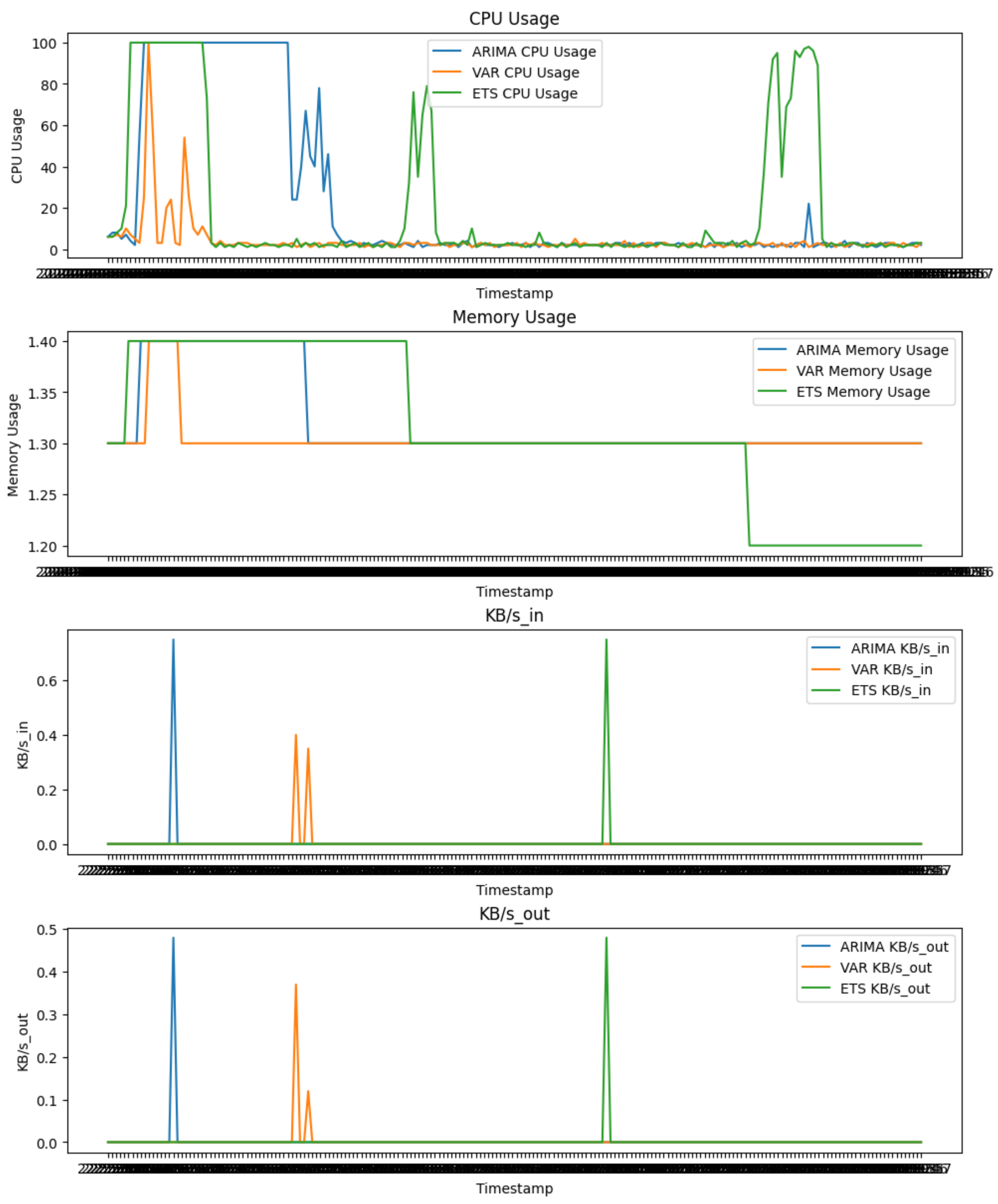

Figure 1: Univariate Forecasting Vs Multivariate Forecating

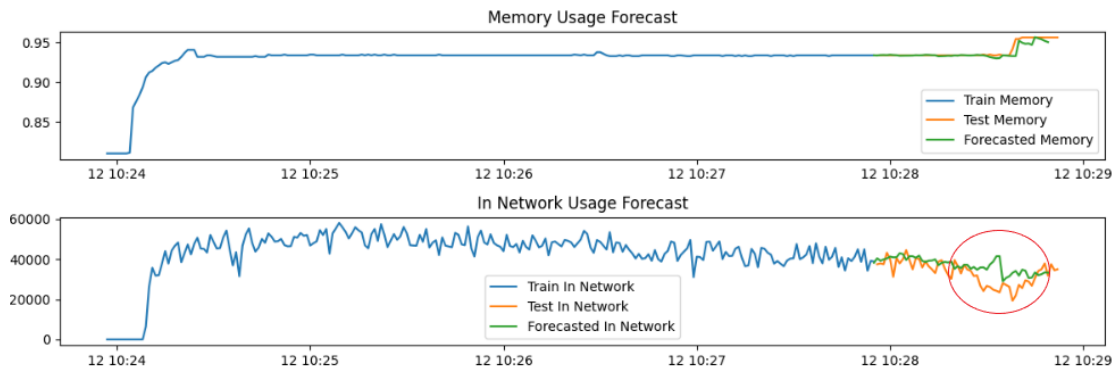

Figure 2: Forecasting Results for YCSB Workloads

4 Design and Methodology

4.1 Experimental Testbed

1. Test-bed setup: The test-bed setup consists of two physical servers that are interconnected using a Gigabit Ethernet. QEMU/KVM is chosen as the hypervisor and a Ubuntu Server is used as the host OS. The VMs consist of a Linux-based OS. The following figure provides a high-level view of the test-bed setup.

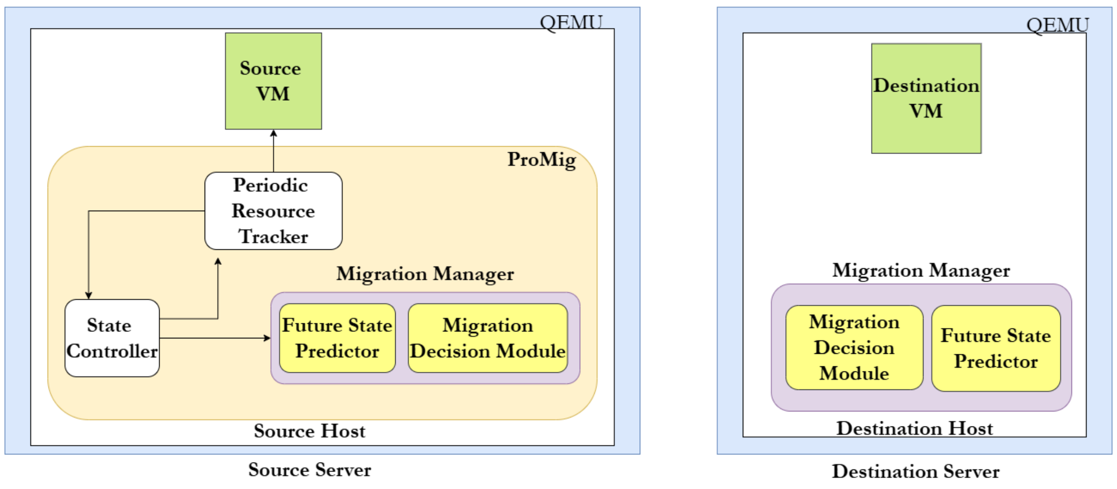

Figure 3: ProMig Architecture

4.2 Main Components of ProMig

ProMig consists of three(3) main components. As shown in the figure 3, there is a Resource Tracker, a State Controller, and a Migration Manager. All three(3) components will be implemented inside the source host. The use case of each of these components is as follows.

- **Periodic Resource Tracker**- Captures the resource usage of the source VM. So it captures the CPU, memory, and incoming and outgoing network usage. For that it uses profiling tools such as **mpstat**, **free -m**, and **Ifstat**. For each second(1s) the resource tracker captures the resource usage of the source VM, converts it to a percentage, and then feeds the results to the state controller.

- **State Controller** - State Controller consists of two(2) main components the count matrix and the transition matrix. For each second it takes input from the resource tracker and based on that the count matrix and the probabilities in the transition matrix will be updated.

- **Migration Manager**- Once a migration is triggered the control goes to the Migration Manager. It consists of a Future State Predictor and a Migration Decision Module. The Future State Predictor predicts the future behavior of the workload with the help of the probability matrix. Then, the Migration Manager analyzes these results and makes the best migration decision by considering current and future behaviors.

Implementation details of these components will be discussed in the section 5.

5 Implementation

In this section, the implementation details of the Resource Tracker, State Controller, and Migration Manager will be discussed.

5.1 Periodic Resource Tracker

Resource Tracker captures the CPU, memory, and incoming and outgoing network of the source VM. The commands and the profiling tools used to capture the resource usages are as follows.

- CPU Usage : To capture the CPU usage the mpstat profiling tool is used. The command that captures and returns the CPU percentage is as follows.
  ```
  mpstat 1 1 | awk '/Average/ {print 100 - $NF}'
  ```
  In the mpstat command, the last column contains the idle CPU usage as a percentage. So to take the used CPU percentage that value will be deducted from 100.

- Memory Usage: To capture the memory usage the free profiling tool is used. The command that captures and returns the memory percentage is as follows.
  ```
  free | awk '/^Mem:/ {print $3 / $2}'
  ```
  Here in the result, 3rd column contains the used memory and the second column contains the total memory usage. So to calculate the percentage value the result of the 3rd column is divided by the result if the second column.

- Network Usage: To capture the network usage the ifstat profiling tool is used. The command that captures and returns the incoming and outgoing network percentage is as follows.
  ```
  ifstat -i eth0 1 1 | awk 'NR==3 {print $1, $2}'
  ```
  This gives both incoming and outgoing network usage for that second. For the experiments, a 1000Mbps Erthenet was used. The result of the ifstat command comes in KBps. So to calculate the network usage as a percentage the result from the ifstat command will be divided with the 125000KBps(1000 MBps converted to KBps).

The Pseudo code for this is shown in algorithm 1.

Algorithm 1 Periodic Resource Tracker

```
function GET_CPU_USAGE( )
    Run command to get CPU usage (e.g., from mpstat)
    Parse output to calculate %CPU usage
    return CPU usage percentage
end function
function GET_MEMORY_USAGE( )
    Run command to get memory usage and calculate %memory usage(e.g., from free -m)
    Parse output to get used memory
    return memory usage percentage
end function
function GET_NETWORK_USAGE( )
    Read current incoming and outgoing network bytes (e.g., from Ifstat)
    Calculate Net_In and Net_Out percentage by dividing from the maximum capacity of the ethernet.
    return Net_In, Net_Out
end function
while True do
    Get current timestamp
    cpu_usage = get_cpu_usage()
    memory_usage = get_memory_usage()
    net_in, net_out = get_network_usage()
    Sleep for 1 second
end while
```

5.2 State Controller

The Resource Tracker's output will be mapped to a state based on the captured resources, and the count and transition matrices will be updated.

5.2.1 State Definition

To define a set of states **CPU** usage, **memory** usage, **incoming network** usage, and **outgoing network** usage will be taken as the resources and **less than X%** and **greater than X%** will be taken as the resource usage levels. Based on these conditions, 16 possible states are identified, representing all combinations of resource usage levels. Here are a few example states:

- **state 0**: CPU $<$ X%, Memory $<$ X%, Incoming Network $<$ X%, Outgoing Network $<$ X%
- **state 1**: CPU $<$ X%, Memory $<$ X%, Incoming Network $>$ X%, Outgoing Network $<$ X%
- **state 2**: CPU $>$ X%, Memory $>$ X%, Incoming Network $<$ X%, Outgoing Network $>$ X%

5.2.2 Implementation of the transition matrix and count matrix

A transition matrix and a count matrix were created to model workload behavior using Markov chains to define all possible state transitions.
CPU, memory, incoming network, and outgoing network usage can be monitored and mapped into percentages by the Resource Tracker to determine the **current state** of the VM accordingly.

The **Count Matrix** represents the number of times the VM state changes from one state to another during the observation period. Since there are 16 possible states (based on CPU, memory, and network usage levels), this matrix will have dimensions of 16×16. Each entry (i, j) in the matrix records the count of transitions from state i to state j.
Example: Suppose during the observation, the VM moves from:

- state 1 to state 2 (5 times),
- state 2 to state 3 (3 times),
- state 1 to state 3 (2 times).

In this case, the count matrix will be updated as follows.

- $Count(1, 2) = 5$, representing 5 transitions from state 1 to state 2,
- $Count(2, 3) = 3$, representing 3 transitions from state 2 to state 3,
- $Count(1, 3) = 2$, representing 2 transitions from state 1 to state 3.

The **Initial Transition Matrix** is derived from the Count Matrix and represents the probability of moving from one state to another. It is also a 16×16 matrix. Each entry (i, j) is calculated as the ratio of the number of transitions from state i to state j to the total number of transitions out of state i.
Example: Using the counts from above, if there were 10 total transitions from state 1, the transition probability is updated as follows:

- $Transition(1,2) = \frac{Count(1,2)}{10} = \frac{5}{10} = 0.5,$
- $Transition(1,3) = \frac{Count(1,3)}{10} = \frac{2}{10} = 0.2.$

If there were a total of 6 transitions from state 2, then:

- $Transition(2,3) = \frac{Count(2,3)}{6} = \frac{3}{6} = 0.5.$

5.2.3 Update the count and transition matrices

Let's consider a VM with three possible states based on CPU and memory usage:

- **state 1**: Low CPU and memory usage
- **state 2**: Medium CPU and memory usage
- **state 3**: High CPU and memory usage

Initially, the **Count Matrix** is set as follows, representing previously observed transitions between states:

$$\text{Count Matrix} = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 0 & 4 \\ 2 & 1 & 0 \end{bmatrix}$$

The **Transition Matrix** is calculated by dividing each element in a row of the Count Matrix by the total number of transitions from that state. For instance:

- For state 1, there are $2 + 3 + 1 = 6$ total transitions.
- For state 2, there are $1 + 0 + 4 = 5$ total transitions.
- For state 3, there are $2 + 1 + 0 = 3$ total transitions.

The resulting **Transition Matrix** is:

$$\text{Transition Matrix} = \begin{bmatrix} \frac{2}{6} & \frac{3}{6} & \frac{1}{6} \\ \frac{1}{5} & 0 & \frac{4}{5} \\ \frac{2}{3} & \frac{1}{3} & 0 \end{bmatrix} = \begin{bmatrix} 0.33 & 0.50 & 0.17 \\ 0.20 & 0 & 0.80 \\ 0.67 & 0.33 & 0 \end{bmatrix}$$

In the next second, the periodic resource tracker captures the following VM resource usage:

- **CPU Usage:** 60%
- **Memory Usage:** 55%

Based on these percentages, the current state of the VM is classified as **state 3** (High CPU and memory usage).

Assume that in the previous second, the VM was in **state 1** and has now transitioned to **state 3**. So the value of the Count(1,3) will be incremented by 1. This transition updates the Count Matrix as follows:

$$\text{Updated Count Matrix} = \begin{bmatrix} 2 & 3 & 2 \\ 1 & 0 & 4 \\ 2 & 1 & 0 \end{bmatrix}$$

Now, the Transition Matrix is recalculated using the updated Count Matrix:

- For state 1, there are now $2 + 3 + 2 = 7$ total transitions.
- For state 2, transitions remain at 5.
- For state 3, transitions remain at 3.

The **Updated Transition Matrix** is:

$$\text{Updated Transition Matrix} = \begin{bmatrix} \frac{2}{7} & \frac{3}{7} & \frac{2}{7} \\ \frac{1}{5} & 0 & \frac{4}{5} \\ \frac{2}{3} & \frac{1}{3} & 0 \end{bmatrix} = \begin{bmatrix} 0.29 & 0.43 & 0.29 \\ 0.20 & 0 & 0.80 \\ 0.67 & 0.33 & 0 \end{bmatrix}$$

This updated Transition Matrix now reflects the most recent observed behavior, providing an improved model for predicting future state transitions based on the latest profiling data. This update process repeats for each second with the use of the resource tracker's output.

5.3 Migration Controller

When migration is triggered, the migration controller is activated. The implementation of the two(2) main components is as follows.

