A TEMPORAL BIN PACKING BASED APPROACH TO VIRTUAL MACHINE PLACEMENT PROBLEM WITH MIGRATION COSTS

EDA AYAZ

ÖZYEĞİN UNIVERSITY

JUNE, 2025

A TEMPORAL BIN PACKING BASED APPROACH TO VIRTUAL MACHINE PLACEMENT PROBLEM WITH MIGRATION COSTS

by
Eda Ayaz

Thesis
Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in
Industrial Engineering

Advisor: Assoc. Prof. Dilek Günneç Danış
Co-advisor: Prof.Ali Ekici

Graduate School of Science and Engineering
Özyeğin University
İstanbul

June, 2025

A TEMPORAL BIN PACKING BASED APPROACH TO VIRTUAL MACHINE PLACEMENT PROBLEM WITH MIGRATION COSTS

Approved by:

Assoc. Prof. Dilek Günneç Danış, Advisor
Department of Industrial Engineering
Özyeğin University

Assoc. Prof. İhsan Yanıkoğlu
Department of Industrial Engineering
Özyeğin University

Assoc. Prof. Eda Yücel
Department of Industrial Engineering
TOBB University of Economics & Technology

Approval Date: May 16, 2025

To my beloved family and friends

DECLARATION OF ORIGINALITY

I hereby declare that I am the sole author of this thesis and that this is the true copy of my thesis, including the final revisions, approved by my thesis committee. All data and information have been obtained, produced, and presented in accordance with the rules of research ethics and principles of academic honesty. As required by these rules, to the best of my knowledge I have acknowledged ideas, thoughts, and any copyrighted material in accordance with the standard referencing rules. I certify that any part of this thesis has not been submitted for a degree or diploma in another educational institution.

Eda Ayaz

ABSTRACT

Cloud computing is widely adopted across many industries today due to its flexibility, scalability, and efficient resource utilization. Virtual machine placement in cloud environments is a critical optimization problem that directly affects operational efficiency. An effective placement strategy both boosts performance in data centers and reduces energy and operational costs. Existing placement algorithms typically focus on dynamic resource allocation and minimizing total costs. In this study, we present a new model for the Temporal Bin Packing (TBP) problem with migration costs, which determines a placement strategy over time that incurs the minimum possible migration.

The proposed model aims to minimize both VM migration and server usage costs by taking time-varying CPU demand into account. Whereas traditional models in the literature assume fixed time slots and non-migration-aware strategies, our approach provides more realistic workload management and adaptive resource allocation by handling variable-length time periods and time-dependent CPU demands.

To solve the VM placement problem efficiently, we introduce T-MIG (Temporal Migration-aware Heuristic), a fast, lightweight heuristic that produces feasible, near optimal placements within seconds. Instead of assuming a fixed number of servers, T-MIG dynamically determines how many servers are needed, enhancing the model's flexibility and scalability under real-world conditions. Experimental results show that, on large-scale instances and under time constraints, T-MIG outperforms BIP solvers and offers a practical alternative for real-time decision-making systems.

Keywords: Cloud Computing, Virtual Machine Placement, Temporal Bin Packing, Data Center Efficiency and Optimization, Virtual Machine Migration

ÖZET

Bulut bilişim, esnek, ölçeklenebilir ve etkin kaynak kullanımı sağlaması nedeniyle günümüzde birçok sektörde yaygın olarak kullanılmaktadır. Bulut bilişim ortamlarında sanal makine (SM) yerleştirme, operasyonel verimliliği doğrudan etkileyen kritik bir optimizasyon problemidir. Etkili bir yerleştirme stratejisi, veri merkezlerinde hem performansı artırmakta operasyonel maliyetleri azaltmaktadır. Mevcut yerleştirme algoritmaları genellikle dinamik kaynak tahsisi ve toplam maliyetlerinin azaltılmasına odaklanmaktadır. Bu çalışmada, Geçici Kutu Paketleme (Temporal Bin Packing) problemine yeni bir bakış açısı getirerek, zaman içinde minimum göçle sonuçlanacak yerleştirme stratejisini belirleyen yeni bir model önerilmiştir. Önerilen model, zamanla değişen CPU kullanım taleplerini dikkate alarak SM göçlerini en aza indirmeyi hedeflemektedir. Literatürdeki geleneksel modeller çoğunlukla sabit zaman dilimlerini ve göçe dayalı olmayan stratejileri esas alırken, bu çalışmada zamanla değişen CPU kullanımını dikkate alan ve değişken uzunlukta zaman periyotları ile çalışan daha gerçekçi bir iş yükü yönetimi ve uyarlanabilir kaynak tahsisi sağlayan bir model önerilmektedir.

Sanal makine yerleştirme problemini verimli bir şekilde çözmek amacıyla, saniyeler içerisinde uygulanabilir yerleştirme stratejileri üreten hızlı ve hafif bir sezgisel algoritma olan T-MIG, önceden tanımlı sabit sunucu sayısı varsayımı yerine, ihtiyaç duyulan sunucu sayısını dinamik olarak belirleyerek modelin gerçek dünya koşullarında esnekliğini ve ölçeklenebilirliğini artırmaktadır. Yapılan deneyler, büyük ölçekli örneklerde ve zaman kısıtı altındaki durumlarda T-MIG'in BIP çözümleyicilerden daha başarılı performans sergilediğini ve gerçek zamanlı karar sistemleri için uygulanabilir bir alternatif sunduğunu göstermektedir.

Anahtar Kelimeler: Bulut Bilişim, Sanal Makine Yerleştirme, Veri Merkezi Verimliliği ve Optimizasyonu, Sanal Makine Göçü

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor Dilek Günneç Danış and co-advisor Ali Ekici for their invaluable support and guidance throughout this research. Their expertise and thoughtful feedback have been instrumental in shaping both the direction and quality of this thesis. My sincere thanks go to Türk Telekom A.Ş. for providing the resources and collaborative environment that made this study possible. I would like to thank my family and friends for their unconditional support, encouragement, and confidence in me. Their unwavering support has been my greatest source of strength.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Cloud computing has become an essential foundation for delivering computing services at scale. It enables users to access vast pools of computational power, storage, and networking resources on demand, without investing in physical infrastructure. This on-demand model provides significant advantages in terms of scalability, cost-effectiveness, and operational flexibility, making it a preferred solution for enterprises, governments, and individuals alike.

One of the core technologies that enable cloud computing is virtualization. Virtualization allows multiple virtual machines (VMs) to run on a single physical server, each operating as an isolated environment with its own operating system and resources. This abstraction layer not only enhances resource utilization but also enables better fault isolation, flexible deployment, and efficient hardware sharing across diverse applications. Through virtualization, cloud service providers can dynamically provision and reallocate resources based on customer needs.

Virtual Machines (VMs) are the fundamental units of computation in virtualized environments. Each VM simulates a complete system and consumes a share of physical resources such as CPU, memory, and I/O. These resource demands are not static; instead, they change over time depending on the type of application running, user traffic, and workload intensity. For cloud providers operating at scale, managing the placement of VMs on physical servers—known as the VM placement problem—becomes a critical optimization challenge. This problem is NP-hard, equivalent to a multi dimensional bin-packing problem, which means exact solutions are computationally intractable for large data centers [1].

An efficient VM placement strategy directly affects the performance, reliability, and cost-efficiency of data center operations. Improper placement can lead to resource fragmentation, server over utilization or underutilization, and excessive migrations, all of

which impact energy consumption, service quality, and infrastructure wear-out. As reported in previous research, data centers account for a significant portion of global energy usage, and intelligent resource management, particularly VM placement, can substantially reduce power consumption [2].

The VM placement problem is further complicated by the need to balance two often competing objectives: maximizing resource utilization and minimizing VM migrations. Although VM migration is a critical aspect of real-world cloud management, it has been insufficiently addressed in many placement models, which tend to assume static allocations or ignore migration costs altogether. Migrations are frequently used to reallocate VMs in response to workload fluctuations; however, they come at the cost of increased network traffic, latency, and energy consumption [1]. Designing placement strategies that avoid frequent migrations while maintaining performance and flexibility remains an open challenge.

This study addresses the Temporal Bin Packing Problem with Migration Costs (TBPMC), a time-aware extension of the classical bin packing problem. Unlike traditional approaches that assume fixed-size items and static placements, TBPMC incorporates both time-varying CPU demands and the cost of migrations into the placement decision. While prior works have relied on static models or reactive reassignment strategies [3], this study introduces a proactive placement model that seeks to minimize long-term migration needs through intelligent initial decisions. Additionally, unlike models that operate on fixed time intervals, our model introduces variable-length time periods, which allow a more realistic representation of workload patterns [4].

To address this complex optimization task, we propose a Binary Integer Programming (BIP) model that jointly considers temporal resource variations and migration costs. While this model enables the derivation of optimal solutions using exact solvers such

as Gurobi, it also faces practical limitations in terms of scalability and memory requirements, especially in large-scale settings [5]. To overcome these challenges, we introduce a heuristic algorithm named T-MIG (Temporal Migration-aware Heuristic). T-MIG generates high-quality solutions within seconds, dynamically determines the number of required physical servers, and eliminates the need for predefined server counts—making it highly applicable to real-world, time-sensitive scenarios. Experimental results demonstrate that T-MIG outperforms exact solvers in several test cases under time or resource constraints and offers a scalable, energy-aware alternative for VM placement in cloud data centers.

Figure 1.1: *An example of temporal VM placement. Colored cells indicate server assignments at each time interval.*

VM	Time 1	Time 2	Time 3	Time 4
VM1	Server 1	Server 1	Server 2	Server 2
VM2	Server 2	Server 2	Server 2	Server 2
VM3	Server 3	Server 3	Server 1	Server 1

Figure 1.1 illustrates a simplified example of the temporal virtual machine placement problem. Each row represents a VM, while each column corresponds to a discrete time interval. The colored cells indicate which physical server (e.g., Server 1, Server 2, Server 3) is responsible for hosting the VM at each time step. For instance, VM1 is initially assigned to Server 1, but is migrated to Server 2 between Time 2 and Time 3, as indicated by the change in color from purple to green. Similarly, VM3 is reassigned from Server 3 to Server 1 during the same period. These migrations incur additional costs, which are considered in the optimization process. Such migrations are primarily driven by fluctuations in the CPU demands of the VMs. As applications experience varying levels of usage over time, their resource requirements, especially CPU usage, change accordingly. This dynamic behavior necessitates the reallocation of VMs to ensure that physical servers are not overloaded and resource utilization is balanced efficiently.

In real data centers, the timing and magnitude of CPU demand fluctuations can be predicted with high accuracy using historical performance data and live monitoring tools. We assume that each virtual machine's CPU demand profile is known deterministically over the entire planning horizon. In particular, some applications generate an almost constant CPU load over time, while others exhibit significant fluctuations. Our model relies on the assumption that both the magnitude of these fluctuations and the exact time points at which they occur are available a priori. This enables the placement algorithm to optimize migration and server use decisions under fully observed, time-varying workloads.

2. LITERATURE REVIEW

In this chapter, we present a review of the existing literature related to our study. The reviewed body of work can be categorized into four main streams. The first stream focuses on cloud computing and virtual machine (VM) placement, highlighting the importance of efficient resource allocation in cloud infrastructures. The second stream includes heuristic and optimization-based approaches for VM placement, with particular attention to metaheuristic algorithms developed for improving placement quality and energy efficiency. The third stream covers bin packing-based models and their temporal extensions, which provide the theoretical foundation for modeling time-dependent placement decisions in dynamic cloud environments. Finally, the fourth stream examines the effects of live migration on virtual servers—studies in this area quantify downtime, performance degradation, network congestion, and energy overhead introduced by migration operations, and demonstrate the critical impact of migration reduction on SLA compliance and system stability.

2.1 Cloud Computing and Virtual Machine Placement

Cloud computing enables flexible and scalable computing resources by leveraging virtualization technologies. One of the key challenges in cloud environments is the efficient placement of virtual machines on physical servers(hosts), which directly impacts performance, energy efficiency, and resource utilization. Several studies have addressed the optimization of VM placement in cloud infrastructures.

Liu et al [6] proposed an energy-aware VM placement scheduling method based on the Ant Colony Optimization (ACO) approach, aiming to minimize energy consumption while ensuring optimal workload distribution in cloud data centers. Similarly, Song at al. [7] introduced an optimization-based VM placement scheme that reduces resource fragmentation and enhances load balancing.

Other studies have focused on energy and resource-aware VM placement strategies. Rjeib et al. [8] proposed VMP-ER, an efficient VM placement algorithm that optimizes both energy and resource allocation in cloud data centers. Mahmoodabadi et al. [9] explored an approximation algorithm for VM placement, demonstrating its effectiveness in reducing the total number of active servers while maintaining performance constraints.

Recent works have also examined the trade-offs between performance and cost in cloud-based VM placement. Atchukatla et al. [10] analyzed cloud-based VM placement designs, identifying key challenges in cost-effective resource provisioning. Tian et al. [11] presented an efficient algorithm that improves load distribution and minimizes response time in cloud environments. These studies emphasize the critical role of efficient VM placement in cloud computing and establish a basis for further optimization efforts in dynamic and energy-conscious environments.

2.2 Virtual Machine Placement Approaches

To address VM placement challenges, researchers have explored various heuristic, metaheuristic, and optimization-based methods. A popular approach involves metaheuristic algorithms inspired by natural phenomena. Alharbi et al. [12] proposed an ACO-based dynamic VM placement system, which adapts to workload fluctuations and improves energy efficiency in cloud environments. Similarly, Al-Moalmi et al. [13] developed a Whale Optimization System for container placement, leveraging a bio-inspired optimization technique to minimize migration overhead and improve resource utilization. Other studies have focused on chaotic and greedy-based algorithms. Ganesan et al. [14] introduced a chaotic simulator for bilevel VM placement optimization, demonstrating its ability to handle large-scale cloud infrastructures. Similarly, Sun et al. [15] introduced a hybrid chaotic and evolutionary particle swarm optimization (PSO) method that incorporates conflict constraints and load balancing into 2D bin packing for cloud scenarios.

Azizi et al. [16] proposed GRVMP, a greedy randomized algorithm that efficiently maps VMs to servers, balancing execution speed and placement quality. These approaches demonstrate that metaheuristic-based methods have the potential to significantly improve VM placement efficiency; however, further advancements are required to address scalability and real-time adaptability challenges in dynamic cloud environments

2.3 Bin Packing and Temporal Bin Packing Studies

VM placement is closely related to the bin packing problem, where the objective is to efficiently pack VMs onto servers while minimizing resource wastage. Several studies have extended traditional bin packing algorithms to accommodate dynamic workloads and time-dependent constraints. Kumaraswamy et al. [17] reviewed bin packing algorithms for VM placement, identifying key techniques for reducing resource fragmentation. Hage et al. [18] explored 2D and 3D bin packing methods for VM allocation, demonstrating how spatial and temporal constraints impact placement efficiency. Pandey et al. [19] conducted a comprehensive analysis of 2D bin packing problems and their complexities, while Fatima et al [20]. directly applied bin packing models to VM placement in cloud data centers.

Advanced bin packing methods have also been applied to vector packing problems. Wei et al. [21] introduced a branch-and-price approach to handle multi-dimensional packing constraints.

A key extension of bin packing in cloud computing is Temporal Bin Packing (TBP), which considers time-varying workloads. Muir et al. [22]. examined a temporal bin packing model with half-capacity jobs, demonstrating the impact of fluctuating resource demands on placement decisions.

These studies highlight the importance of incorporating temporal constraints into bin packing models to enable more adaptive and efficient VM placement strategies in

cloud computing environments. However, the existing literature lacks sufficient focus on minimizing migration and usage costs under temporal variability, which this study aims to address.

In this context, Aydın et al. [23] proposed a multi-objective temporal bin packing model for virtual machine placement, considering both the minimization of the number of active servers and the number of server activations (fire-ups). Their work presents a comprehensive heuristic and exact solution methods for large-scale problems. Although their study incorporates the temporal dimension and energy efficiency objectives, it does not directly address migration cost minimization during dynamic resource allocation. Similarly, Cauwer et al. [24] formulated the Temporal Bin Packing Problem (TBPP) as a constraint programming model for workload management in data centers, considering time windows and resource continuity. Their work demonstrates the computational difficulty of TBPP and the importance of resource reuse over time, yet it does not address migration cost dynamics explicitly.

2.4 Migration Effects on Virtual Servers

The impacts of VM migrations on performance have been examined in numerous studies. In his work, Strunk [25] analyzed in detail how live migration operations affect system performance, showing that memory-intensive applications in particular suffer a 5% – 15% performance loss during migration. One of the pioneering studies on live migration technology, Clark et al. [26] reported that the downtime induced by migration typically ranges from 60 to 300 milliseconds. When considered cumulatively, these interruptions can have serious repercussions in systems with stringent SLA requirements. In a large-scale study conducted by Microsoft Research on Azure, Cortez et al.[27] showed that migration operations account for approximately 8% of total data-center energy consumption.

Meng et al. [28] demonstrated that migration operations cause network congestion, which in turn degrades the performance of other applications. In particular, the load imposed on backbone network links was observed to reduce the performance of other critical services. Wang et al. [29] showed that reducing migration operations plays a critical role in achieving 99.99% SLA targets, especially for latency-sensitive applications. In the financial services sector, Zhang et al. [30] demonstrated that migration-induced interruptions increase transaction failure rates and negatively impact customer satisfaction.