5.3.1 Future State Predictor

Future State Predictor will analyze the **current state** of the VM and the **transition matrix** to predict the VM's next behavior over n look-ahead steps.
The steps to calculate the next state are as follows. Consider the transition matrix created in section 5.2.3. Assume a look-ahead of $n = 2$ steps and an initial **current state vector** where the VM starts in **state 1** (represented as $[1, 0, 0]$).
1. Calculate T^n, the transition matrix raised to the power of $n = 2$:

$$T^2 = T \times T = \begin{bmatrix} 0.392 & 0.277 & 0.331 \\ 0.536 & 0.099 & 0.365 \\ 0.467 & 0.578 & 0.0 \end{bmatrix}$$

2. Multiply T^2 by the initial state vector $[1, 0, 0]$:

$$\text{Next state probabilities} = \begin{bmatrix} 0.392 & 0.277 & 0.331 \\ 0.536 & 0.099 & 0.365 \\ 0.467 & 0.578 & 0.0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.392 \\ 0.536 \\ 0.467 \end{bmatrix}$$

3. Select the Next state: Since **state 2** has the highest probability (0.536), it is chosen as the next predicted state after looking ahead two steps.
If the look-ahead of n=10, it means we have to consider the behavior of next 10 steps before taking the migration decision.

5.3.2 Migration Decision Module

After calculating the next behavior the Migration Decision Module determines the initial migration method—pre-copy, post-copy, or hybrid. For that, the following four(4) conditions are used.

- If a VM is memory-intensive, it is migrated using **post-copy**
- For a network-intensive workload with mostly outgoing packets, the VM is migrated in **post-copy**

- If there are mostly incoming packets, they are migrated in **pre-copy.**
- All other VM workloads are migrated adapting the **hybrid method**.

If Pre-Copy or Hybrid is chosen, the algorithm monitors for high memory usage, or if certain limits are met, switches to post-copy if necessary. It means if there is a memory-intensive behavior in the predicted future state the migration decision should be altered as Post-Copy. Also, the hybrid migration method evaluates workload and iteration conditions to determine whether a switch to post-copy is beneficial. If there is a converging behavior, it will be identified using exponential decay, which shows that the memory dirty page rate has reduced for at least 3 consecutive intervals.

A summary of all the decision mappings is presented in the algorithm 2 clearly. A full diagram consisting of all 3 components and the decision-making flow is in the figure 4

Algorithm 2 Migration Decision Algorithm

```
1: Input:Look_ahead_States = $n$
2: Initialize: Current_Migration_Decision ← Get the migration decision based
   on the current State
3: Next_behavior ← Calculate the behavior of the upcoming $n$ States using future
   state predictor
4: if Current_Migration_Decision == post_copy then
5:     No switching during the migration
6: else if Current_Migration_Decision == pre_copy then
7:     if Next_behavior is memory-intensive then
8:         Start post_copy Migration
9:     else
10:        Start vanilla pre_copy
11:    end if
12: else if Current_Migration_Decision == Hybrid then
13:    if Next_behavior is memory-intensive then
14:        Start post_copy Migration
15:    else
16:        if Dirty Page rate shows an exponential decay then
17:            Switch to post_copy
18:        else
19:            Hybrid Migration
20:        end if
21:    end if
22: end if
```

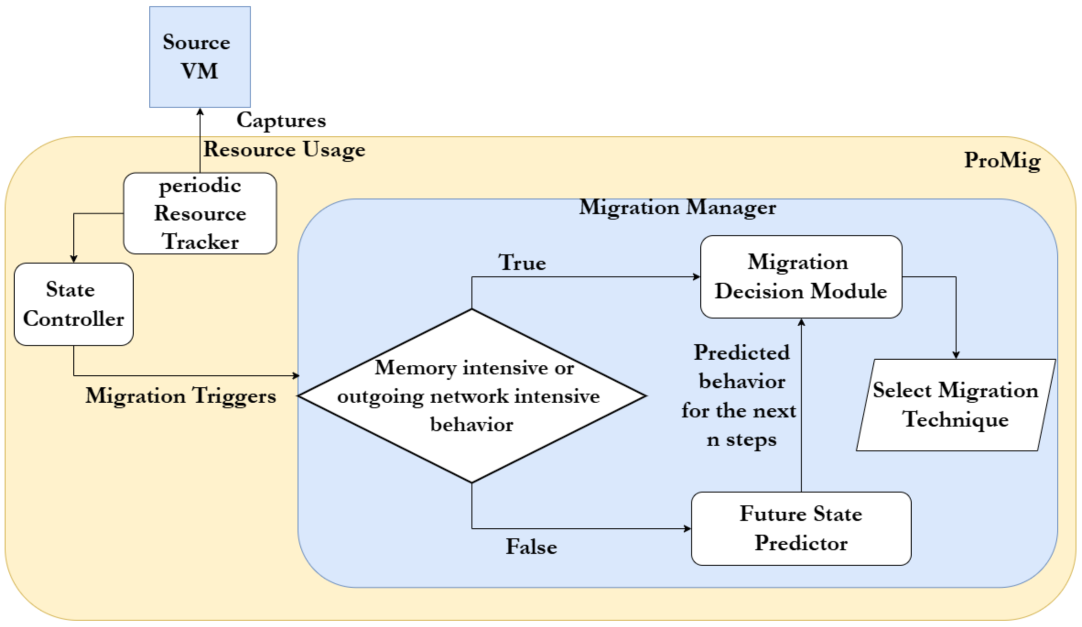
Source VM
Captures Resource Usage
ProMig
periodic Resource Tracker
Migration Manager
State Controller
True
Migration Decision Module
Predicted behavior for the next n steps
Select Migration Technique
Migration Triggers
Memory intensive or outgoing network intensive behavior
Future State Predictor
False

Figure 4: Components and Decision Flow of ProMig

6 Testing and Evaluation

6.1 Reaserch Tools Used

To conduct this research, a combination of virtualization, monitoring, and benchmarking tools was employed to simulate and analyze workload behavior.

QEMU: An open-source virtualization tool used to create and manage virtual machines (VMs). QEMU enabled flexible testing and migration of VMs under various workloads, which was essential for workload prediction experiments.

RealVNC: This remote access software facilitated real-time monitoring and control of VM performance during testing. RealVNC allowed for seamless interaction with VMs, enabling observation and data collection for workload analysis.

NFS (Network File System): NFS was utilized to set up shared storage across networked environments. This system facilitated resource sharing among VMs, which was necessary for testing workload migration and replication scenarios.

These tools collectively supported a comprehensive research setup, allowing for accurate workload simulation, monitoring, and analysis.

6.2 Evaluation Metrics

For the evaluation, several key matrices were used such as the total migration time(TMT), downtime (DT), and the application performance degradation. Explanation of these terms are as below.

- Total migration time- Duration between the start and end of migration
- Downtime - The amount of time VM will be unavailable during the migration process.
- Application performance degradation - The effect of the migration on the performance of migrating VM.

These factors was compared with the vanilla Pre-Copy, vanilla Post-Copy, and Hybrid approaches. For the experiments, a set of VMs with different sizes such as 1GB,2GB,4GB,8GB, and 12GB was used.

6.3 Evaluation Criteria

For the evaluation, three main categories of workloads were used: **persistent workloads, real-time workloads, and synthetic workloads**. The YCSB benchmark was used to generate persistent workloads, while the Stress benchmark was employed to create real-time workloads. For synthetic workloads, a combination of Workingset, Sysbench, and iPerf3 benchmarks was utilized to generate various dynamic workload scenarios, enabling the assessment of key use cases of ProMig. Each workload category included six different workloads, capturing diverse behavioral patterns for a comprehensive evaluation.

Each workload was migrated three to five times, and the average values were calculated after removing outliers. The results from each workload category were then compared against vanilla migration techniques to assess performance differences. This section provides a detailed explanation of the workloads used during the evaluation.

YCSB (Yahoo! Cloud Serving Benchmark): This benchmark was used to generate persistent workloads. YCSB is a popular tool for testing storage systems, and it provides a reliable method for creating stable, long-running workloads to simulate database and application server tasks. It provides the flexibility to generate workloads by changing the ratio between the read, insert, scan, delete, update, and readModifyWrite operations. Six(6) different workloads highlighting different behaviors were used for the evaluation

Stress: The Stress tool was employed to create real-time workloads. This benchmark generates high CPU, memory, and I/O usage on demand, which is ideal for simulating the resource-intensive and bursty demands characteristic of real-time applications. By providing different parameters to change the CPU and memory over time, 6 different workloads were generated.

Workingset: The Workingset benchmark was used to generate memory-intensive workloads by repeatedly selecting a memory block and modifying (dirtying) it from start to end. This process stresses memory resources, leading to high memory bandwidth consumption and increased page dirtying rates.

Sysbench: The Sysbench benchmark was used to generate CPU-intensive workloads by executing computationally heavy tasks, such as prime number calculations. This allowed for the evaluation of processor performance under varying levels of concurrency and complexity.

Iperf3: The iPerf3 benchmark was used to generate network-intensive workloads by measuring both incoming and outgoing network usage. It creates controlled network traffic to evaluate bandwidth, latency, and throughput under different load conditions.

Synthetic Workloads: A combination of benchmarks was executed sequentially to generate a synthetic workload, altering the workload behavior over time. Six(6) different workloads were generated, and the following notations were used for the naming.

Notation	Benchmark Used	Generated Behavior
C	Sysbench	CPU intensive behavior
M	Workingset	Memory intensive behavior
NI	iperf3	Incoming network intensive behavior
NO	iperf3	Outgoing network intensive behavior

Table 2: Benchmarks and their Generated Behavior

6.4 Stress Workloads Evaluation

In the stress benchmark, CPU and memory usage can be dynamically adjusted by modifying two key parameters, **–cpu** and **–vm**. The **–cpu N** option specifies the number of worker threads that generate computational load, directly affecting CPU utilization. Similarly, the **–vm M –vm-bytes SIZE** option controls memory stress by defining the number of memory workers (M) and the amount of memory allocated per worker. Six different workloads were generated using six(6) distinct parameter sets, and their downtime and total migration time were evaluated accordingly.