Therefore, VM placement algorithms frequently overlook the true impact of live migration, treating it as a negligible overhead despite the fact that migration pauses, additional network traffic, and potential packet loss can all lead to SLA violations, increased energy consumption, and degraded end-user experience. By explicitly incorporating migration cost metrics alongside time varying workload patterns, our work addresses this critical gap in the literature. In doing so, we aim not only to improve migration cost efficiency but also to demonstrate how migration-aware placement strategies can materially enhance service availability and reduce operational risk.

3. PROBLEM DEFINITION AND A MATHEMATICAL MODEL

3.1 Problem Definition

In cloud computing environments, where virtualization enables dynamic allocation of computing resources, the efficient placement of virtual machines (VMs) onto physical servers is a fundamental operational challenge. The difficulty of this task increases when considering time-varying workloads and the associated migration costs, both of which are common in real-world data centers [31].

Most traditional VM placement models either assume static resource demands or ignore the implications of VM migrations. However, in practical settings, VM CPU demands fluctuate across time periods, and relocating VMs to adapt to these changes incurs non-negligible migration costs including additional network usage, energy consumption, latency, and potential service disruption. Overlooking these dynamics leads to suboptimal resource usage, increased energy consumption, and elevated operational costs.

This study addresses a time-aware and cost-sensitive version of the VM placement problem, Temporal Bin Packing Problem with Migration Costs (TBPMC). Unlike conventional bin packing models, TBPMC explicitly incorporates both temporal variability in CPU demands and migration penalties between consecutive time periods, making it more representative of real-world cloud operations.

The core challenge is to assign each VM to a physical server in each time period, such that:

- The CPU capacities of servers are not exceeded.
- The total number of active servers is minimized to reduce energy costs.
- The number and cost of migrations are minimized to preserve service quality.

Moreover, the difficulty of this problem escalates with the scale of the data center

and the time horizon, rendering exact optimization techniques such as Binary-Integer Programming (BIP) impractical for large instances due to memory and time limitations. To overcome this, the study proposes a heuristic algorithm (T-MIG) that:

- Generates near-optimal placements with significantly reduced computation time,
- Generates subsets of previous VM assignments to reduce unnecessary migrations,
- Dynamically adjusts the number of servers used instead of relying on fixed assumptions.

Thus, the objective of this research is to develop a scalable and migration-aware placement strategy that maintains a balance between resource efficiency and operational stability under temporal uncertainty. The proposed model and heuristic are designed to fill a critical gap in the literature by jointly optimizing VM placement and migration decisions across time—contributing both to theoretical understanding and practical applicability in modern cloud data centers.

Figure 3.1: *Server consolidation through VM migration reduces energy consumption by shutting down underutilized servers.*

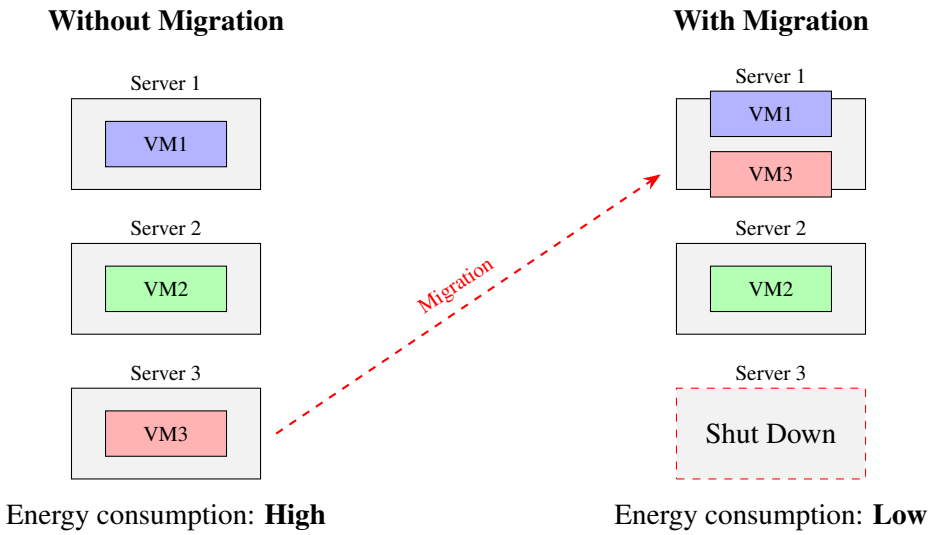


Figure 3.1 illustrates the impact of virtual machine (VM) migration on energy

consumption in a cloud data center. In the left-hand scenario (“Without Migration”), each VM is hosted on a separate physical server, resulting in all servers remaining active and consuming energy. On the right (“With Migration”), VM3 is migrated to Server 1, allowing Server 3 to be shut down. As a result, the number of active servers is reduced, leading to lower overall energy usage. This example highlights how migration strategies can enable workload consolidation and contribute to energy-efficient resource management. While Figure 3.1 illustrates the benefits of VM consolidation in terms of energy savings, it is important to note that migrating a virtual machine is not cost-free. Each migration incurs a certain overhead, such as CPU usage for live migration, network bandwidth consumption, and temporary performance degradation due to VM state transfer and disk I/O synchronization. Moreover, in large-scale systems, excessive or frequent migrations can lead to network congestion and increased system instability.

Therefore, determining when and which VMs should be migrated becomes a non-trivial optimization problem. The goal is to balance the energy savings achieved through consolidation against the operational cost of performing migrations. A migration decision that reduces energy consumption but introduces excessive migration overhead may ultimately be counterproductive.

Although some existing studies take migration costs into account, these costs are generally not treated as a primary factor and are not effectively integrated into the decision-making process. [1] In contrast, the approach proposed in this paper considers migration costs not merely as static penalties, but as dynamic decision variables distributed over time. Each VM migration is evaluated by taking into consideration both its immediate energy-saving potential and its long-term cost implications across future time intervals. This enables the system to make forward-looking, balanced placement decisions aimed at minimizing the total cost over the entire planning horizon, rather than optimizing for a single time step.

3.2 Mathematical Model

In this section, we formally define the Virtual Machine Placement Problem with Temporal Bin Packing with migration costs and present its integer programming formulation. The problem consists of a set of virtual machines (VMs) that need to be placed onto physical servers (bins/hosts) over a set of time intervals with varying durations. The objective is to minimize the total cost, which consists of VM migration costs and server utilization costs, while ensuring that the CPU capacity constraints of physical servers are respected.

Unlike classical bin packing problems, this study introduces temporal variations in CPU demand and incorporates migration costs associated with moving VMs between bins across different time periods. The set of time periods is denoted by k, where each interval may have a different duration, T_k, reflecting real-world workload fluctuations.

A VM migration occurs when a VM is placed in a different bin at a subsequent time period. The migration decision is represented by a binary variable $m_{i,k}$, which takes the value 1 if VM i changes its assigned bin at time k, and 0 otherwise. The goal of the problem is to determine an initial placement that minimizes both the total migration cost and the server usage cost, ensuring an efficient and stable allocation of VMs while respecting CPU capacity constraints.

Let $\mathcal{I} = \{1, 2, \ldots, i\}$ be the index set of virtual machines, where each VM i has a time-dependent CPU demand $p_{i,k}$ at time period k. The set of available physical servers is denoted by $\mathcal{B} = \{1, 2, \ldots, b\}$, each having a fixed CPU capacity P. The placement of VMs occurs over a set of time intervals $\mathcal{K} = \{1, 2, \ldots, k\}$, where each interval k has a variable duration denoted by T_k, representing the time difference between two consecutive periods, i.e., between k and $k + 1$.

Sets

$\mathcal{I}$: Set of virtual machines

$\mathcal{B}$: Set of physical servers (bins)

$\mathcal{K}$: Set of time periods

Decision Variables

$x_{i,b,k}$: equals 1 if VM $i \in \mathcal{I}$ is assigned to physical server $b \in \mathcal{B}$ at time period $k \in \mathcal{K}$, and 0 otherwise.

$y_{b,k}$: equals 1 if physical server $b \in \mathcal{B}$ is active at time period $k \in \mathcal{K}$, and 0 otherwise.

$m_{i,k}$: equals 1 if VM $i \in \mathcal{I}$ is migrated to a different physical server at time $k \in \mathcal{K}$, and 0 otherwise.

Parameters

$p_{i,k}$: CPU demand of VM i at time period k.

P: CPU capacity of a physical server.

T_k: Duration of time period k.

U: Server usage cost.

c_i: Migration cost for VM i.

Objective Function

$$\min \sum_{i \in \mathcal{I}} \sum_{k \in \mathcal{K}} c_i \cdot m_{i,k} + \sum_{b \in \mathcal{B}} \sum_{k \in \mathcal{K}} U \cdot T_k \cdot y_{b,k} \tag{3.1}$$

where $c_i \cdot m_{i,k}$ represents the migration cost for VM i at time period k and $U \cdot T_k \cdot y_{b,k}$ represents the server usage cost, considering the time duration T_k for which a physical server remains active.