6.4.1 Total Migration Time

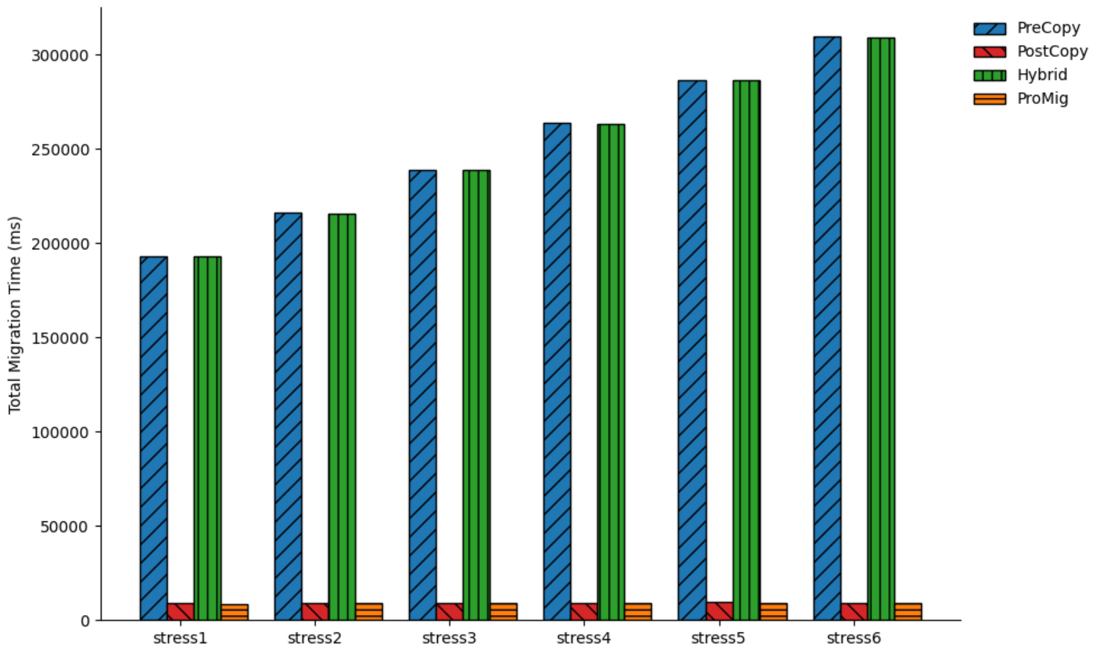

Figure 5: Total Migration Time for Stress Workloads- 1GB

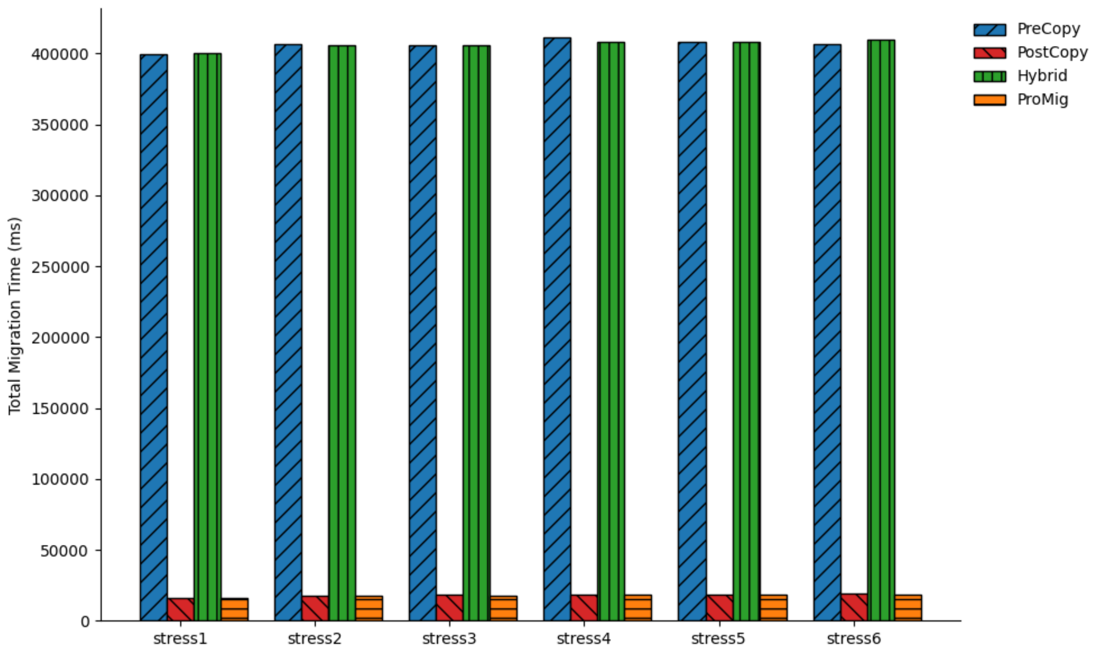

Figure 6: Total Migration Time for Stress Workloads- 2GB

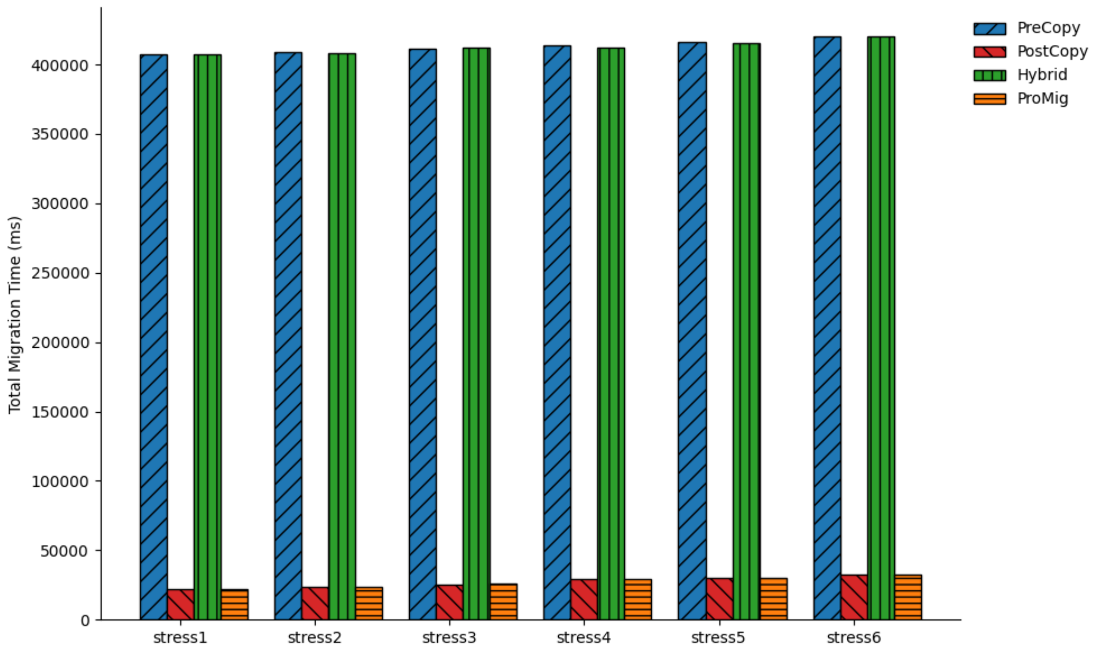

Figure 7: Total Migration Time for Stress Workloads- 4GB

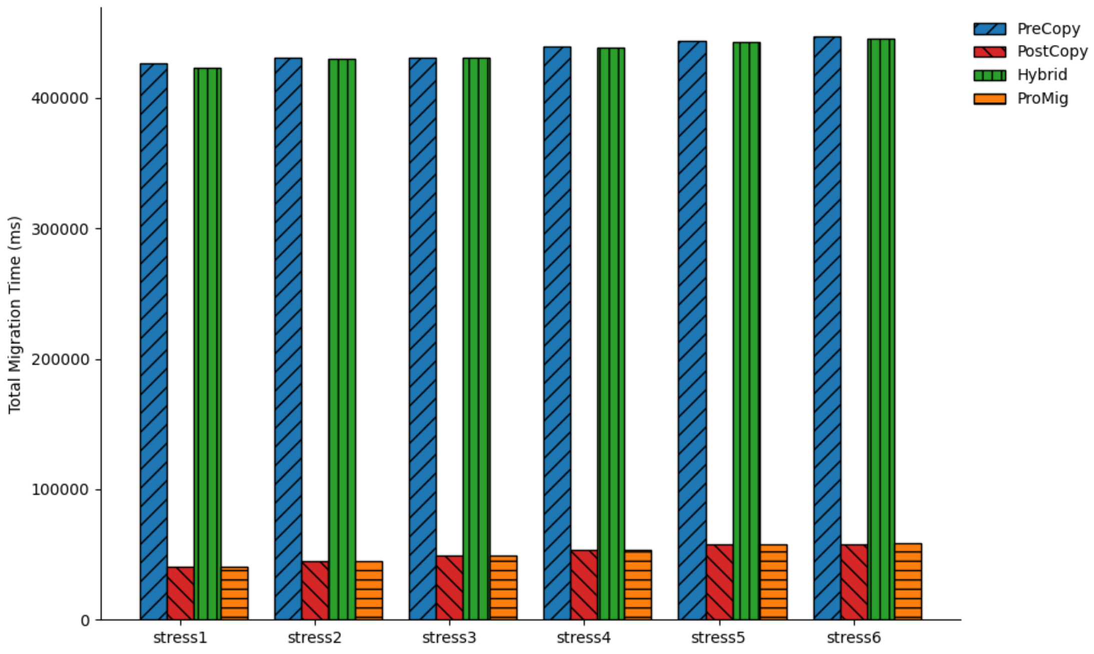

Figure 8: Total Migration Time for Stress Workloads- 8GB

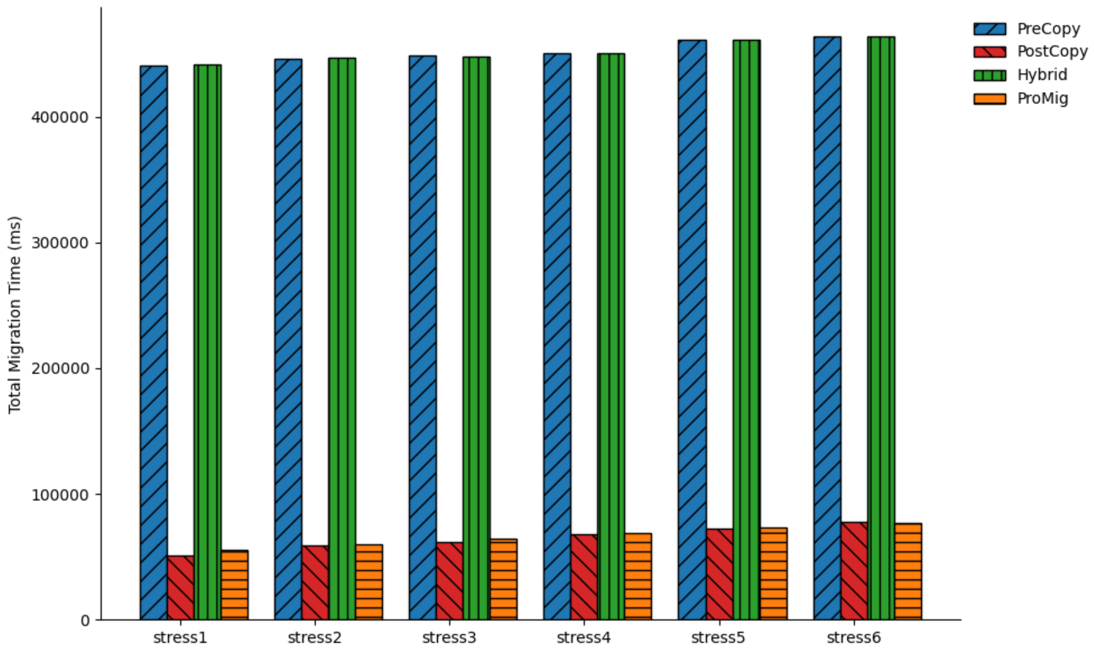

Figure 9: Total Migration Time for Stress Workloads- 12GB

According to the graphs, it is clear that the stress workloads have the lowest TMT with the Post-Copy migration technique, and ProMig selects Post-Copy as the most suitable migration technique at all times.

6.4.2 Donwtime

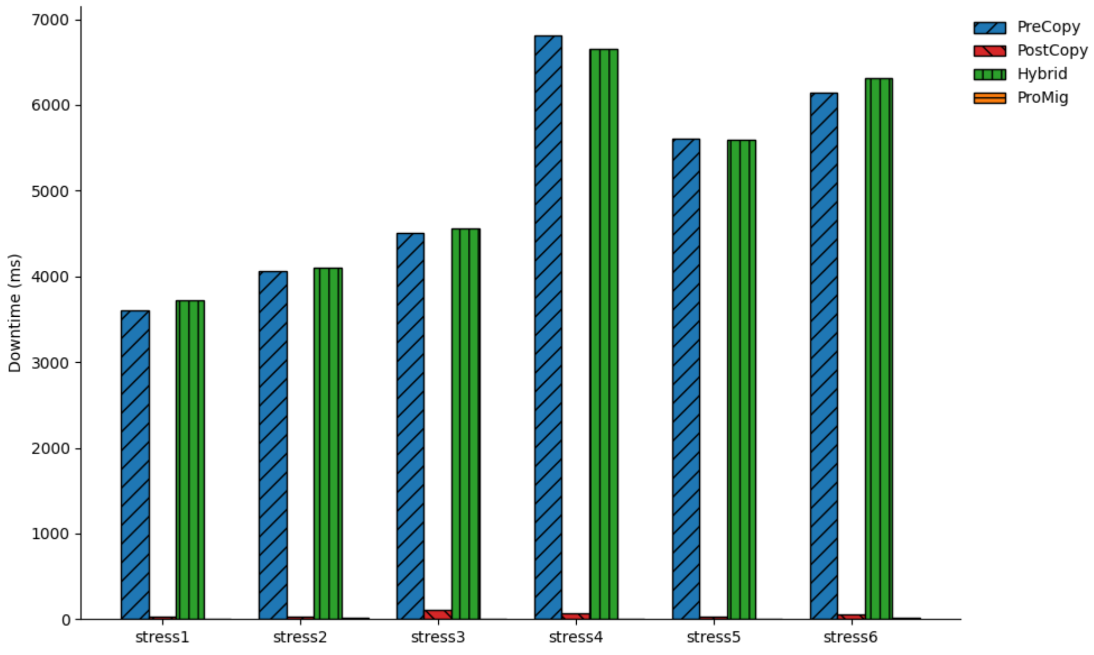

Figure 10: Downtime for Stress Workloads- 1GB

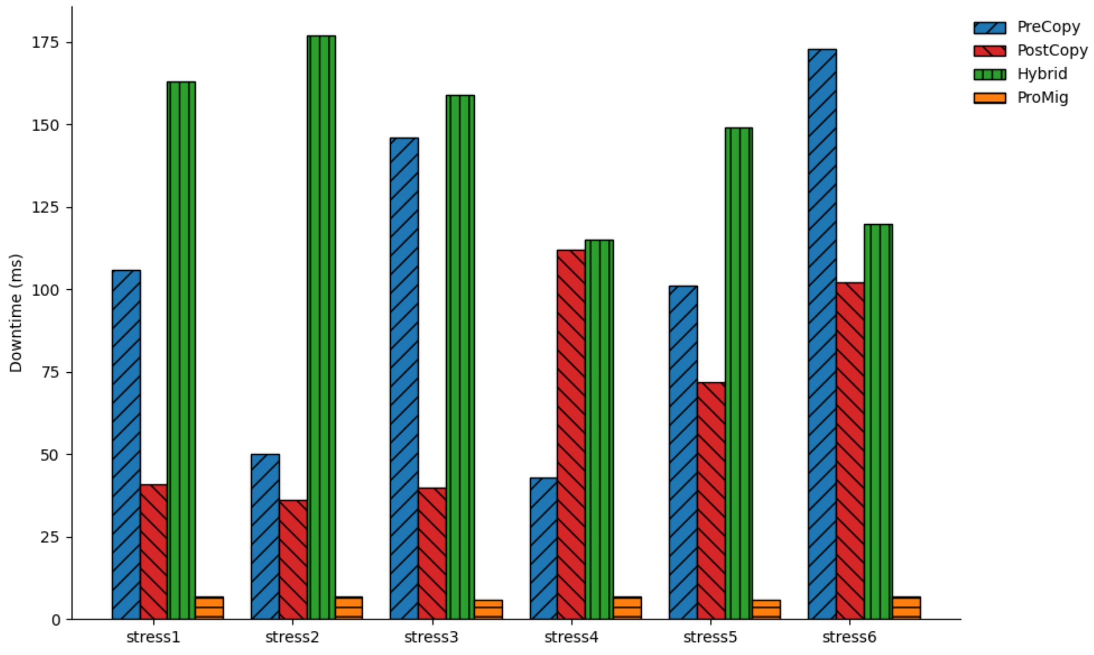

Figure 11: Downtime for Stress Workloads- 2GB

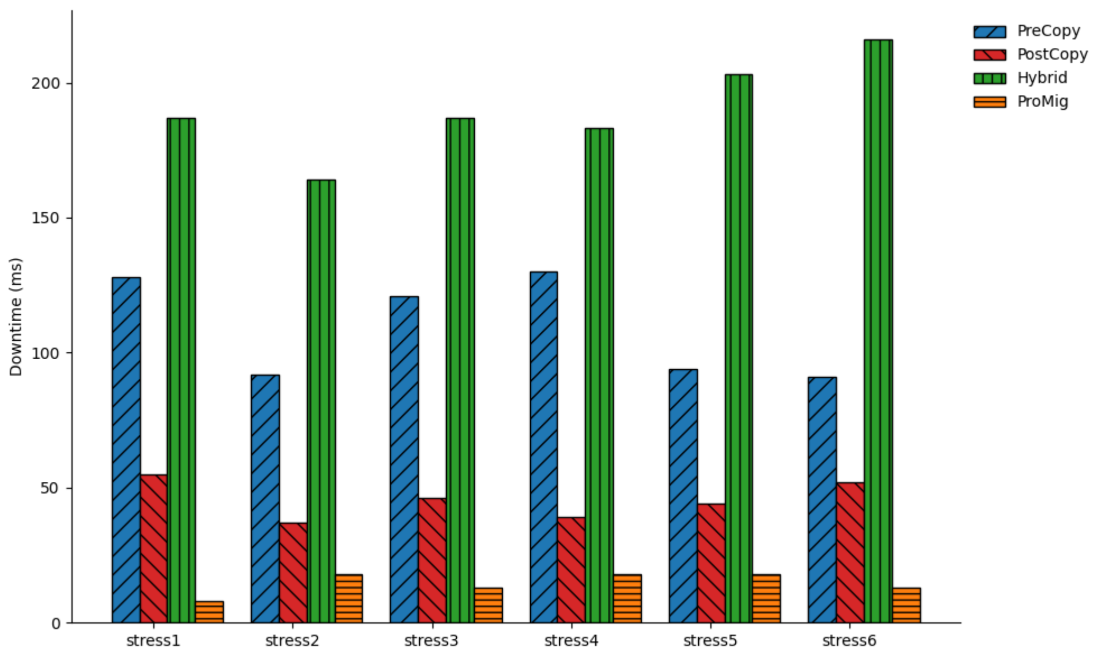

Figure 12: Downtime for Stress Workloads- 4GB

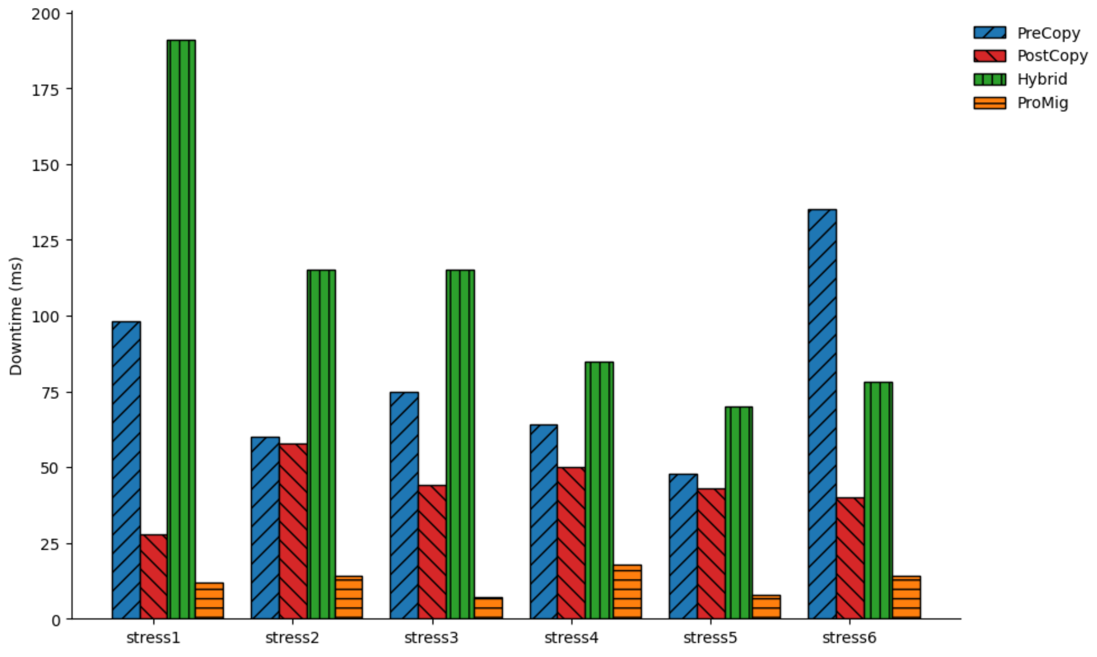

Figure 13: Downtime for Stress Workloads- 8GB

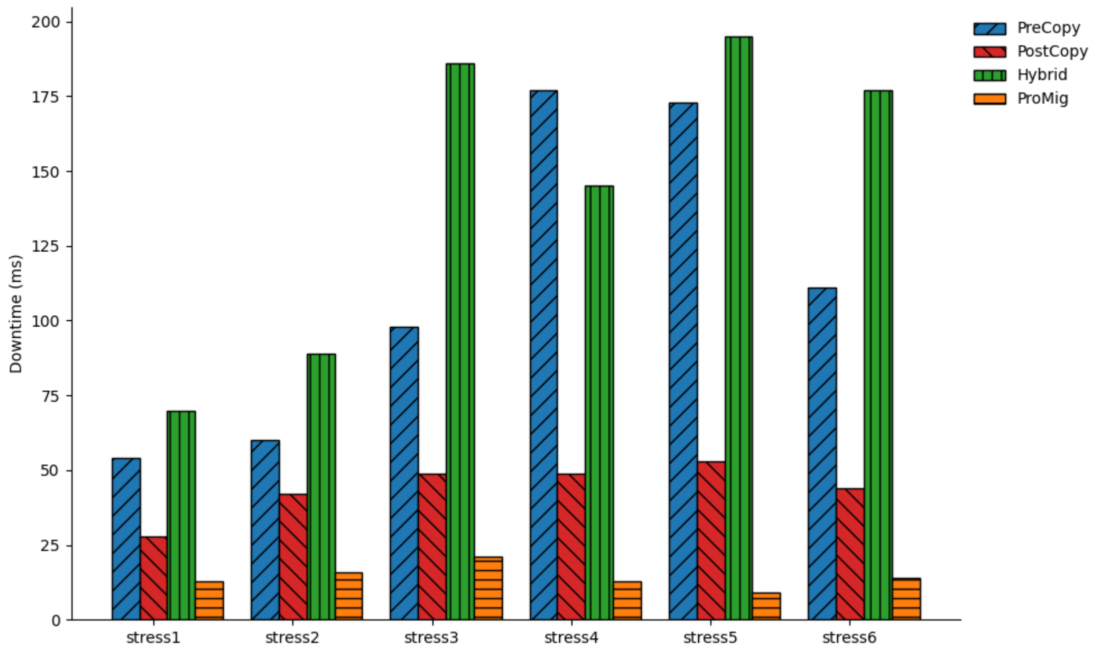

Figure 14: Downtime for Stress Workloads- 12GB

According to the graphs, the downtime of ProMig is significantly lower than that of the Pre-Copy and Hybrid techniques, making it nearly negligible in comparison. Across all six workloads, ProMig consistently selects the optimal migration technique, achieving the lowest total migration time and downtime. Compared to the Pre-Copy and Hybrid approaches, ProMig demonstrates an improvement of over 80%.

6.5 Synthetic Workloads

Six different synthetic workloads, each highlighting a specific behavior, were generated using the Workingset, Sysbench, and iPerf3 benchmarks. The notation used for naming follows the convention described in Table 2. The selected workloads are as follows. Each workload is migrated at the middle of the execution of the third benchmark.

Notation	**Benchmark Execution Order**	**Description**
NO/M/NI/M	iperf3-outgoing, workingset, iperf3-incoming, workingset	• Migration starts during iperf3-incoming. • Pre-Copy seems suitable initially, but due to upcoming memory-intensive behavior, Post-Copy is preferred.
NO/C/NI/C	iperf3-outgoing, sysbench, iperf3-incoming, sysbench	• Migration starts during iperf3-incoming. • Since the upcoming behavior is not memory-intensive, Pre-Copy gives the best total migration time and downtime.
NI/NO/N/C	iperf3-incoming, iperf3-outgoing, workingset, sysbench	• Migration starts during the execution of workingset. • This leads to a Post-Copy decision.
NI/M/NO/C	iperf3-incoming, workingset, iperf3-outgoing, sysbench	• Migration starts during iperf3-outgoing execution. • This results in Post-Copy selection.
NO/M/C/NI	iperf3-outgoing, workingset, sysbench, iperf3-incoming	• Migration starts during sysbench execution. • As Workingset causes memory-intensive behavior later, Post-Copy is preferred.
NO/NI/C/NI	iperf3-outgoing, iperf3-incoming, sysbench, iperf3-incoming	• Migration starts during sysbench execution. • With no upcoming memory-intensive behavior, the Hybrid approach works best.

Table 3: Synthetic Workloads Description

6.5.1 Total Migration Time

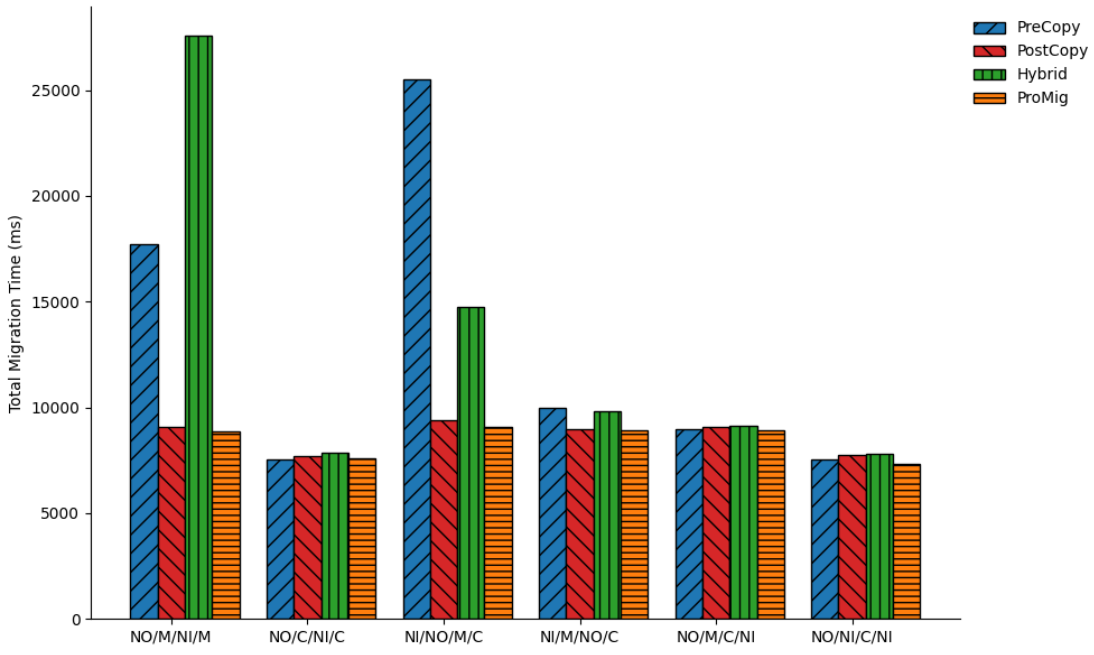

Figure 15: Total Migration Time for Synthetic Workloads- 1GB

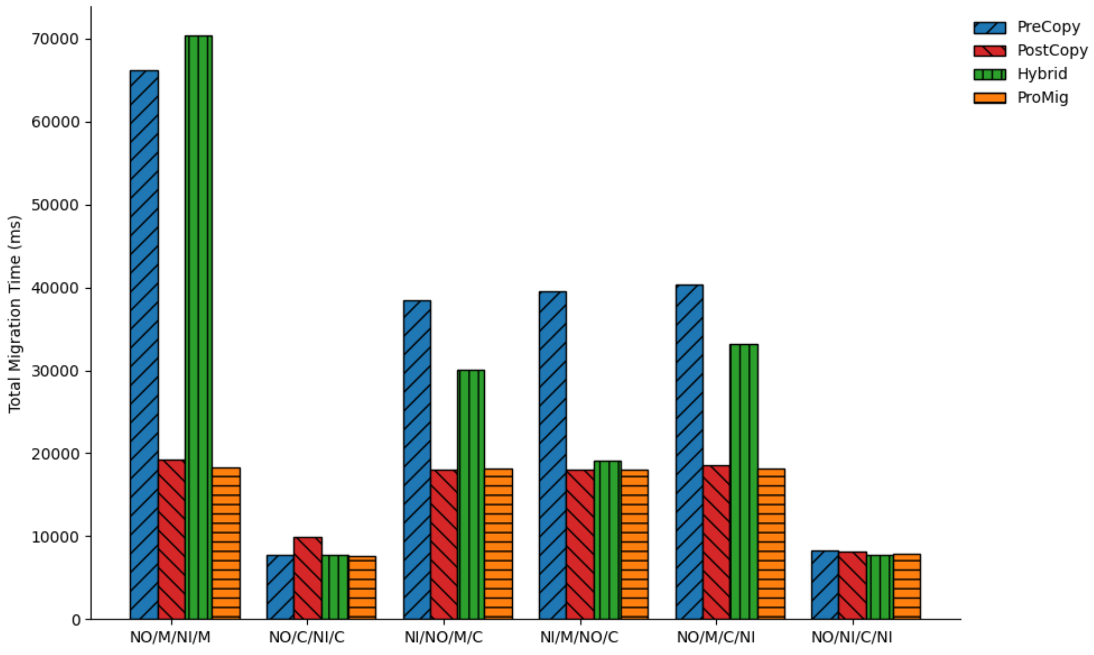

Figure 16: Total Migration Time for Synthetic Workloads- 2GB

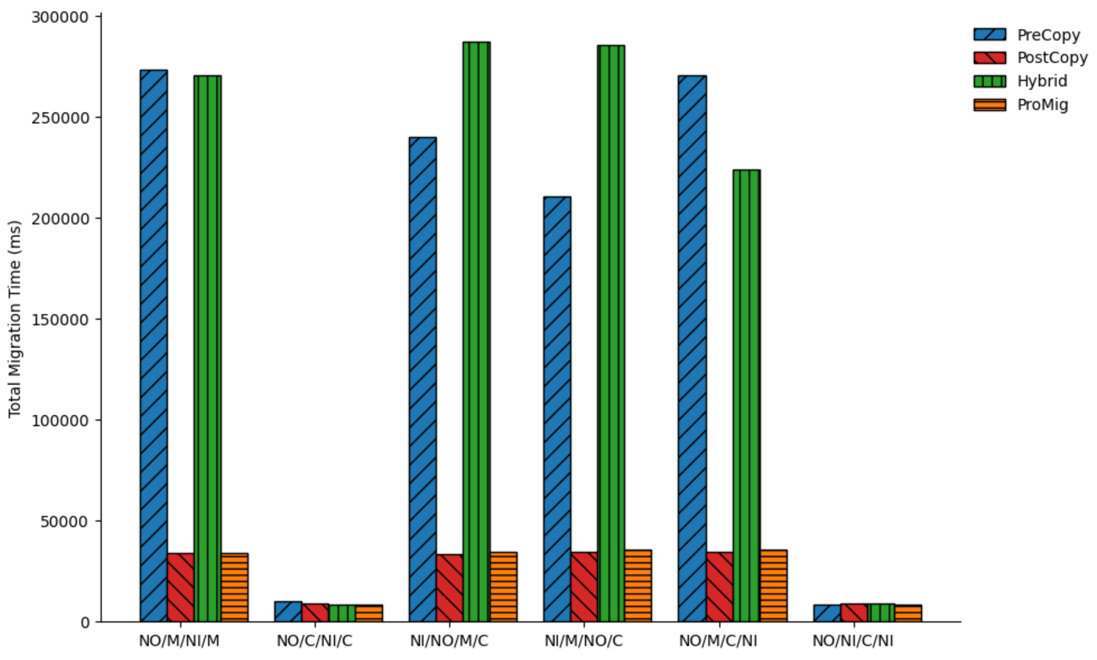

Figure 17: Total Migration Time for Synthetic Workloads- 4GB

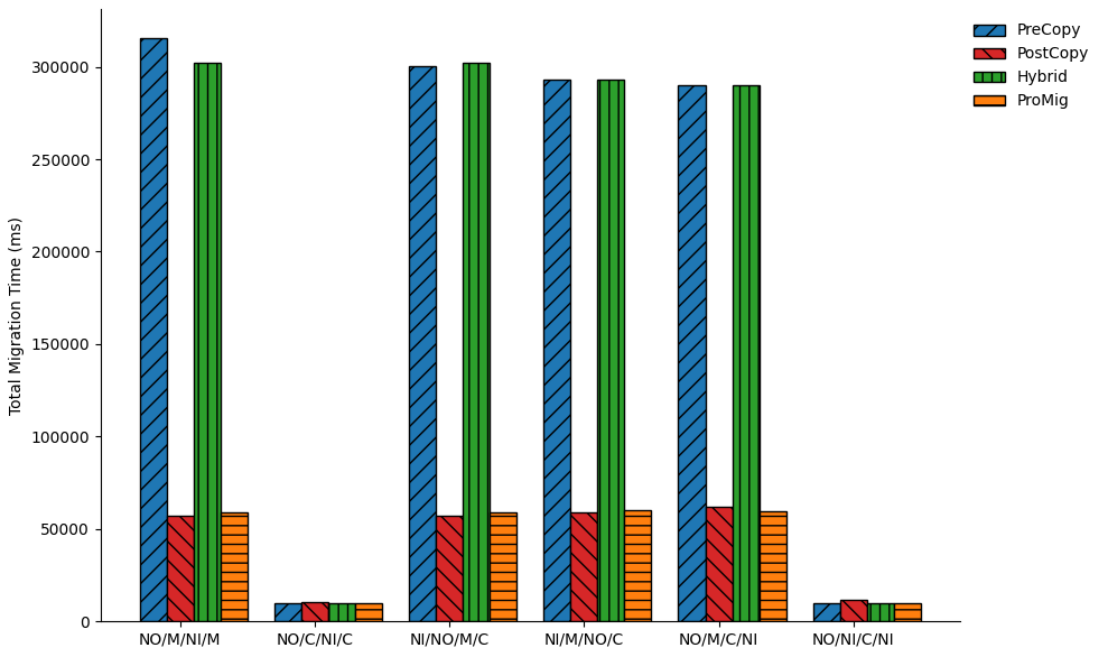

Figure 18: Total Migration Time for Synthetic Workloads- 8GB

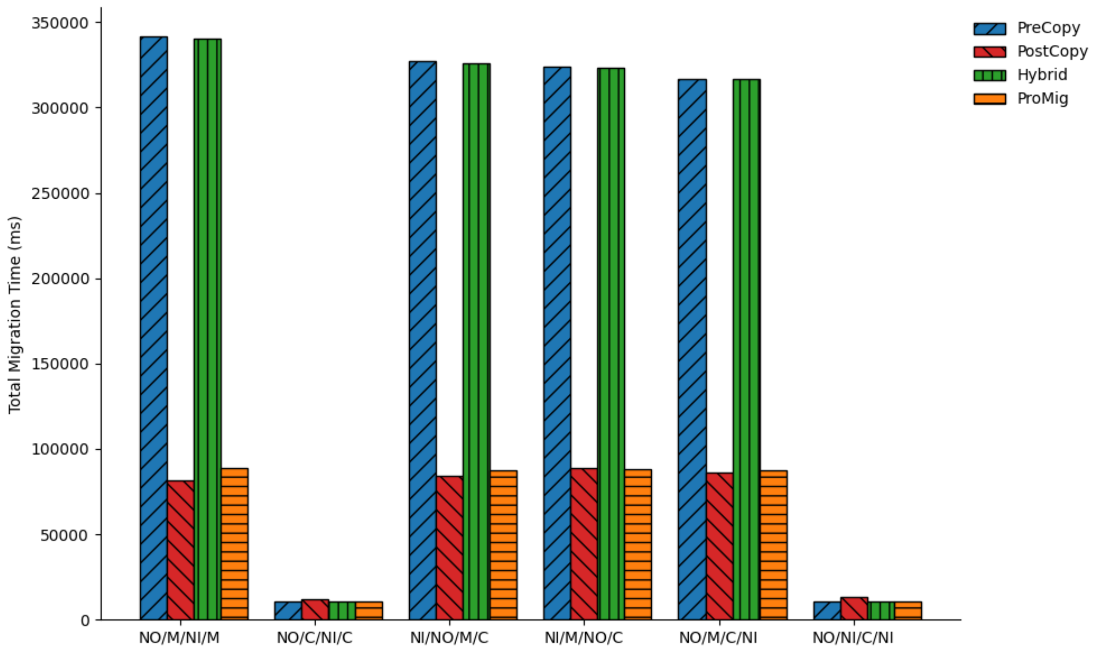

Figure 19: Total Migration Time for Synthetic Workloads- 12GB

According to the graphs, it is evident that these six different workload behaviors select different migration techniques based on their nature. Some behaviors, such as **NO/M/NI/M** and **NO/M/C/NI**, exhibit upcoming memory spikes that cannot be predicted solely by analyzing the current behavior. Since Promig includes a future state predictor to anticipate workload behavior, it consistently selects the optimal migration technique by considering both the current and future states.