Constraints

VM Assignment Constraint: Each VM must be assigned to exactly one physical server at any given time.

$$\sum_{b \in \mathcal{B}} x_{i,b,k} = 1, \quad \forall i \in \mathcal{I}, \forall k \in \mathcal{K}, \tag{3.2}$$

CPU Capacity Constraint: The total CPU demand of VMs placed on a physical server cannot exceed its capacity.

$$\sum_{i \in \mathcal{I}} p_{i,k} x_{i,b,k} \leq P y_{b,k}, \forall b \in \mathcal{B}, \forall k \in \mathcal{K} \tag{3.3}$$

VM Migration Constraint: A VM is considered migrated if it is assigned to a different server than in the previous time period.

$$m_{i,k} \geq x_{i,b,k} - x_{i,b,k-1}, \quad \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \setminus \{1\}, \forall b \in \mathcal{B} \tag{3.4}$$

Domain Constraints: The decision variables are binary and parameters have specific value restrictions.

$$x_{i,b,k} \in \{0,1\}, \quad \forall i \in \mathcal{I}, \forall b \in \mathcal{B}, \forall k \in \mathcal{K} \tag{3.5}$$

$$y_{b,k} \in \{0,1\}, \quad \forall b \in \mathcal{B}, \forall k \in \mathcal{K} \tag{3.6}$$

$$m_{i,k} \in \{0,1\}, \quad \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \tag{3.7}$$

An Illustrative Example

To clarify the migration decision, consider the following example where three VMs are assigned to different servers over two time periods:

Table 3.1: *VM migration decision example.*

VM	Time period $k-1$	Time period k	Previous Bin $x_{i,b,k-1}$	Current Bin $x_{i,b,k}$	Migration ($m_{i,k}$)	Elapsed Time period T_k
1	Bin 1	Bin 1	1	1	0	3 hours
2	Bin 2	Bin 3	1	0	1	2 hours
3	Bin 3	Bin 3	1	1	0	4 hours

Table 3.1 provides a detailed example to illustrate the logic behind the migration decision variable $m_{i,k}$. In this example, three virtual machines are assigned to specific physical servers (bins) over two consecutive time periods. For each VM, the table compares its server allocation in the previous period ($k-1$) and the current period (k). A migration is said to occur ($m_{i,k} = 1$) if the VM is assigned to a different server in the current period compared to the previous one. Otherwise, $m_{i,k} = 0$.

In the given example:

- VM 1 remains on Bin 1 across both periods, so no migration is present ($m_{1,k} = 0$).
- VM 2 moves from Bin 2 to Bin 3, resulting in a migration ($m_{2,k} = 1$).
- VM 3 continues on Bin 3, thus no migration occurs ($m_{3,k} = 0$).

Additionally, the example incorporates the actual duration of each time interval (T_k), shown in the last column of the table. This allows for a more accurate cost evaluation, as the server usage cost is now weighted by the length of the time period instead of assuming uniform durations. This enhancement enables the model to better reflect real-world variations in workload durations.

Since the model now incorporates T_k, the cost calculation for server usage considers the exact duration of each time interval, rather than assuming equal-length periods.

The proposed integer programming model efficiently optimizes VM placement over time, reducing both migration costs and server utilization costs. By incorporating variable-length time intervals, the model provides a more flexible and realistic approach to resource allocation in cloud computing environments.

4. PROPOSED SOLUTION APPROACHES

In this research, a series of heuristic approaches was developed prior to the proposal of the final T-MIG algorithm. Each algorithm addressed shortcomings identified in its predecessor, resulting in a method that effectively minimizes migration costs across time periods. This section outlines the algorithmic journey and the rationale behind each design decision.

Prior to the formulation of T-MIG, several strategies were explored: Initially, a simple best-fit placement; subsequently, a lookahead-enhanced variant to predict and reduce future migrations; and, ultimately, the retention-first approach, which provided the optimal balance between migration reduction and server utilization.

4.1 Initial Approach: Cost-Aware Best-Fit (CABF)

Our first heuristic approach to the VM placement problem was a modified version of the traditional Best-Fit algorithm that incorporates both migration and server usage costs. We named this approach Cost-Aware Best-Fit (CABF) as it makes placement decisions based on a cost function that considers both types of expenses.

As shown in Algorithm 4.1, this method iteratively places VMs in each time period by first sorting them in descending order of CPU demand and then assigning each VM to the server that minimizes the sum of (a) migration cost—zero if the VM remains on its previous server, otherwise a fixed penalty—and (b) usage cost—incurred only the first time a server is used in that period. If no existing server has sufficient capacity, a new server is provisioned. In this way, the approach balances migration and server-usage costs to reduce the total cost over all time periods.

While the CABF algorithm provided a baseline implementation that considered both migration and server usage costs, our experimental results revealed several significant limitations:

Figure 4.1: *Cost-aware VM placement algorithm.*

```
1:  for each time period t do
2:    Sort VMs by CPU demand at t (descending)
3:    Initialize 'activeServers = {}', 'availableCap[s] = capacity' for all servers
4:    for each VM v in sorted list do
5:      bestCost ← ∞, bestServer ← null
6:      for each server s do
7:        if availableCap[s] ≥ demand_{v,t} then
8:          migCost ← (prevServer_v==s)? 0 : MIG_COST
9:          useCost ← (s in activeServers)? 0 : USAGE_COST
10:         totalCost ← migCost + useCost
11:         if totalCost < bestCost then
12:           bestCost ← totalCost, bestServer ← s
13:         end if
14:       end if
15:     end for
16:     if bestServer is null then
17:       Provision new server s', set availableCap[s']=capacity
18:       bestServer ← s', add s' to server set
19:     end if
20:     Assign VM v to bestServer
21:     availableCap[bestServer] -= demand_{v,t}
22:     Add bestServer to activeServers
23:     Update migration  usage cost totals
24:   end for
25: end for
```

- **Short-sighted Optimization:** The algorithm makes decisions based solely on the current time period and the immediate previous placement, without considering future demand patterns.

- **Excessive Migrations:** We observed numerous unnecessary migrations, particularly during periods of minor demand fluctuations, leading to unnecessarily high migration costs.

- **Suboptimal Server Utilization:** The greedy nature of the algorithm often resulted

in fragmented resource allocation, making inefficient use of server capacity across time periods.

- **Lack of Global Optimization:** By treating each time period largely independently, the algorithm failed to achieve a globally optimal placement strategy across the entire planning horizon.

The basic nature of this approach and its inability to achieve our desired balance between migration costs and efficient server utilization led us to explore more sophisticated strategies. Our analysis of CABF's performance indicated that we needed an algorithm that could make more informed decisions by incorporating knowledge about future demand patterns and their impact on placement decisions.

This realization prompted us to develop a more advanced algorithm that could address these limitations by introducing temporal awareness and predictive capabilities into the decision-making process, which we describe in the following section.

4.2 Enhanced Approach: Lookahead Best-Fit (LBF)

Our second approach, the Lookahead Best-Fit (LBF) algorithm, given in Figure 4.2, addresses the limitations of CABF by incorporating future-awareness into the decision process. The LBF algorithm extends the basic cost model by simulating the impact of current placement decisions on future time periods. For each VM and potential server, it calculates not only the immediate migration and usage costs, but also estimates future migration costs by looking ahead up to a configurable number of time steps. This lookahead strategy enables the algorithm to make more informed placement decisions that minimize costs over a longer horizon, potentially avoiding migrations that would be necessary with a myopic approach. By incorporating temporal prediction for each VM's future demand and capacity requirements, LBF achieves a better balance between immediate costs and anticipated future costs, resulting in more stable placements across time periods.

Figure 4.2: *Lookahead VM placement algorithm.*

```
1:  Define LOOKAHEAD_DEPTH (number of future periods to consider)
2:  for each time period t at index idx do
3:      nextPeriods ← up to LOOKAHEAD_DEPTH periods after t
4:      Sort VMs by CPU demand at t (descending)
5:      for each VM v in sorted list do
6:          bestCost ← ∞, bestServer ← null
7:          for each server s do
8:              if availableCap[s] ≥ demand_{v,t} then
9:                  migCost ← (prevServer_v==s)? 0 : MIG_COST
10:                 useCost ← (s in activeServers)? 0 : USAGE_COST
11:                 capacityAfter ← availableCap[s]
12:                 futureCost ← 0
13:                 for each future period ft in nextPeriods do
14:                     futureDemand ← demand of VM v at period ft
15:                     if s ≠ prevServer_v OR capacityAfter < futureDemand then
16:                         futureCost ← futureCost + MIG_COST
17:                     end if
18:                 end for
19:                 totalCost ← migCost + useCost + futureCost
20:                 if totalCost < bestCost then
21:                     bestCost ← totalCost, bestServer ← s
22:                 end if
23:             end if
24:         end for
25:         if bestServer is null then
26:             Provision new server, update states as in CABF
27:         end if
28:         Assign VM v to bestServer and update all states
29:         Update migration & usage cost totals (excluding future cost)
30:     end for
31: end for
```

While the Lookahead Best-Fit (LBF) algorithm improved upon CABF by incorporating future awareness, our extensive testing revealed several significant limitations:

- **Isolated Future Projections:** The LBF algorithm projects future migrations for each VM individually, without considering the placement and potential conflicts with other VMs in future time periods. This isolation leads to overly optimistic future cost estimations, as the algorithm fails to account for the cascading effects of multiple VMs competing for the same server resources in future time periods.
- **Partial Optimization:** While the algorithm reduces migrations in some cases, it sometimes creates unnecessary migrations in others due to its partial view of the future. It optimizes for a fixed lookahead window rather than the entire planning horizon, resulting in locally optimal but globally suboptimal decisions.