6.5.2 Donwtime

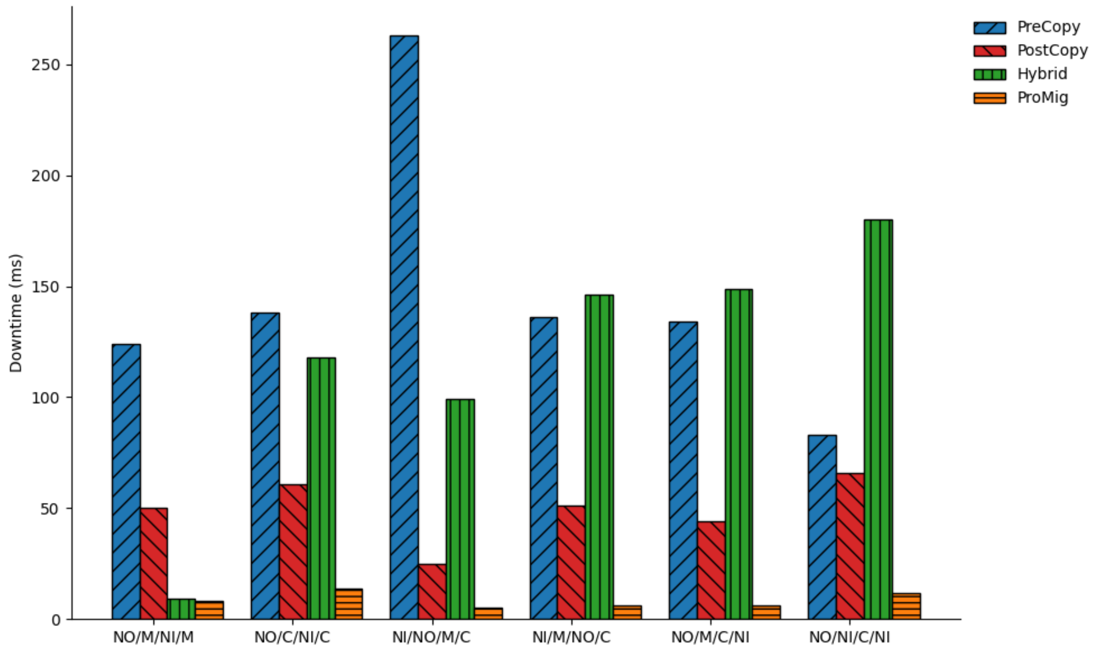

Figure 20: Downtime for Synthetic Workloads- 1GB

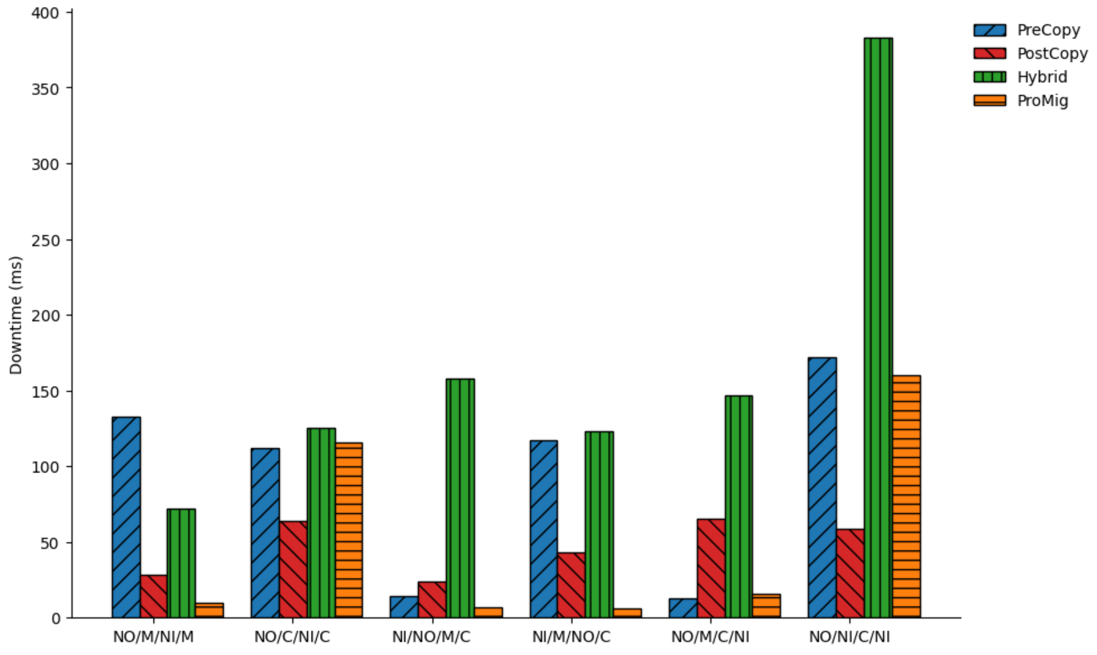

Figure 21: Downtime for Synthetic Workloads- 2GB

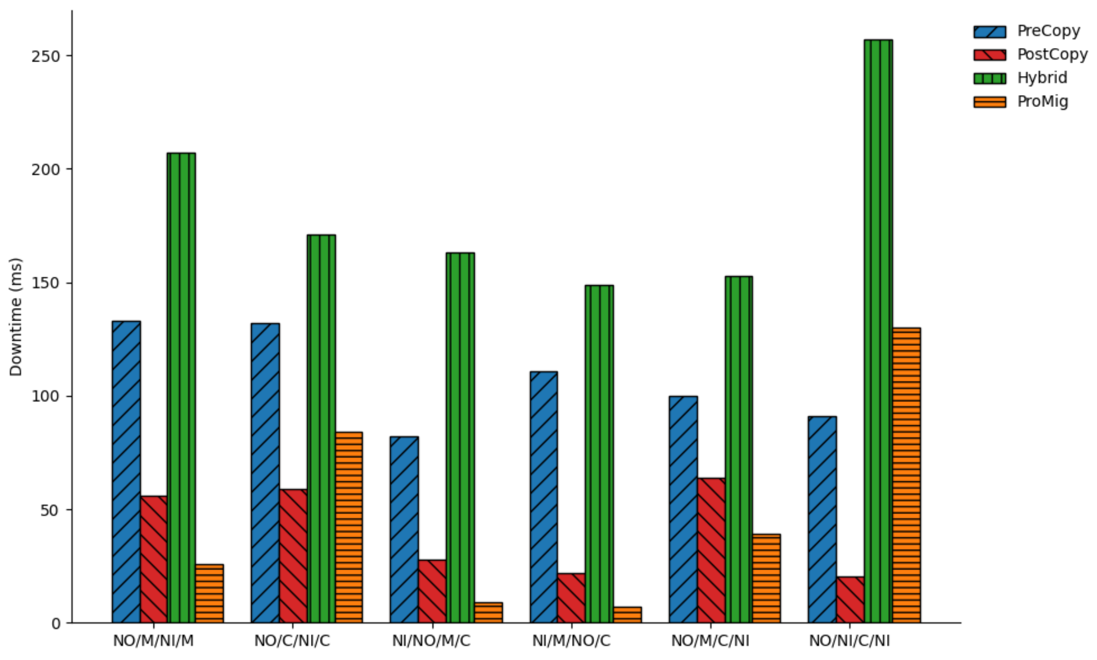

Figure 22: Downtime for Synthetic Workloads- 4GB

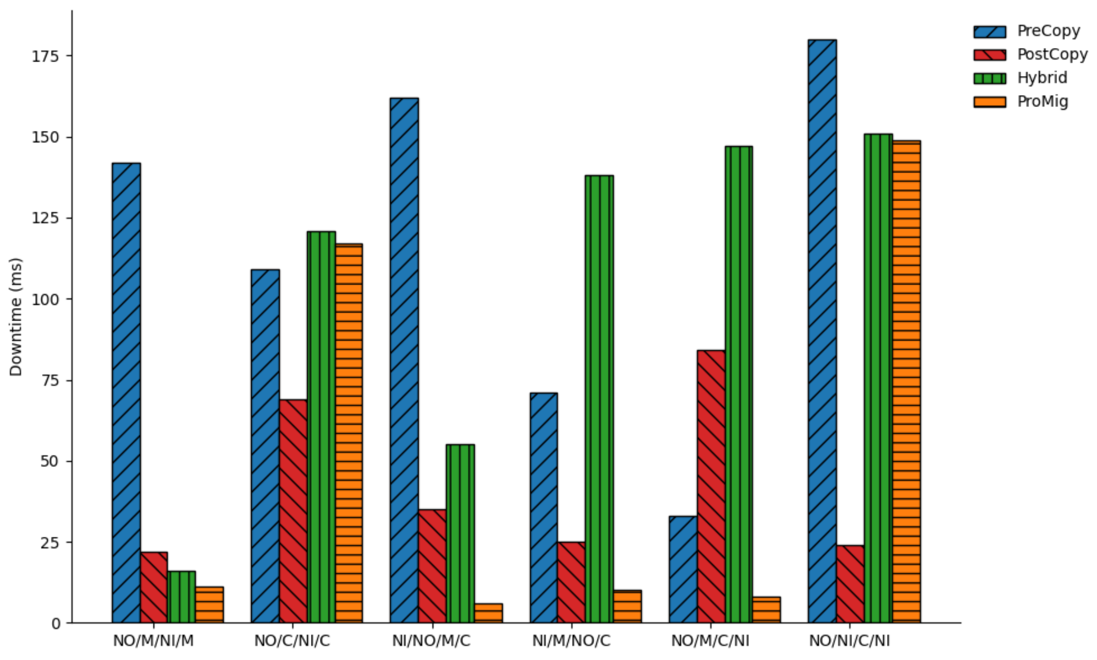

Figure 23: Downtime for Synthetic Workloads- 8GB

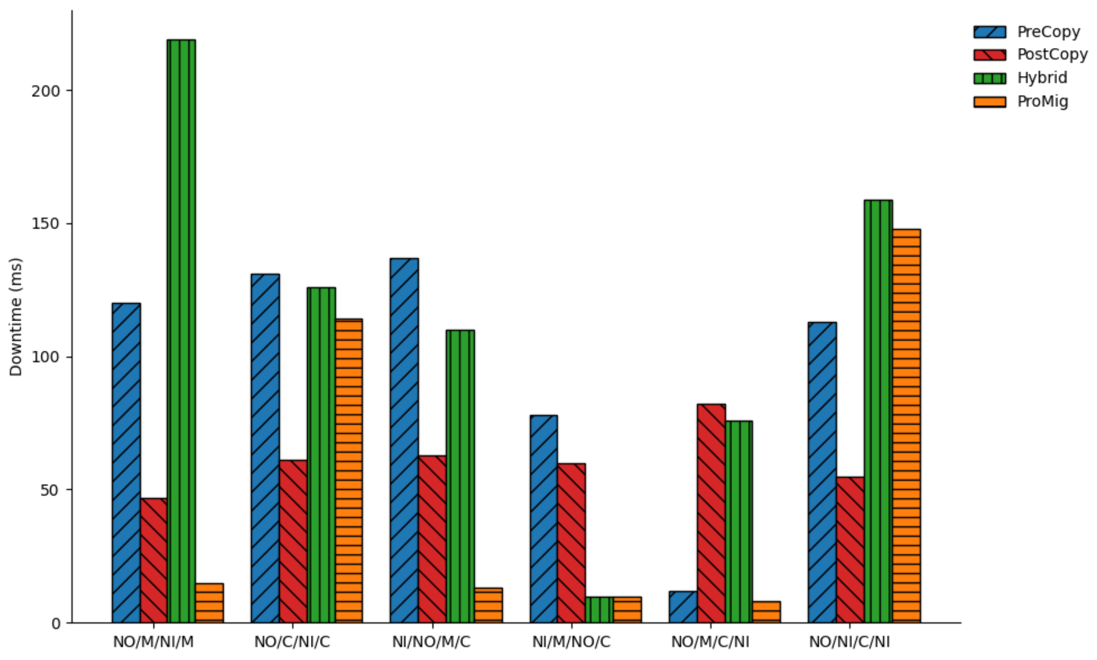

Figure 24: Downtime for Synthetic Workloads- 12GB

When analyzing downtime, the post-copy technique presents a unique case. In post-copy migration, downtime is defined as the period between stopping the source VM, transferring a minimal number of pages to the destination, and starting the destination VM. Since only a small number of pages are sent before resuming execution, post-copy achieves significantly lower downtime compared to pre-copy and hybrid approaches.
Since ProMig focuses on reducing both total migration time (TMT) and downtime (DT), it selects the most suitable migration technique based on the workload characteristics. In some cases, this requires choosing pre-copy or hybrid, which involve transferring a large number of pages during the stop-and-copy phase before starting the destination VM. As a result, ProMig may sometimes experience higher downtime compared to post-copy alone. However, when compared to other migration techniques, its downtime remains unchanged or is even reduced. This demonstrates that ProMig effectively balances TMT and DT, ensuring an optimized migration process.

6.6 YCSB Workloads

YCSB falls under the category of persistent workloads and provides the flexibility to adjust the ratio between six different operations: read, write, scan, update, delete, and read-modify-write. By varying these ratios, six distinct workload types were generated: **read-heavy, read-modify-write, insert-heavy, scan-heavy, update-heavy, and delete-heavy**. Total migration time and downtime evaluation for YCSB workloads are as below.

6.6.1 Total Migration Time

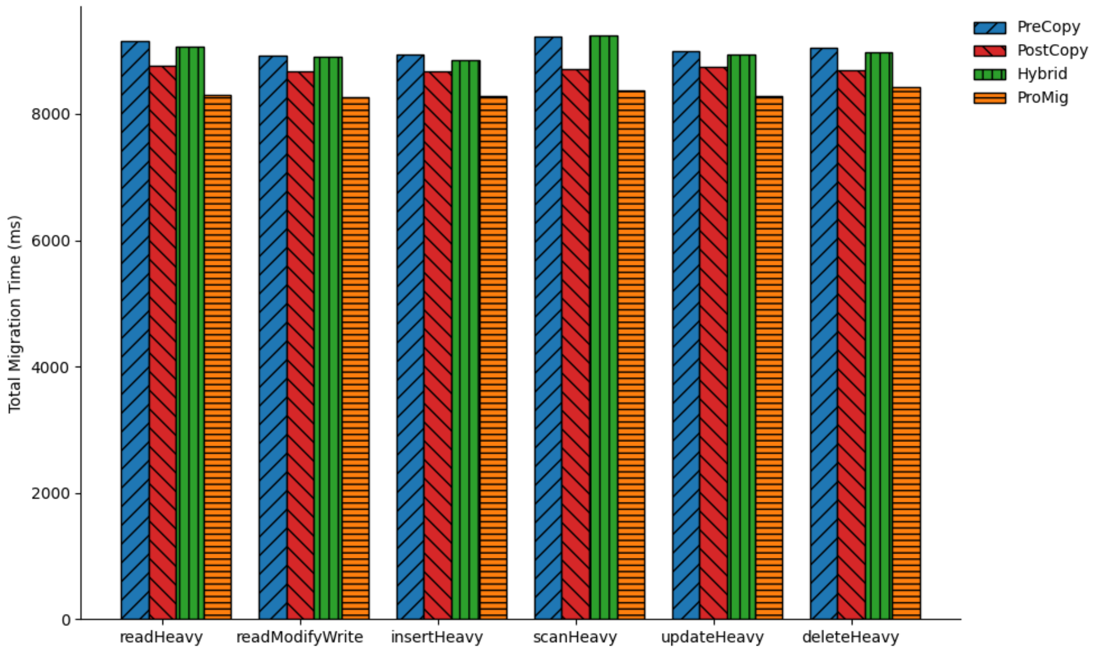

Figure 25: Total Migration Time for YCSB Workloads- 1GB

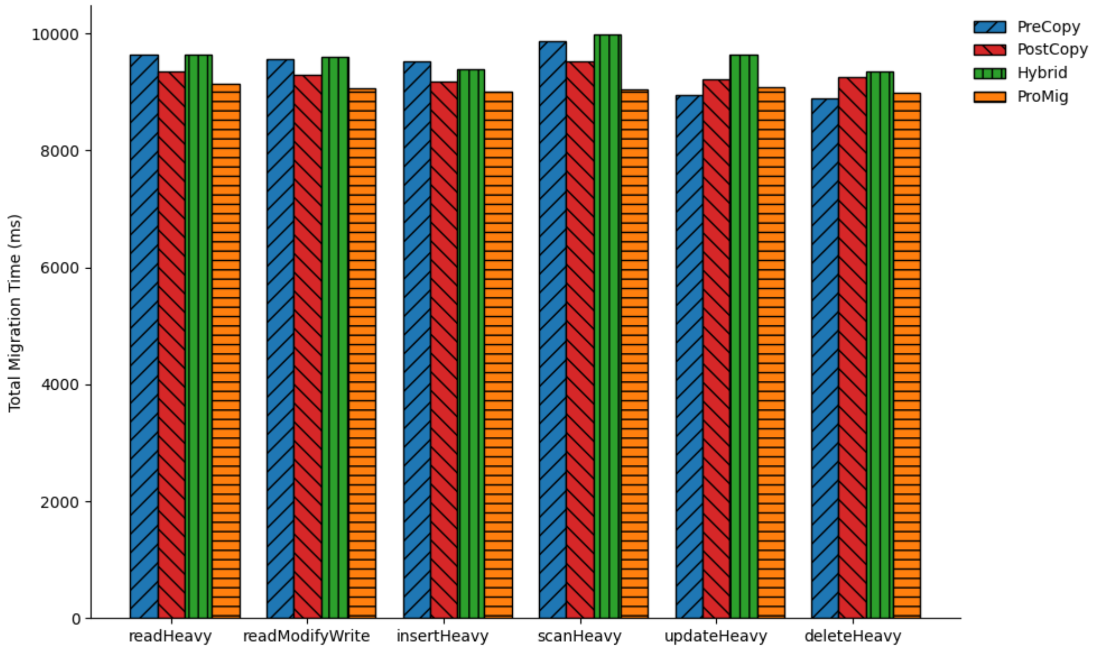

Figure 26: Total Migration Time for YCSB Workloads- 2GB

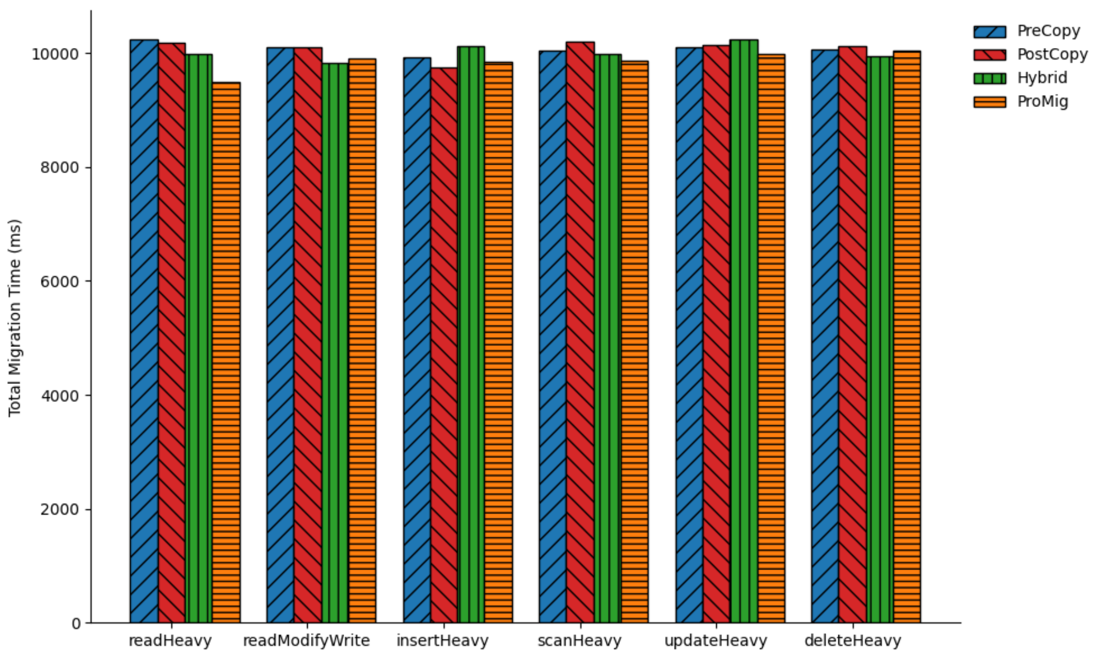

Figure 27: Total Migration Time for YCSB Workloads- 4GB

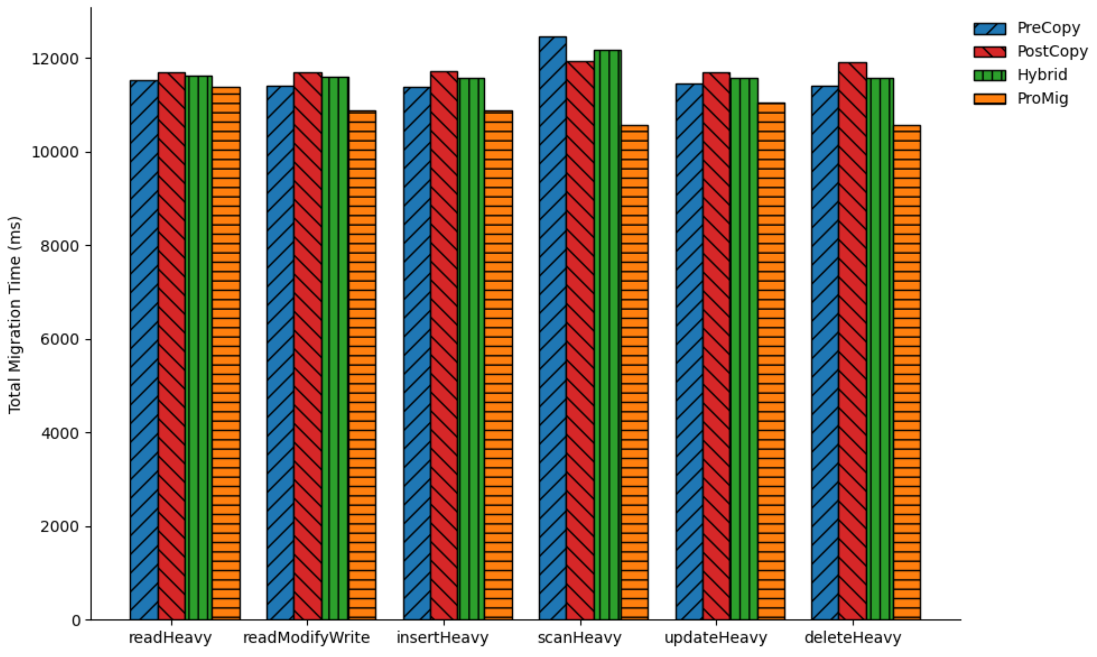

Figure 28: Total Migration Time for YCSB Workloads- 8GB

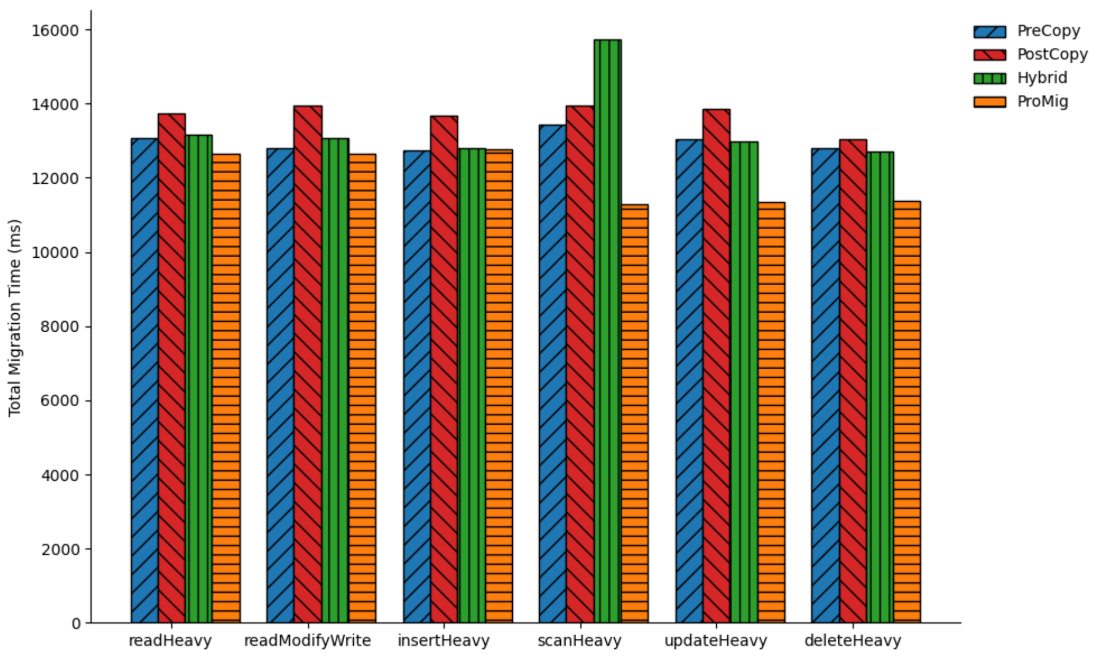

Figure 29: Total Migration Time for YCSB Workloads- 12GB

Similar to previous workloads, ProMig consistently selects the optimal migration technique, resulting in the lowest total migration time.

An important consideration is that for **1GB and 2GB VMs, post-copy performs bes**t, whereas for **larger VM sizes(4GB, 8GB and 12 GB), the hybrid migration technique yields better results** with these workloads. In both cases, ProMig makes the correct migration decision. This is because ProMig selects the migration technique based on resource usage percentage rather than absolute values. As VM size increases, the same workload exhibits different characteristics, **for smaller VMs, it behaves as memory-intensive, while for larger VMs, it shows lower relative memory usage**. Since ProMig relies on usage percentages, it effectively captures these variations and selects the most suitable migration technique for both scenarios.