Our experimental studies demonstrated that the LBF algorithm provided notable improvements compared to CABF in certain scenarios. Particularly for VM sets with predictable demand patterns, the LBF algorithm reduced the number of migrations without causing a significant decrease in server utilization efficiency. Although LBF provided better results than the initial version, we recognized potential for improvement as it did not consider other VMs in the targeted time periods. Therefore, we sought a new algorithm that would more directly address the temporal and migration-focused nature of the VM placement problem, one that would prevent migrations by preserving existing placements as much as possible. This search led to the development of the TMIG approach, which focuses on minimizing migration costs and maximizing placement stability.

4.3 Temporal - Migration-Aware Heuristic (T-MIG)

In this section, we present our T-MIG heuristic—named "Temporal - Migration-aware" to emphasize its focus on both time-varying demands and migration overhead—for

the Virtual Machine Placement Problem with Temporal Bin Packing and Migration Costs. The algorithm builds an initial placement and then, in each later interval, retains as many VMs as possible on their current hosts before assigning the remainder. Our goal is to minimize the migration costs across all periods. Before arriving at T-MIG, we experimented with several strategies: initially a simple best-fit placement, then a lookahead-enhanced version to predict future migrations, and finally the current retention-first approach, which achieved the best balance between migration reduction and server utilization.

Initial Assignment: At the very first interval, we sort all virtual machines by their CPU demand (highest first). Then, one by one, each VM is placed on the first already-active server with enough spare capacity. If no existing server can accommodate it, we start a new one. This "best-reuse" step ensures we keep the number of active servers—and therefore usage cost—as low as possible from the outset.

Retention-First Phase: For every subsequent interval, we look back at where each VM was running. We group VMs by their previous server and, for each group, we keep the largest subset whose combined demand still fits that server's capacity. Those VMs stay put, directly minimizing the number of migrations in this interval.

Any VMs not retained are then re-sorted by current demand (highest first) and placed greedily: each goes to the first server with sufficient free capacity, or triggers the activation of a new server if needed.

In each interval we record:

$$\text{Migration cost} = (\#\text{ of VMs that moved}) \times (\text{per-VM migration cost})$$
$$\text{Usage cost} = (\#\text{ of active servers}) \times (\text{per-server usage cost})$$

These components are summed across intervals to yield the total cost. By front-loading retention—preserving maximal subsets on each server—and following with a simple greedy fallback, T-MIG balances the trade-off between migration overhead and server utilization. It runs quickly (polynomial in the number of VMs and servers per interval)

and consistently produces near-optimal total cost in our experiments.

Figure 4.3 shows the T-MIG algorithm, which processes VM CPU demands at each time step k to minimize the combined cost of server usage and VM migrations. At the start of each period, all VMs are sorted in descending order of their current CPU demand $p_{i,k}$. If $k > 1$, for each server b the largest subset of VMs from the previous period that still fits within capacity P is retained, and the remaining VMs are released into an "unassigned" pool. Those unassigned VMs are then re-sorted by demand, and for each VM i and each candidate server b two cost components are computed: migration cost c_i if the VM moves, and usage cost $U \cdot T_k$ if the server must be activated. The sum $c_{total} = c_i + U \cdot T_k$ determines the best server; if no existing server can host the VM, a new server is added. After assignments, the period's actual migration and usage costs are accumulated into the running total. Once all time steps are processed, the algorithm outputs the VM–time–server assignment matrix $x_{i,b,k}$ and the overall cost.

Figure 4.3: *T-MIG Heuristic for VM Placement with Migration Costs.*

Input: VM set $\mathcal{I}$, time intervals $\mathcal{K}$, CPU capacity P, usage cost U, migration cost c_i
Output: Assignment $x_{i,b,k}$ and total cost
Initialize server pool $\mathcal{B} \leftarrow \{1, \ldots, m\}$ with capacity P
Initialize $x_{i,b,k} \leftarrow \emptyset$, $\mathrm{TotalCost} \leftarrow 0$, $\mathrm{MigCost} \leftarrow 0$, $\mathrm{UseCost} \leftarrow 0$
for each time step $k \in \mathcal{K}$ **do**
  Let $p_{i,k}$ be the CPU demand of VM i at time k
  if k is first time step **then**
    Sort $\mathcal{I}$ by decreasing $p_{i,k}$
    for each $i \in \mathcal{I}$ **do**
      if exists $b \in \mathcal{B}$ with remaining capacity $\geq p_{i,k}$ **then**
        assign i to the first such b
      else
        create new server b_{new} with capacity P, add to $\mathcal{B}$, assign i to b_{new}
      end if
      update capacity of chosen server
    end for
    $\mathrm{UseCost} \mathrel{+}= |\mathcal{B}_{\text{used}}| \times U \cdot T_k$
  else
    Let $k' \leftarrow$ previous time step
    Group VMs by their server at k'
    for each server $b \in \mathcal{B}$ **do**
      Let $\mathcal{I}_b$ be VMs on b at k'
      Find subset $R \subseteq \mathcal{I}_b$ with maximal $|R|$ such that $\sum_{i \in R} p_{i,k} \leq P$
      Retain R on b, mark others in $\mathcal{I}_b \setminus R$ as unassigned
    end for
    Sort all unassigned VMs by decreasing $p_{i,k}$
    for each unassigned VM i **do**
      if exists $b \in \mathcal{B}$ with remaining capacity $\geq p_{i,k}$ **then**
        assign i to that b
      else
        create new server and assign
      end if
      update capacities
    end for
    Compute migration and usage cost at k
  end if
  $\mathrm{TotalCost} = \mathrm{MigCost} + \mathrm{UseCost}$
end for
return $x_{i,b,k}$, $\mathrm{TotalCost}$

4.4 Step-by-Step Execution of T-MIG Algorithm on a 3×3 Instance

To illustrate how T-MIG operates in practice, we now present its step-by-step execution on a small 3×3 instance (3 VMs over 3 time periods). This detailed example clarifies the algorithm's decision process and optimization strategy.

The parameters of the algorithm are defined as follows:

Table 4.1: *Algorithm parameters*

Parameter	Description	Value
B	Set of servers	server 1, server 2,
I	Set of virtual machines	vm 1, vm2
K	Set of time periods	time 1, time 2, time 3
P	Server CPU capacity	32
U	Server usage cost	5
c_i	VM migration cost	3
pi, k	CPU demand of VM i at time k	Given as a matrix

4.4.1 Scenario and Input Data

Table 4.2 shows the CPU demand of each virtual machine (VM) in each time period. The system parameters are as follows: CPU_Capacity = 32, Server_Usage_Cost = 5, Migration_Cost = 3.

Table 4.2: *CPU Demands of VMs in Each Time Period*

VM	time1	time2	time3
VM1	16	14	10
VM2	12	10	16
VM3	4	22	8

4.4.2 Time Period 1

In the first time period, VMs are sorted by CPU demand in descending order: VM1 (16) $>$ VM2 (12) $>$ VM3 (4). They are then placed using the First-Fit Decreasing algorithm:

- VM1 (16) is placed on Server1, remaining capacity: $32 - 16 = 16$

- VM2 (12) is placed on Server1, remaining capacity: $16 - 12 = 4$
- VM3 (4) is placed on Server1, remaining capacity: $4 - 4 = 0$

In this case, all VMs are accommodated on a single server (Server1), and the server capacity is fully utilized.

Table 4.3: *VM Assignments at the End of Time Period 1*

Server	VMs	CPU Utilization
Server1	VM1, VM2, VM3	32/32

Cost for Time Period 1:

- Server usage cost: $1 \times 5 = 5$
- Migration cost: 0 (first time period)
- Total: 5

4.4.3 Time Period 2

In the second time period, the CPU demands of the VMs have changed. Previous assignment and new demands:

- Previous assignment: Server1: [VM1, VM2, VM3]
- New CPU demands: VM1 (14), VM2 (10), VM3 (22)
- Total demand: $14 + 10 + 22 = 46 > 32$, therefore all VMs cannot fit on Server1

The algorithm evaluates all possible VM subsets for Server1. The subset evaluation is shown in Table 4.4:

The algorithm determines the subsets that fit and require the minimum number of migrations for Server1. Both $\{$VM1, VM2$\}$ and $\{$VM2, VM3$\}$ subsets require 1 migration. The algorithm selects $\{$VM1, VM2$\}$ as it is encountered first in the evaluation order.

As a result:

- VM3 is migrated to Server2

Table 4.4: *Subset Evaluation for Time Period 2*

Subset	CPU Demand	Fits?	Number of Migrations
{VM1, VM2, VM3}	46	No	0
{VM1, VM2}	24	Yes	1 (VM3)
{VM1, VM3}	36	No	1 (VM2)
{VM2, VM3}	32	Yes	1 (VM1)
{VM1}	14	Yes	2 (VM2, VM3)
{VM2}	10	Yes	2 (VM1, VM3)
{VM3}	22	Yes	2 (VM1, VM2)
{}	0	Yes	3 (VM1, VM2, VM3)

Table 4.5: *VM Assignments at the End of Time Period 2*

Server	VMs	CPU Utilization
Server1	VM1, VM2	24/32
Server2	VM3	22/32

- Server1: [VM1, VM2], total demand: 24/32
- Server2: [VM3], total demand: 22/32

  Cost for Time Period 2:

- Server usage cost: $2 \times 5 = 10$, Migration cost: $1 \times 3 = 3$ (VM3 was migrated)
- Total Cost: 13

4.4.4 Time Period 3

In the third time period, the CPU demands of the VMs have changed again:

- Previous assignments: Server1: [VM1, VM2], Server2: [VM3]
- New CPU demands: VM1 (10), VM2 (16), VM3 (8)

  Server1 evaluation:

VM1, VM2 $\rightarrow 10 + 16 = 26 < 32$, fits, no migration required

  Server2 evaluation:

VM3 $\rightarrow 8 < 32$, fits, no migration required

  Cost for Time Period 3:

- Server usage cost: $2 \times 5 = 10$, Migration cost: 0 (no migrations)
- Total Cost: 10

Table 4.6: *VM Assignments at the End of Time Period 3*

Server	VMs	CPU Utilization
Server1	VM1, VM2	26/32
Server2	VM3	8/32

4.4.5 Total Cost

Total cost for all time periods:

- Total server usage cost: $5 + 10 + 10 = 25$, Total migration cost: $0 + 3 + 0 = 3$
- Overall total cost: 28

Table 4.7: *VM Assignments in All Time Periods*

VM	time1	time2	time3
VM1	Server1	Server1	Server1
VM2	Server1	Server1	Server1
VM3	Server1	Server2	Server2

This VM placement algorithm is fundamentally designed to minimize the number of VM migrations. In each time period, the algorithm evaluates all possible subsets of VMs on the servers and selects the subset that requires the minimum number of migrations. This example was tested with Gurobi and reached the same output, demonstrating that the algorithm successfully achieves its goal of minimizing the number of migrations. While Gurobi uses mathematical optimization to reach an exact solution, our algorithm employs a heuristic approach. The fact that both methods yield the same result shows that the heuristic approach can be effective for these types of problems. Though the methodologies differ (Gurobi uses integer programming while our algorithm evaluates subsets), in this particular example, both reached the optimal solution by reducing the number of migrations to 1.

5. COMPUTATIONAL RESULTS

5.1 Data Generation:

In this section, we describe the procedure used to generate synthetic datasets for evaluating the performance of the proposed VM placement algorithm under varying workload scales. The datasets are designed to represent small, medium, and large-scale cloud environments, enabling comprehensive testing of both solution quality and scalability.

The CPU demand values were derived from real data center traces and then classified based on their temporal load patterns. Specifically, we categorized the instances into mixed load profiles to reflect different workload dynamics observed in practice.

Each dataset consists of a time indexed matrix, where rows represent virtual machines (VMs) and columns represent time periods. The entries correspond to the CPU demand of each VM in each time step. These CPU demands are generated using different random patterns to reflect temporal variability and workload diversity. By simulating dynamic and heterogeneous resource requirements, we aim to assess the robustness of the algorithm under realistic and challenging cloud conditions.

Four classes of datasets are constructed as follows:

- **Small scale instances:** 20 VMs across 50 time periods, and 30 VMs across 80 time periods.
- **Medium scale instances:** 40 or 50 VMs across 100 time periods.
- **Medium scale instances with heterogeneous server usage costs:** Same VM and time configurations as standard medium-scale instances (i.e., 40 or 50 VMs over 100 time periods), but with varied server usage cost values (1 or 25 units) to test cost sensitivity.
- **Large scale instances:** 100 VMs across 100 or 150 time periods.

Each dataset adopts a different CPU demand pattern to reflect specific workload scenarios. In particular, we generate mixed load CPU demand profiles across time periods to create diverse and realistic test cases. This allows us to evaluate the algorithm's effectiveness not only in uniform conditions but also under varying workload dynamics.

All instances are stored in Excel format, with each sheet containing a two dimensional CPU demand matrix. These instances are consistently used in both exact and heuristic solution approaches to ensure fair comparison. In addition to our baseline choice (Migration cost $M = 3$, Usage cost $U = 5$), we performed a parameter sweep over $U \in \{1, 5, 25\}$ (keeping $M = 3$) to explore how the relative weight of server usage vs. migration cost affects both absolute cost and the difference between T-MIG and Gurobi. Results are reported in the tables below. The load level of the system significantly affects migration costs. In heavily loaded systems, there is higher resource consumption and potential risk of service interruption during virtual machine migration. As demonstrated in the study by Liu et al. [32], the migration cost ratio is expected to be higher in systems operating at high utilization levels. Particularly in virtual machines running memory intensive or I/O-intensive applications, this cost increases further due to the high amount of dirty page generation during migration [33]. In our VM placement analysis, we used different cost ratios to represent different system load scenarios.

To obtain optimal or near optimal solutions for benchmarking, we formulated the problem as a Binary Integer Programming model and solved it using the Gurobi Optimizer (version 12.0.1, build v12.0.1rc0). All computational experiments were conducted on a 64-bit Windows 10 system equipped with an 11th Gen Intel(R) Core(TM) i7-1185G7 CPU running at 3.00GHz, with instruction set support for SSE2, AVX, AVX2, and AVX512.

Table 5.1: *Test Instances and Their Characteristics*

Instance(s)	Number of VMs	Time Intervals	CPU Capacity	Migration Unit Cost	Server Usage Cost
SMALL INSTANCES					
1–5	20	50	32	3	5
6–10	30	80	32	3	5
MEDIUM INSTANCES					
11–15	40	100	32	3	5
16–20	50	100	32	3	5
LARGE INSTANCES					
21–27	100	100	32	3	5
28–30	100	150	32	3	5
MEDIUM INSTANCES – VARIANT (Different Usage Cost)					
31–35	40	100	32	3	1
36–40	50	100	32	3	25
41–45	40	100	32	3	1
46–50	50	100	32	3	25

Table 5.1 summarizes the characteristics of the test instances used in our experiments. The datasets are grouped into four categories—Small, Medium, Large, and Medium Variant—based on the number of virtual machines (VMs), time intervals, and cost configurations. Each instance simulates a virtual machine placement scenario over a specific time horizon, with constant CPU capacity and predefined cost parameters.

In all instances, the CPU capacity of physical servers is fixed at 32 units. The cost of migrating a VM is set to 3 units. For Small instances, which include either 20 VMs over 50 time intervals or 30 VMs over 80 intervals, the server usage cost is fixed at 5 units. Medium instances consist of 40 or 50 VMs evaluated over 100 time intervals, also with a usage cost of 5 units. Large instances contain 100 VMs, with time intervals of either 100 or 150, under the same cost settings.

Additionally, we introduce a set of 20 Medium Variant instances that maintain the same VM and time interval configurations as the Medium group, but vary in their

server usage costs (1 or 25 units). This extended set enables a broader evaluation of our algorithm's sensitivity to operational cost parameters and its adaptability under different pricing conditions.

5.2 Numerical Results

The computational results presented in this section demonstrate the evolution and comparative performance of our three proposed heuristics CABF, LBF, and T-MIG for solving the temporal VM placement problem with migration costs. Through a series of experiments on various datasets, we analyze how each algorithm addresses the dual objectives of minimizing migration costs while maintaining efficient server utilization.

We begin by examining the performance of our initial Cost-Aware Best-Fit (CABF) approach, which serves as our baseline. While CABF provides reasonable solutions by considering immediate migration and server usage costs, its myopic decision making leads to suboptimal placements across multiple time periods, particularly in scenarios with fluctuating demands. Next, we evaluate the Lookahead Best-Fit (LBF) algorithm, which enhances CABF by incorporating future awareness into placement decisions. Our results show that LBF reduces migration costs compared to CABF in certain scenarios. However, its performance varies significantly depending on the dynamics of the workload and the lookahead depth parameter. The algorithm's reliance solely on current capacity information when projecting future migrations, without considering the impact of other VMs in target time periods, limits its effectiveness in highly dynamic environments. Finally, we present the results for our proposed T-MIG heuristic, which adopts a fundamentally different retention-first approach. As illustrated in the following tables and figures, T-MIG consistently outperforms both CABF and LBF, producing solutions with lower migration costs across diverse problem instances. The performance of T-MIG is particularly notable on large-scale datasets, where exact solvers such as Gurobi either

exceed acceptable time limits or fail due to memory constraints. These results highlight T-MIG's potential as a scalable and responsive alternative for dynamic, cost sensitive VM placement in production environments. The comparative analysis of these three heuristics reveals not only their relative strengths and limitations but also demonstrates the value of our iterative approach to algorithm development. By systematically addressing the shortcomings identified in each predecessor, we ultimately arrived at T-MIG, which successfully balances the tradeoff between migration cost minimization and efficient server utilization across time periods.