6.6.2 Donwtime

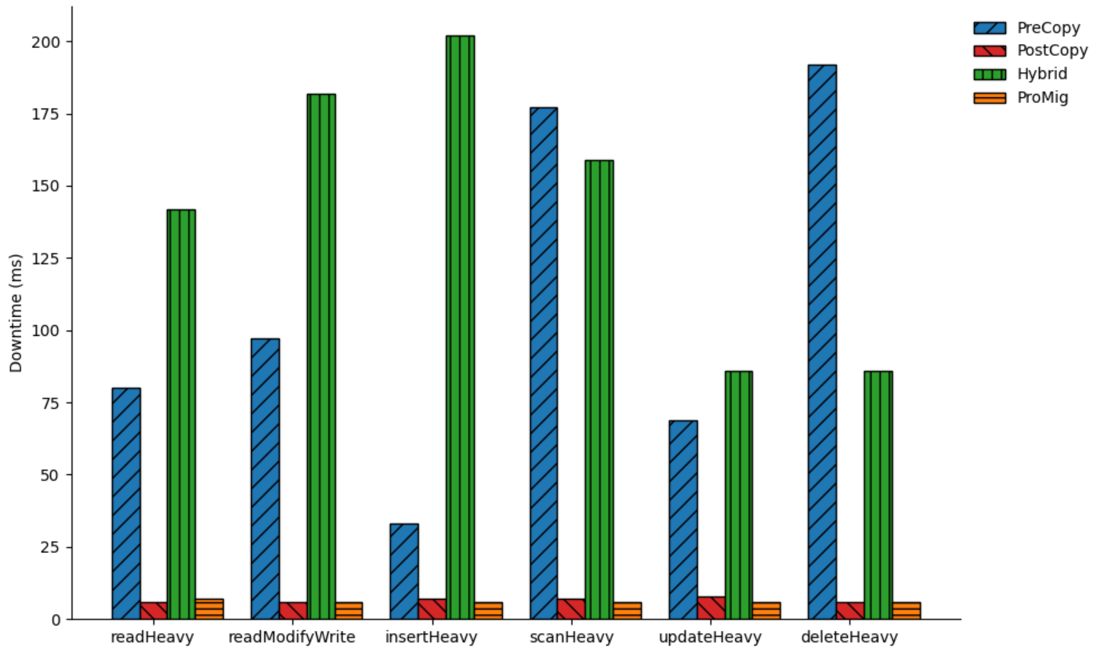

Figure 30: Downtime for YCSB Workloads- 1GB

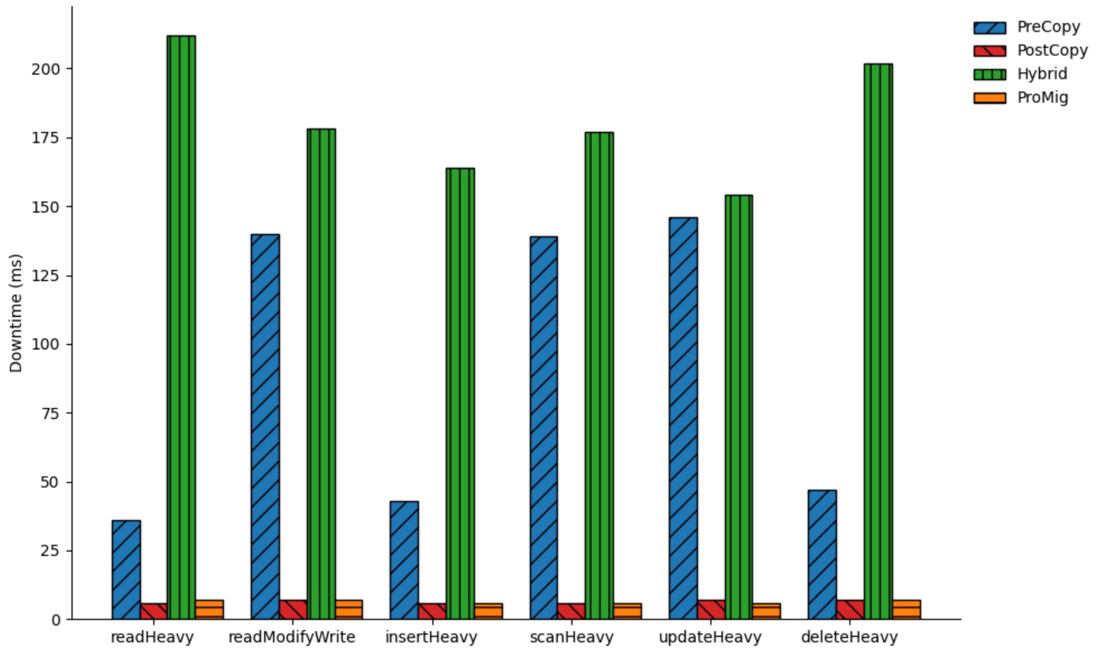

Figure 31: Downtime for YCSB Workloads- 2GB

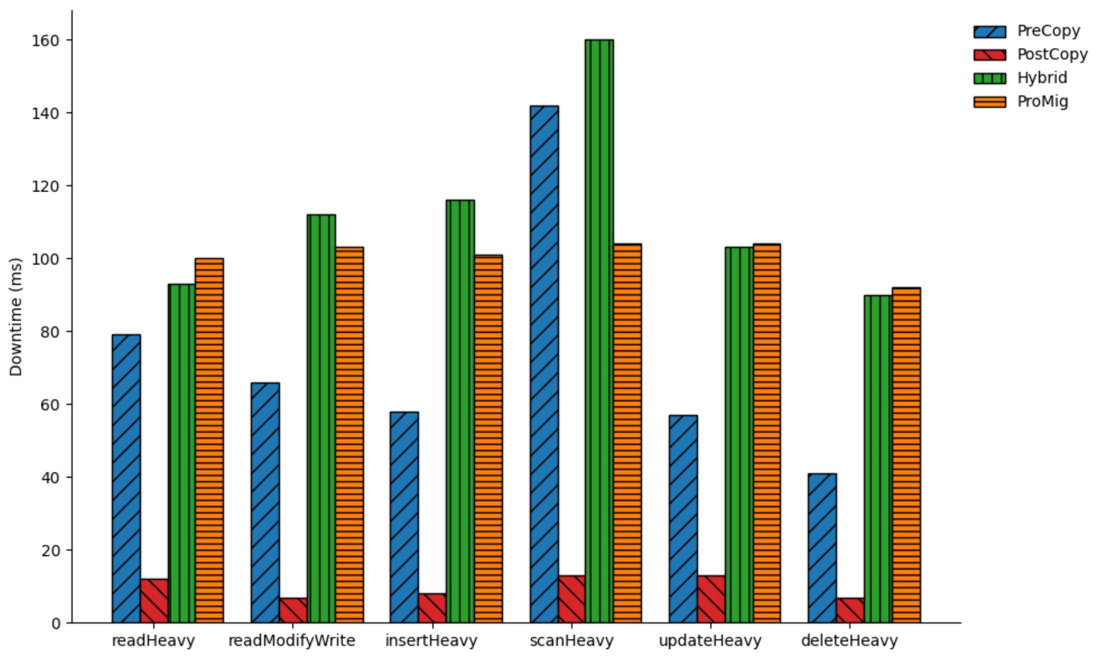

Figure 32: Downtime for YCSB Workloads- 4GB

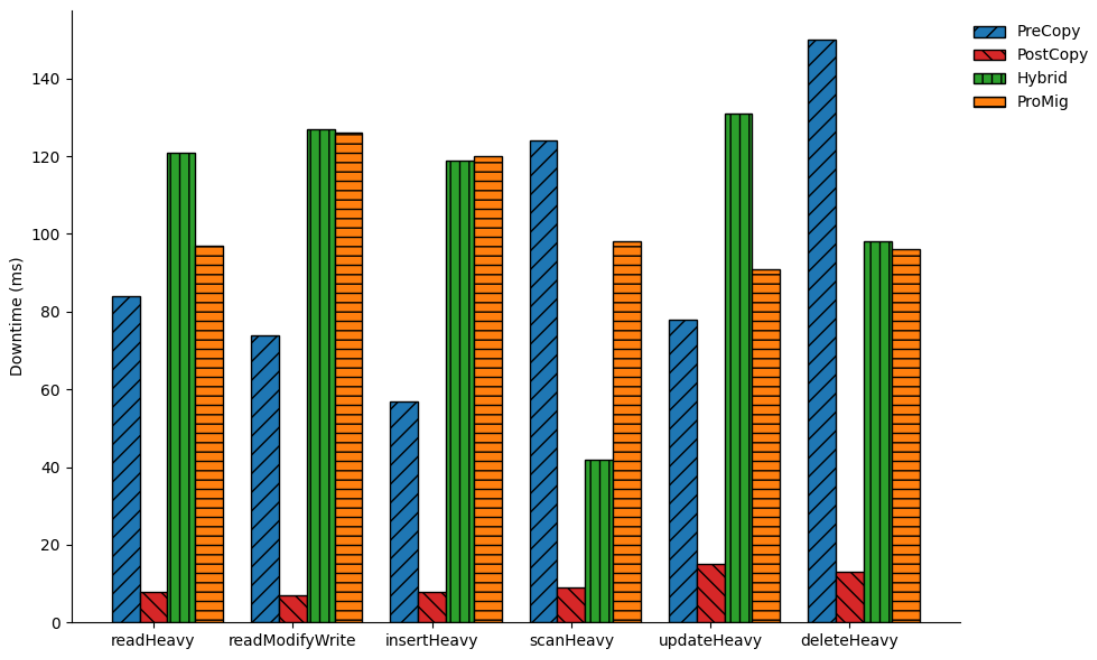

Figure 33: Downtime for YCSB Workloads- 8GB

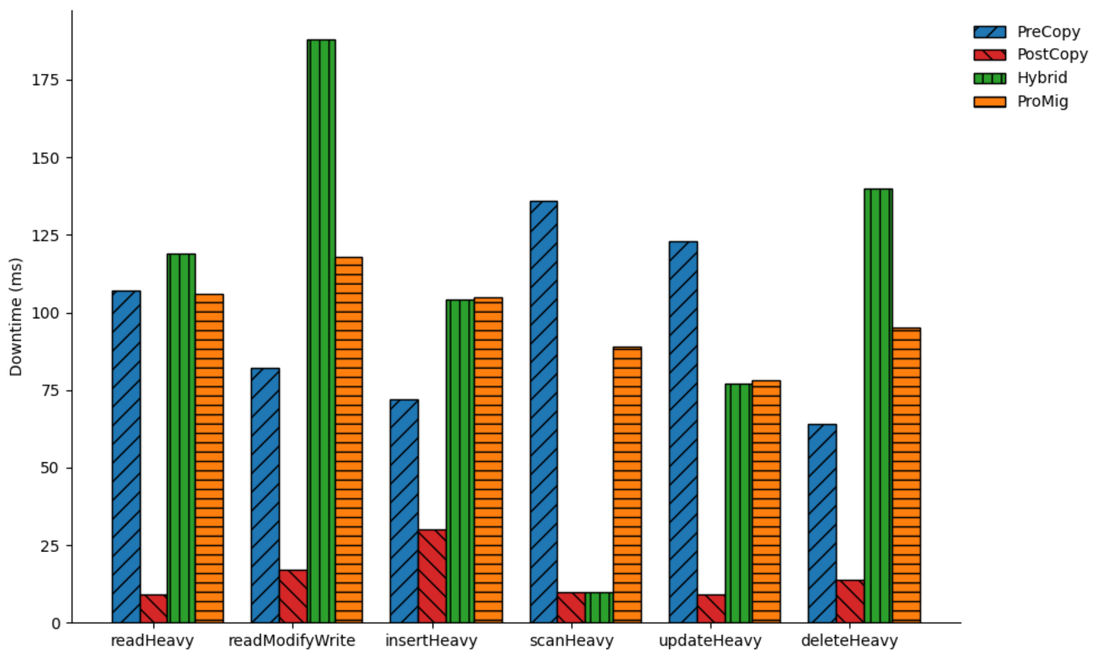

Figure 34: Downtime for YCSB Workloads- 12GB

Similar to synthetic workloads, the downtime of YCSB workloads is higher compared to post-copy because post-copy resumes execution with minimal data transfer before fetching the remaining pages on demand. However, when compared to pre-copy and hybrid techniques, the downtime remains unchanged, as both pre-copy and hybrid methods transfer a significant number of pages before starting the destination VM, leading to similar downtime behavior.

However, in certain cases, ProMig selects the hybrid approach and achieves lower downtime compared to standard hybrid migration. This is because ProMig detects **exponential decay behavior** in workload patterns, ensuring that migration occurs when the workload is approaching a convergence point with a decreasing dirty page rate. By initiating the stop-and-copy phase at this optimal moment, ProMig effectively reduces downtime while maintaining efficient migration performance.

Across all three workload categories—stress workloads, synthetic workloads, and YCSB workloads—ProMig consistently selects the optimal migration technique based on workload characteristics. For stress workloads, which generate high memory pressure and dirty page rates, ProMig effectively balances total migration time (TMT) and downtime (DT) by adapting to the changing workload intensity. For synthetic workloads, it accurately captures workload variations and selects the best technique, ensuring minimal TMT while maintaining downtime comparable to pre-copy and hybrid approaches. For YCSB workloads, ProMig recognizes workload-specific behaviors and accounts for VM size variations, leading to efficient migration decisions. In all cases, ProMig demonstrates its ability to dynamically adjust to workload demands, reducing TMT while maintaining or even improving downtime compared to conventional migration techniques.

6.7 Prediction Time

The time taken for future state prediction was measured, and as a percentage, it remains significantly lower compared to the total migration time. However, it increases as the number of prediction steps increases.
The total migration time varies depending on the size of the VM; as the VM size increases, the total migration time also increases. Since each prediction step estimates the workload behavior for the next second, a larger VM requires more prediction steps to cover the extended migration duration.
The variation in total migration time is illustrated in Graph 35, while the variation in prediction time is shown in Graph 36. According to the graph it is clear that the prediction time is less that 1% of the total migration time.

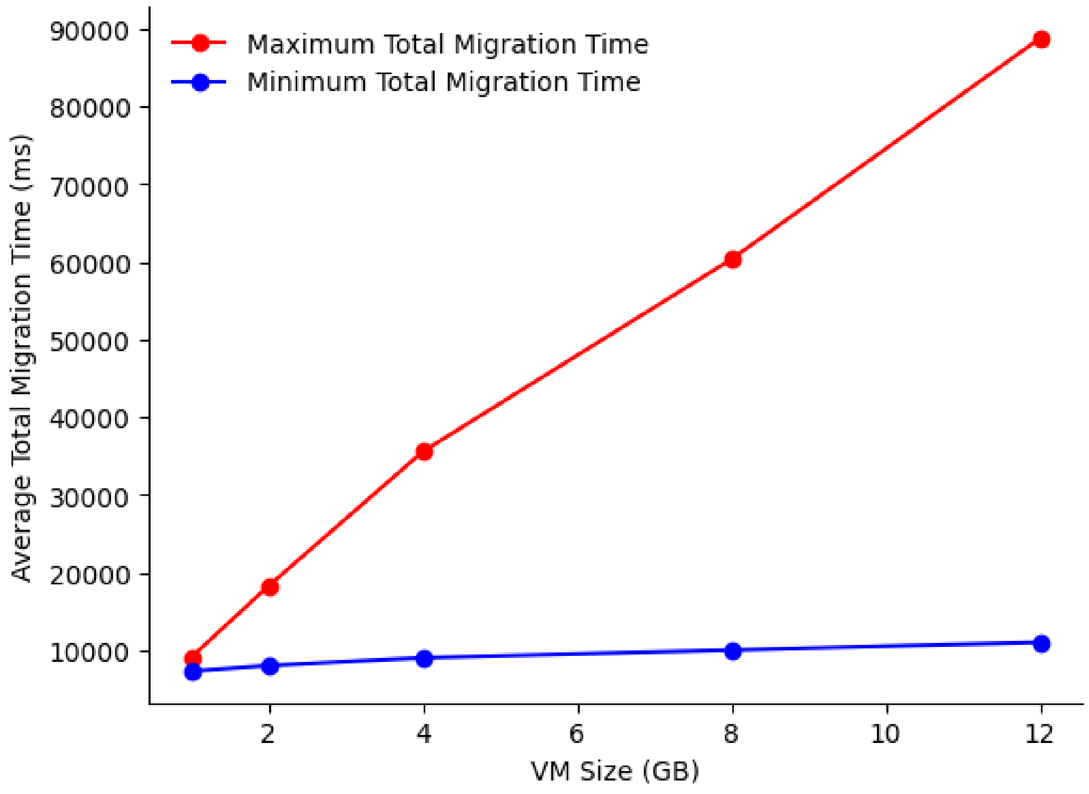

Figure 35: Total Migration Time Variation with VM Size

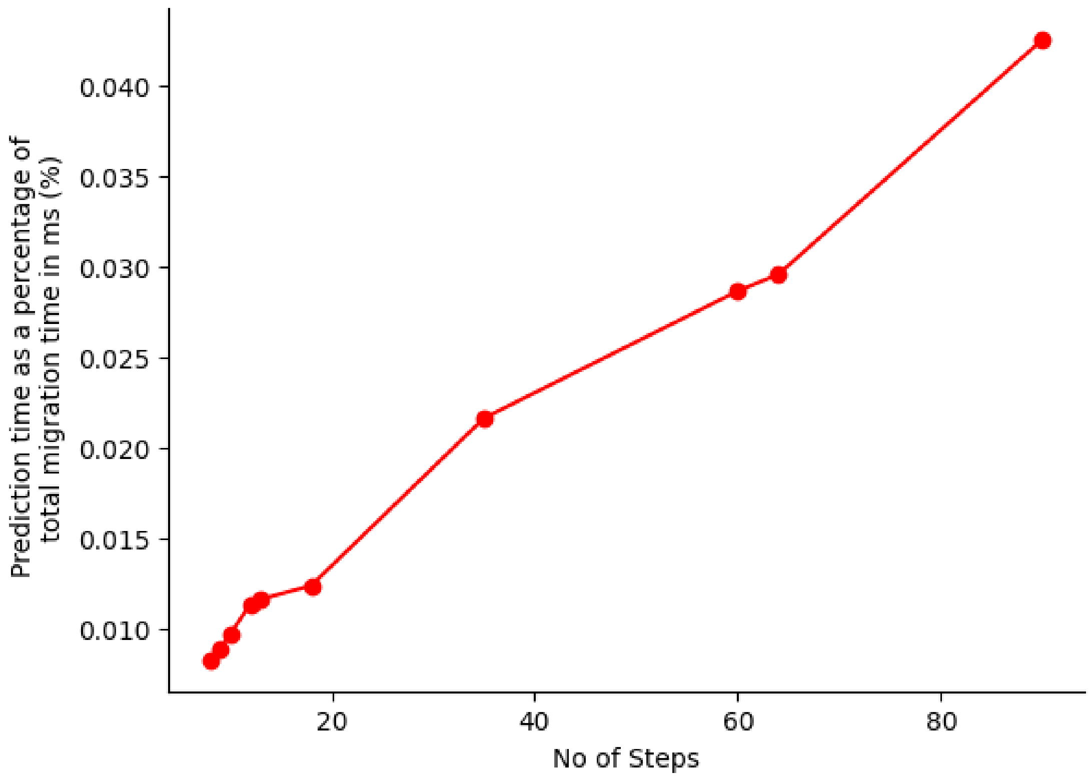

Figure 36: Prediction Time Variation

6.8 Threshold for State Definitions

First, a random value such as X% was used as the threshold when deciding the state. So, workloads with CPU, memory, or network usage greater than X% were considered a high resource usage level, while others considered them low resource usage.
To select the actual value for X, different resource usage percentages were set as the threshold, and the total migration time was measured. The results are as shown in the graphs 37 and 38.

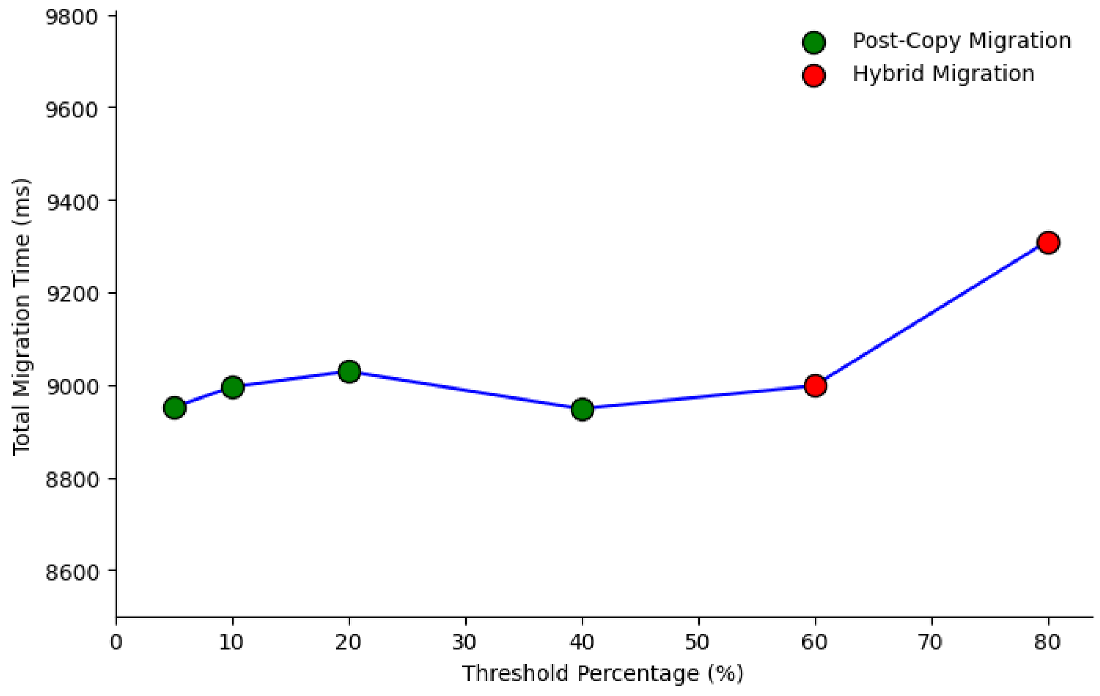

Figure 37: Threshold Selection - 4GB VM

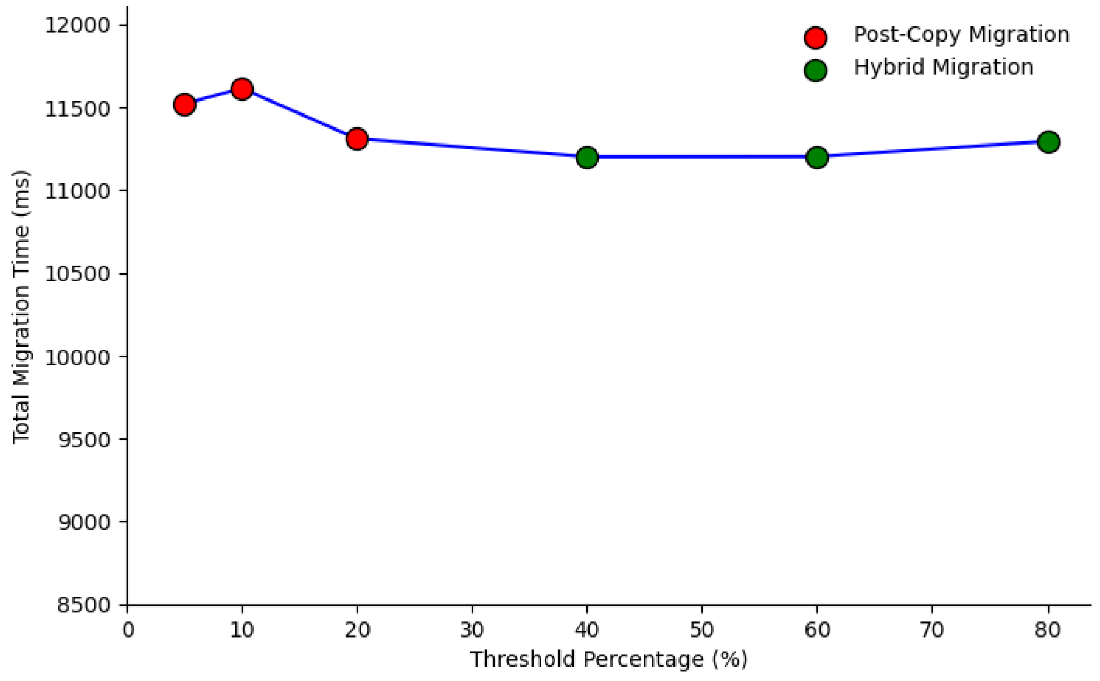

Figure 38: Threshold Selection - 8GB VM

As shown in Figure 37, for a 4GB VM, a threshold of 40% selects the optimal migration technique, which is post-copy migration. All values below 40% also lead to post-copy migration being chosen. Similarly, in Figure 38, a threshold of 40% selects the best migration technique, which is hybrid migration, with all values above 40% also favoring hybrid migration. This indicates that 40% serves as a critical threshold in both cases, determining the preferred migration strategy. Hence, 40% was used as the most suitable value for X%.

6.9 Look Ahead State

The predicted workload behavior after a certain number of steps is referred to as the look-ahead state. The primary goal of this approach is to anticipate high memory spikes that may occur during migration.
As discussed in Section 6.7, an increase in VM size leads to longer migration times. Consequently, predicting more steps enhances the accuracy of selecting the optimal migration decision.