To evaluate the performance difference between our proposed heuristic algorithm (T-MIG) and the exact solver (Gurobi), we use the relative percentage difference, denoted as Diff. which is calculated as shown in equation 5.1 below:

$$\text{Diff. } (\%) = \left(\frac{\text{T-MIG Cost} - \text{Gurobi Cost}}{\text{Gurobi Cost}} \right) \times 100 \tag{5.1}$$

This metric quantifies the relative deviation of T-MIG, LBF, CABF from the optimal or near-optimal solutions obtained by Gurobi. A positive difference indicates that the heuristic yields a higher cost, while a negative difference implies that the heuristic performed better than Gurobi for that specific cost component.

Gurobi's Scalability Limitations

Given the real time decision making requirements in cloud data centers where allocation decisions must often be made within seconds, we enforced a 3600-second (1-hour) time limit on Gurobi for solving each instance. This constraint reflects the urgency of operational responsiveness in practical scenarios, where delays in resource allocation, particularly those arising from communication overhead, can lead to performance degradation and system inefficiencies [34]. Under this time limit, Gurobi often failed to reach

optimality for medium and largescale datasets and was only able to return feasible solutions. For the largest datasets, the solver encountered out of memory errors, which highlights its limitations in terms of scalability. In contrast, T-MIG produced feasible solutions within seconds for all instances, offering a practical alternative in time constrained environments.

5.2.1 Performance Evaluation of the CABF Heuristic (Cost-Aware Best-Fit)

In this section, we provide a systematic evaluation of our CABF heuristic by progressively analyzing its performance on small, medium, and large instance sets under varying cost weightings. We begin with the baseline configuration (Migration Cost = 3, Usage Cost = 5) on small instances (Table 5.12), then extend to medium and large workloads (Tables 5.13 and 5.14), and finally explore the impact of adjusting the Usage Cost weight to 25 and 1 on medium cases (Tables 5.5 and 5.6). Through these sequential analyses, we highlight CABF's strengths in reducing server-usage expenses as well as its challenges in controlling migration overhead, thereby identifying key areas for algorithmic refinement.

5.2.1.1 Comparison of CABF and Gurobi on Small Instances

Table 5.2: *Comparison of CABF and Gurobi on Small Instances*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF
1	3981	4658	17.01	651	1593	144.70	3330	3065	-7.96	3600	0.6
2	4038	4767	18.05	573	1542	169.11	3465	3225	-6.93	3600	0.4
3	3002	3622	20.65	387	1197	209.30	2615	2425	-7.27	3600	0.5
4	2989	3643	21.88	399	1248	212.78	2590	2395	-7.53	3600	0.6
5	4043	4729	16.97	588	1614	174.49	3455	3115	-9.84	3600	0.8
6	10964	11388	3.87	924	3483	276.95	10040	7905	-21.26	3600	0.8
7	11520	11728	1.81	435	2763	535.17	11085	8965	-19.12	3600	0.9
8	10950	11494	4.97	795	3444	333.21	10155	8050	-20.73	3600	0.9
9	11194	11620	3.81	954	3180	233.33	10240	8440	-17.58	3600	0.8
10	11466	11892	3.72	531	2982	461.58	10935	8910	-18.52	3600	0.8
Average			**11.27**			**275.06**			**-13.67**		

*Gurobi execution time limited to 1 hour

As shown in Table 5.2, our initial CABF algorithm achieves notable gains in Usage Cost but falls short on Total Cost and Migration Cost. Specifically, while the average Usage Cost decreases by 13.67%, the Total Cost improvement is only 11.27%, and Migration Cost actually increases by an average of 275.06%. This pattern indicates that CABF is effective at consolidating VM placements to reduce server-usage expenses but requires further refinement to curb the high costs associated with VM migrations.

5.2.1.2 Comparison of CABF and Gurobi on Medium Instances

Table 5.3: *Comparison of CABF and Gurobi on Medium Instances.*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF
11	14811	15414	4.07	2031	5694	180.35	12780	9720	-23.94	3600	0.1
12	19869	19823	-0.23	969	4338	347.68	18900	15485	-18.07	3600	0.3
13	17147	18484	7.80	2652	6744	154.30	14495	11740	-19.01	3600	0.4
14	20234	19656	-2.86	2889	4326	49.74	17345	15330	-11.62	3600	0.5
15	19972	19759	-1.07	2802	4809	71.63	17170	14950	-12.93	3600	0.5
16	24894	24661	-0.94	2979	5916	98.59	21915	18745	-14.46	3600	0.3
17	24433	24708	1.13	2043	5373	163.00	22390	19335	-13.64	3600	0.4
18	24923	24544	-1.52	2298	5874	155.61	22625	18670	-17.48	3600	0.6
19	24647	24669	0.09	1737	5964	243.35	22910	18705	-18.35	3600	0.3
20	25799	24727	-4.16	4284	5367	25.28	21515	19360	-10.02	3600	0.4
Average			**0.23**			**148.95**			**-15.95**		

*Gurobi execution time was limited to 1 hour.

As shown in Table 5.3, CABF on medium-sized instances yields almost no change in Total Cost (0.23% improvement) but leads to a dramatic average increase of 148.95% in Migration Cost. Meanwhile, Usage Cost decreases by 15.95%. These results confirm that, although CABF excels at consolidating workloads to cut server-usage expenses, it incurs substantially higher migration overhead for medium workloads and thus needs further tuning to balance both objectives.

5.2.1.3 Comparison of CABF and Gurobi on Large Instances

Table 5.4: *Comparison of CABF and Gurobi on Large Instances*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF
21	42294	41736	-1.32	17634	20286	15.04	24660	21450	-13.02	3600	0.5
22	50403	49733	-1.33	6033	12333	104.43	44370	37400	-15.71	3600	0.5
23	50310	49737	-1.14	7380	12462	68.86	42930	37275	-13.17	3600	0.6
24	50629	49423	-2.38	7914	12723	60.77	42715	36700	-14.08	3600	0.7
25	49093	48385	-1.44	7443	13995	88.03	41650	34390	-17.43	3600	0.6
26	50909	49754	-2.27	8154	12354	51.51	42755	37400	-12.52	3600	0.6
27	76153	74021	-2.80	13398	19896	48.50	62755	54125	-13.75	3600	0.9
28	74672	74355	-0.42	10722	20160	88.02	63950	54195	-15.25	3600	0.8
29	75545	74864	-0.90	11760	19944	69.59	63785	54920	-13.90	3600	0.5
30	76245	74785	-1.91	12210	19650	60.93	64035	55135	-13.90	3600	0.9
Average			**-1.59**			**65.57**			**-14.27**		

*Gurobi execution time was limited to 1 hour.

As shown in Table 5.4, our initial CABF algorithm on large instances yields a modest decrease in Total Cost (–1.59%) and a substantial reduction in Usage Cost (–14.27%), demonstrating its strength in consolidating VMs to lower server-usage expenses. However, Migration Cost increases by 65.57%, indicating that the algorithm's aggressive reshuffling of VMs incurs significant overhead at scale and requires further tuning to balance migration and usage objectives.

5.2.1.4 Comparison of CABF and Gurobi on High Usage Cost Instances

Table 5.5: *Comparison of CABF and Gurobi on High Usage Cost Instances*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF
31	62640	54294	-9.52	2790	5694	104.09	59850	48600	-18.80	3600	0.3
32	89519	81763	-8.66	3144	4338	37.98	86375	77425	-10.36	3600	0.3
33	72591	65444	-9.85	3741	6744	80.27	68850	58700	-14.74	3600	0.3
34	88151	80976	-8.14	3426	4326	26.27	84725	76650	-9.53	3600	0.5
35	86778	79559	-8.32	3228	4809	48.98	83550	74750	-10.53	3600	0.6
36	109396	99641	-8.92	3996	5916	48.05	105400	93725	-11.08	3600	0.7
37	111164	102048	-8.20	3039	5373	76.80	108125	96675	-10.59	3600	0.5
38	112328	99224	-11.67	2928	5874	100.61	109400	93350	-14.67	3600	0.4
39	112536	99489	-11.59	2811	5964	112.17	109725	93525	-14.76	3600	0.5
40	109278	102167	-6.51	4878	5367	10.02	104400	96800	-7.28	3600	0.6
Average			**-9.52**			**64.52**			**-12.23**		

*Gurobi execution time was limited to 1 hour.