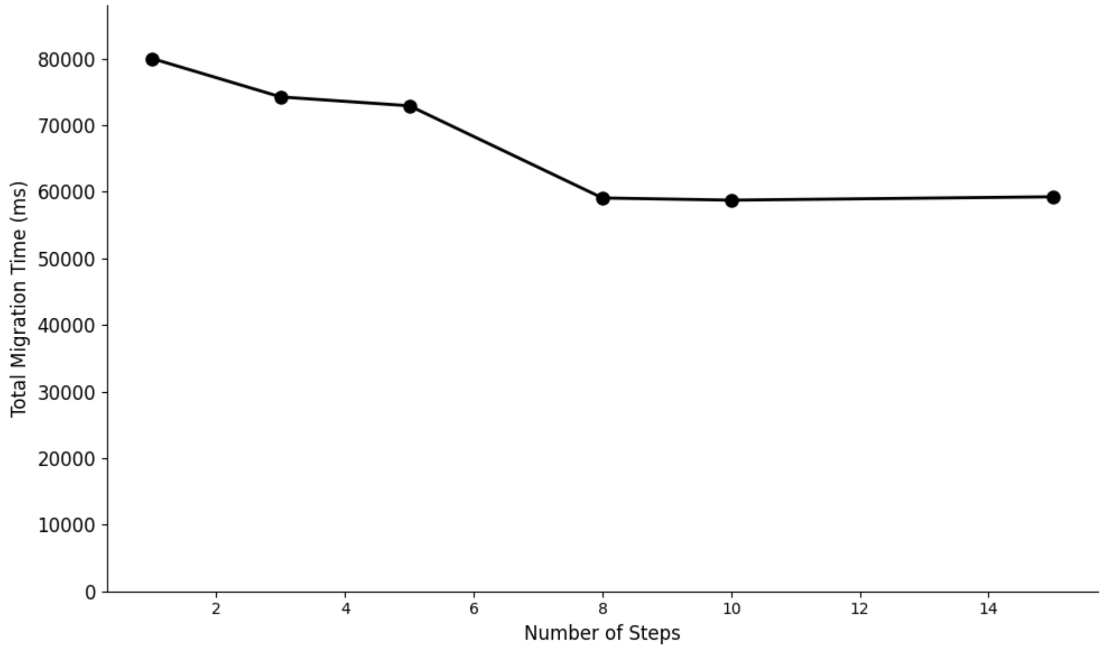

Figure 39: Impact of Prediction Steps on Total Migration Time

Figure 39 illustrates the relationship between the number of prediction steps and total migration time. The graph presents migration times for the same workload with varying prediction steps. It is evident that the accuracy of migration technique selection depends on the number of steps predicted. As shown, increasing the number of prediction steps reduces total migration time by improving the detection of memory-intensive behavior, thereby facilitating a more optimal migration strategy.

6.10 Performance Degradation

Performance degradation was measured using the Quicksort algorithm, which runs continuously and outputs the number of sorting operations performed per second. Pre-copy and post-copy migrations were conducted with and without ProMig, while Quicksort was executed before, during, and after migration.
The results, shown in the graphs below, indicate that ProMig has a minimal impact on performance during the migration process.

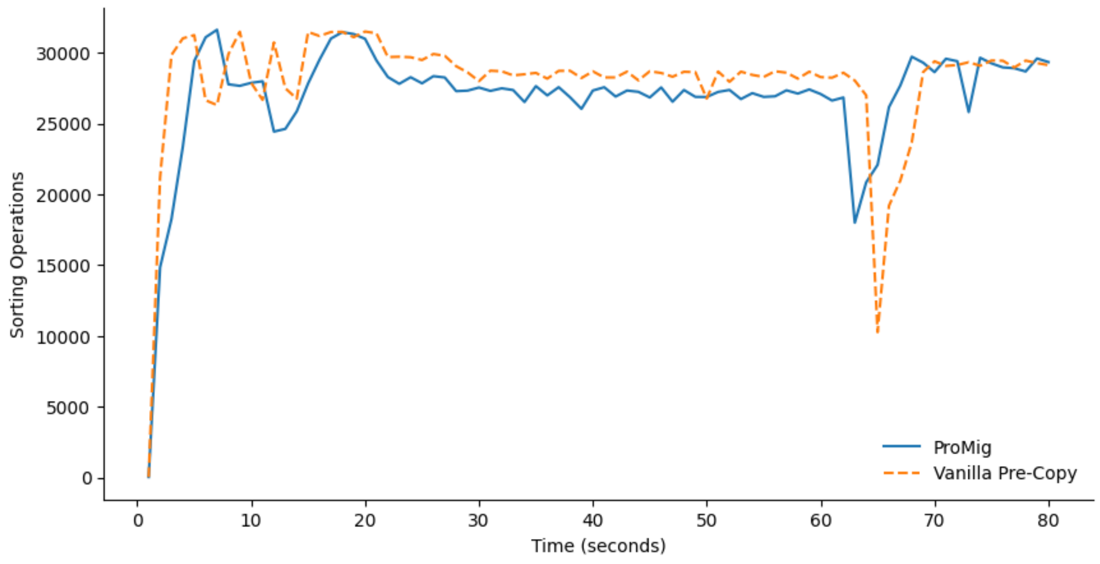

Figure 40: Comparison of Performance Degradation: ProMig vs. Vanilla Pre-Copy

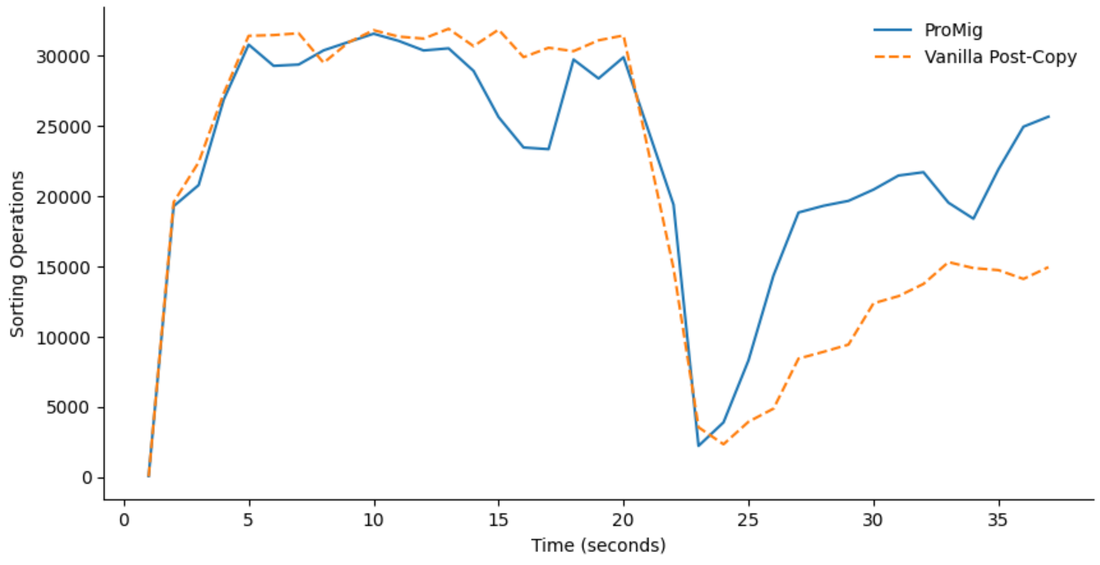

Figure 41: Comparison of Performance Degradation: ProMig vs. Vanilla Post-Copy

7 Discussion and Conclusion

The evaluation results indicate that different workload categories exhibit distinct migration behavior. Some workloads consistently lead to the selection of the same migration technique, such as stress workloads, which always follow a predefined migration path. However, other workloads require a more dynamic migration approach. For example, in the YCSB category, the migration decision varied depending on the VM size, even when running the same workload. Similarly, in synthetic workloads, the initially selected migration technique was sometimes adjusted due to anticipated high memory spikes in the future workload behavior.

These observations highlight the importance of selecting an adaptive migration strategy. Unlike traditional migration approaches that follow a fixed decision-making process, ProMig dynamically selects the most suitable migration technique based on workload characteristics, VM size, and expected future behavior. Whether the migration decision remains consistent across executions or varies dynamically, ProMig ensures that the best possible technique is always chosen, resulting in the lowest total migration time across various scenarios. As a percentage, it has 22% to 87% improvement in total migration time compared to the standard migration techniques.

Another critical aspect of migration performance is downtime and its impact on application performance. ProMig effectively minimizes downtime by leveraging the exponential decay behavior of workloads. Compared to standard hybrid migration, it achieves lower downtime while maintaining comparable performance in other cases. As a percentage, there is an 8% to 34% improvement compared to the standard hybrid migration. Additionally, it imposes minimal performance overhead, which is 4.21% compared to traditional pre-copy and post-copy migration techniques.

These findings demonstrate that ProMig is well-suited for dynamic and diverse workload environments. Its ability to anticipate future workload behavior allows it to handle sudden resource usage spikes efficiently. Furthermore, ProMig dynamically determines the convergence point based on the exponential decay trend observed in the workload during execution. This adaptability ensures optimized migration performance while maintaining system stability.

In conclusion, ProMig provides a robust and intelligent migration framework that effectively adapts to different workload conditions. By incorporating predictive capabilities and workload-aware decision-making, it enhances the efficiency of live VM migration while reducing downtime and total migration time.

8 Limitations and Future Directions

ProMig considers CPU, memory, and network usage as key resources when determining the system state and predicting future workload behavior. However, some workloads exhibit a high dirty page rate despite having low overall memory usage. This occurs when workloads frequently modify data in memory without allocating additional memory. Incorporating dirty page rate as a resource metric in future work could enhance the accuracy of migration decisions, particularly for workloads that generate frequent memory modifications.

Additionally, future research could explore extending this approach to multi-VM migration scenarios, where multiple VMs need to be migrated simultaneously while minimizing performance impact. Another potential direction is the integration of advanced predictive models to further refine migration decision-making, improving adaptability to dynamic workload patterns.

References

Altahat, M. A., Agarwal, A., Goel, N. & Kozlowski, J. (2020), 'Dynamic hybrid-copy live virtual machine migration: Analysis and comparison', *Procedia Computer Science* **171**, 1459–1468. Third International Conference on Computing and Network Communications (CoCoNet'19).
URL: `https://www.sciencedirect.com/science/article/pii/S1877050920311352`

AWS (2024), 'Amazon web services'.
URL: `https://aws.amazon.com/`

Banerjee, P., Roy, S., Modibbo, U. M., Pandey, S. K., Chaudhary, P., Sinha, A. & Singh, N. K. (2023), 'Optidjs+: A next-generation enhanced dynamic johnson sequencing algorithm for efficient resource scheduling in distributed overloading within cloud computing environment', *Electronics* **12**(19), 4123.

Calheiros, R. N., Masoumi, E., Ranjan, R. & Buyya, R. (2014), 'Workload prediction using arima model and its impact on cloud applications' qos', *IEEE transactions on cloud computing* **3**(4), 449–458.

Cerotti, D., Gribaudo, M., Piazzolla, P. & Serazzi, G. (2012), Flexible cpu provisioning in clouds: A new source of performance unpredictability, *in* '2012 Ninth International Conference on Quantitative Evaluation of Systems', IEEE, pp. 230–237.

Choudhary, A., Govil, M. C., Singh, G., Awasthi, L. K., Pilli, E. S. & Kapil, D. (2017), 'A critical survey of live virtual machine migration techniques', *Journal of Cloud Computing* **6**(1), 1–41.

Christopher, C. (2005), Live migration of virtual machines, *in* 'NSDI, 2005'.

Devi, K. L. & Valli, S. (2023), 'Time series-based workload prediction using the statistical hybrid model for the cloud environment', *Computing* **105**(2), 353–374.

Fernando, D., Yang, P. & Lu, H. (2020), Sdn-based order-aware live migration of virtual machines, *in* 'IEEE INFOCOM 2020 - IEEE Conference on Computer Communications', pp. 1818–1827.

Gambs, S., Killijian, M.-O. & del Prado Cortez, M. N. (2012), Next place prediction using mobility markov chains, *in* 'Proceedings of the first workshop on measurement, privacy, and mobility', pp. 1–6.

Ganapathi, A., Chen, Y., Fox, A., Katz, R. & Patterson, D. (2010), Statistics-driven workload modeling for the cloud, *in* '2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)', pp. 87–92.

GCP (2024), 'Google cloud platform'.
URL: `https://cloud.google.com/`

Han, J., Hong, Y. & Kim, J. (2020), ‘Refining microservices placement employing workload profiling over multiple kubernetes clusters’, *IEEE access* **8**, 192543–192556.

Hashem, I. A. T., Anuar, N. B., Gani, A., Yaqoob, I., Xia, F. & Khan, S. U. (2016), ‘Mapreduce: Review and open challenges’, *Scientometrics* **109**, 389–422.

He, T. (2021), ‘Migration management in software-defined networking-enabled edge and cloud computing environments’, *degree of Doctor of Philosophy, School of Computing and Information Systems, THE UNIVERSITY OF MELBOURNE, ORCID: 0000-0002-5472-7681* .

Hines, M. R., Deshpande, U. & Gopalan, K. (2009), ‘Post-copy live migration of virtual machines’, *SIGOPS Oper. Syst. Rev.* **43**(3), 14–26.
URL: `https://doi.org/10.1145/1618525.1618528`

Hossain, M. A. & Song, B. (2016), ‘Efficient resource management for cloud-enabled video surveillance over next generation network’, *Mobile Networks and Applications* **21**, 806–821.

Khelghatdoust, M., Gramoli, V. & Sun, D. (2016), Glap: Distributed dynamic workload consolidation through gossip-based learning, *in* ‘2016 IEEE International Conference on Cluster Computing (CLUSTER)’, IEEE, pp. 80–89.

Kim, I. K., Wang, W., Qi, Y. & Humphrey, M. (2018), Cloudinsight: Utilizing a council of experts to predict future cloud application workloads, *in* ‘2018 IEEE 11th international conference on cloud computing (CLOUD)’, IEEE, pp. 41–48.

Lin, W., Wang, J. Z., Liang, C. & Qi, D. (2011), ‘A threshold-based dynamic resource allocation scheme for cloud computing’, *Procedia Engineering* **23**, 695–703.

Lu, H., Xu, C., Cheng, C., Kompella, R. & Xu, D. (2015), vhaul: Towards optimal scheduling of live multi-vm migration for multi-tier applications, *in* ‘2015 IEEE 8th International Conference on Cloud Computing’, IEEE, pp. 453–460.

Malhotra, L., Agarwal, D., Jaiswal, A. et al. (2014), ‘Virtualization in cloud computing’, *J. Inform. Tech. Softw. Eng* **4**(2), 1–3.

Microsoft (2024), ‘Microsoft azure’.
URL: `https://azure.microsoft.com/en-us`

Naik, K. J. (2022), ‘An adaptive push-pull for disseminating dynamic workload and virtual machine live migration in cloud computing’, *International Journal of Grid and High Performance Computing (IJGHPC)* **14**(1), 1–25.

Oracle (2024), ‘Oracle — cloud applications and cloud platform’.
URL: `https://www.oracle.com/`

Pacheco-Sanchez, S., Casale, G., Scotney, B., McClean, S., Parr, G. & Dawson, S. (2011), Markovian workload characterization for qos prediction in the cloud, *in*

‘2011 IEEE 4th International Conference on Cloud Computing’, IEEE, pp. 147–154.

Sah, S. K. & Joshi, S. R. (2014), Scalability of efficient and dynamic workload distribution in autonomic cloud computing, *in* ‘2014 international conference on issues and challenges in intelligent computing techniques (ICICT)’, IEEE, pp. 12–18.

Sahni, S. & Varma, V. (2012), A hybrid approach to live migration of virtual machines, *in* ‘2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)’, pp. 1–5.

Watada, J., Roy, A., Kadikar, R., Pham, H. & Xu, B. (2019), ‘Emerging trends, techniques and open issues of containerization: A review’, *IEEE Access* **7**, 152443–152472.

Wei, Y., Kudenko, D., Liu, S., Pan, L., Wu, L. & Meng, X. (2018), A reinforcement learning based workflow application scheduling approach in dynamic cloud environment, *in* ‘Collaborative Computing: Networking, Applications and Worksharing: 13th International Conference, CollaborateCom 2017, Edinburgh, UK, December 11–13, 2017, Proceedings 13’, Springer, pp. 120–131.

Wu, Q. & Wolf, T. (2008), Dynamic workload profiling and task allocation in packet processing systems, *in* ‘2008 International Conference on High Performance Switching and Routing’, IEEE, pp. 123–130.

Xing, Y. & Zhan, Y. (2012), Virtualization and cloud computing, *in* ‘Future Wireless Networks and Information Systems: Volume 1’, Springer, pp. 305–312.

YCSB (2024), ‘Ycsb: Yahoo! cloud serving benchmark’.
URL: `https://github.com/brianfrankcooper/YCSB`

Ye, K., Wu, Z., Wang, C., Zhou, B. B., Si, W., Jiang, X. & Zomaya, A. Y. (2014), ‘Profiling-based workload consolidation and migration in virtualized data centers’, *IEEE Transactions on Parallel and Distributed Systems* **26**(3), 878–890.

Zhang, Q. & Boutaba, R. (2014), Dynamic workload management in heterogeneous cloud computing environments, *in* ‘2014 IEEE Network Operations and Management Symposium (NOMS)’, IEEE, pp. 1–7.