As shown in Table 5.5, lowering the Usage Cost weight to 25 leads CABF to achieve an average Total Cost reduction of 9.52% and a Usage Cost drop of 12.23%, but at the expense of a 64.52% increase in Migration Cost. This indicates that while tuning the model to favor usage-cost savings improves overall consolidation, it also amplifies VM reshuffling overhead, necessitating additional strategies to mitigate migration costs.

5.2.1.5 Comparison of CABF and Gurobi on Low Usage Cost Instances

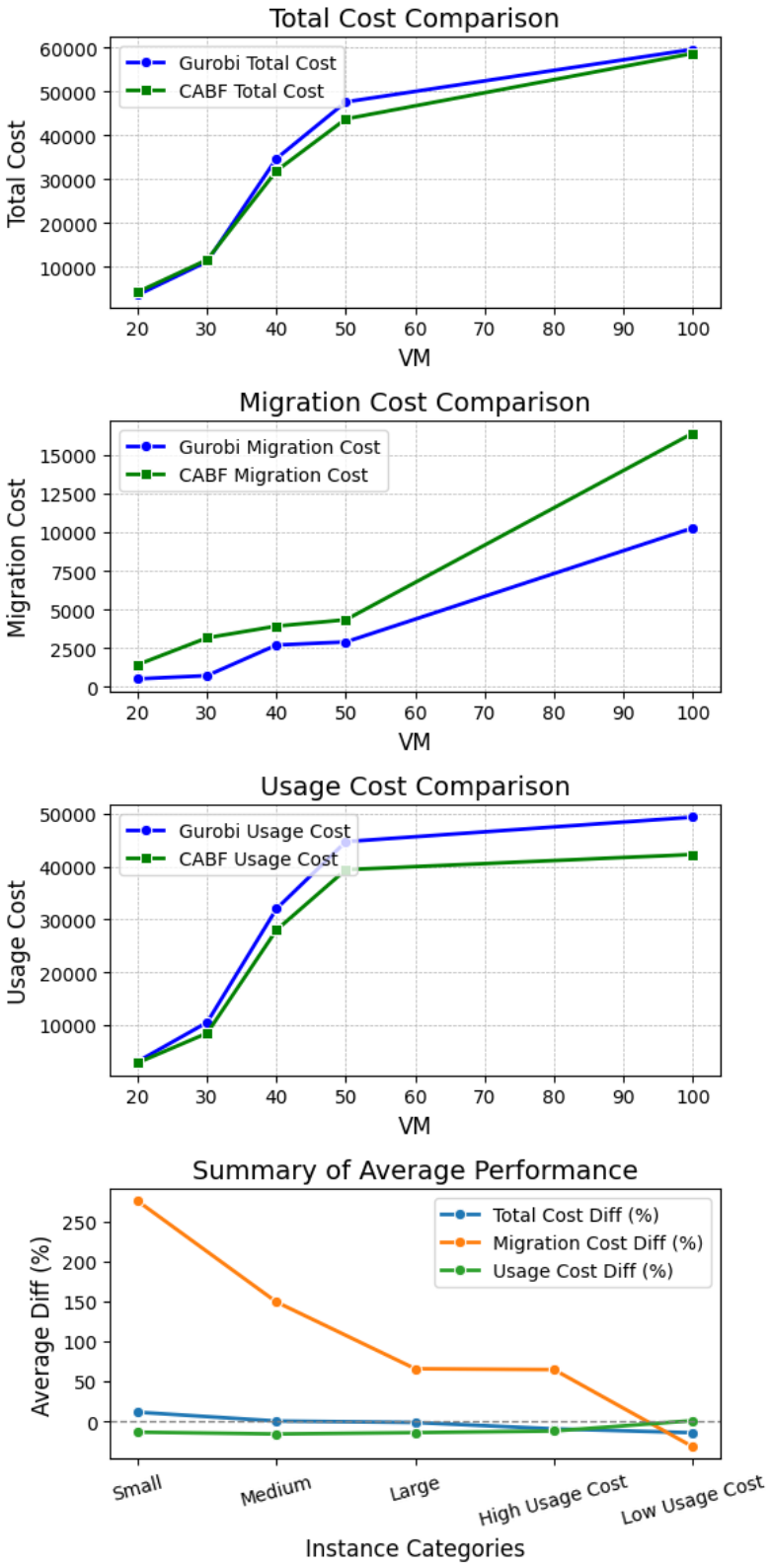
Table 5.6: *Comparison of CABF and Gurobi on Medium Instances with Low Usage Cost*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF	Diff. (%)	Gurobi*	CABF
41	4471	4008	-10.36	1899	1437	-24.33	2572	2571	-0.04	3600	0.5
42	7106	4437	-37.56	3690	681	-81.50	3416	3756	9.90	3600	0.5
43	5142	5302	3.11	2067	2271	9.87	3075	3031	-1.43	3600	0.6
44	6367	4756	-25.30	2820	1074	-61.91	3547	3682	3.81	3600	0.3
45	5941	5107	-14.04	2415	1596	-33.91	3526	3511	-0.43	3600	0.3
46	7193	6357	-11.62	2694	2001	-25.72	4499	4356	-3.18	3600	0.4
47	6304	5695	-9.66	3725	1080	-71.03	4579	4615	0.79	3600	0.5
48	6696	6306	-5.82	2145	1929	-10.07	4551	4377	-3.82	3600	0.4
49	6273	6357	1.34	1638	2001	22.16	4635	4356	-6.02	3600	0.4
50	8654	5700	-34.13	4326	1047	-75.80	4328	4653	7.51	3600	0.3
Average			**-14.40**			**-31.86**			**0.71**		

*Gurobi execution time was limited to 1 hour.

As shown in Table 5.6, reducing the Usage Cost weight to 1 drives CABF to achieve a substantial average Total Cost reduction of 14.40% and a 31.86% decrease in Migration Cost, while Usage Cost remains essentially unchanged (+0.71%). This outcome indicates that deprioritizing usage costs compels the algorithm to favor migration savings, significantly lowering both overall and migration expenses without materially increasing server-usage costs.

Figure 5.1: *CABF and Gurobi: Total, Migration and Usage Cost Comparison*



Overall, as shown in Figure 5.1 CABF consistently excels at reducing server-usage expenses across all instance sizes, yet it incurs significant VM migration overhead that undermines its total-cost benefits. To address this imbalance, we turn our attention to a lookahead-enhanced placement strategy. In the next section, we will introduce our Lookahead Best Fit LBF algorithm and share its performance results; in doing so, we will examine how integrating future-demand predictions can reduce migration costs and shift the balance with usage-cost efficiency.

5.2.2 Performance Evaluation of the LBF Heuristic (Lookahead Best-Fit)

Building on the CABF framework, our Lookahead Best-Fit (LBF) algorithm incorporates a one-step demand forecast into the placement decision. At each time period, LBF evaluates not only the current resource requirements but also anticipated workloads in the next period, allowing it to favor VM assignments that reduce future migrations while maintaining high server-usage efficiency. In this section, we will examine its performance across different datasets. For these experiments, the lookahead horizon was set to 10 time periods.

5.2.2.1 Comparison of LBF and Gurobi on Small Instances

Table 5.7: *Comparison of Lookahead Best-Fit (LBF) and Gurobi on Small Instances*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF
1	3981	4568	14.75	651	378	-41.94	3330	4190	25.83	3600	0.7
2	4038	4850	20.11	573	240	-58.12	3465	4610	33.04	3600	1.1
3	3002	3615	20.42	387	255	-34.11	2615	3360	28.49	3600	0.8
4	2989	3330	11.41	399	495	24.06	2590	2835	9.46	3600	0.7
5	4043	4647	14.94	588	177	-69.90	3455	4470	29.38	3600	1.3
6	10964	11594	5.75	924	564	-38.96	10040	11030	9.86	3600	2.8
7	11520	11708	1.63	435	543	24.83	11085	11165	0.72	3600	2.6
8	10950	11398	4.09	795	633	-20.38	10155	10765	6.01	3600	2.9
9	11194	11224	0.27	954	984	3.14	10240	10240	0.00	3600	2.5
10	11466	11627	1.40	531	537	1.13	10935	11090	1.42	3600	2.4
Average			**9.48**			**-21.02**			**14.42**		

*Gurobi execution time was limited to 1 hour.

As shown in Table 5.7, the LBF algorithm increases Total Cost by 9.48% compared to Gurobi, reduces Migration Cost by 21.02%, but worsens Usage Cost by 14.42%. This indicates that while lookahead integration effectively curbs costly VM migrations, it introduces additional server-usage expenses, highlighting a trade-off between migration reduction and usage efficiency.

5.2.2.2 Comparison of LBF and Gurobi on Medium Instances

Table 5.8: *Comparison of Lookahead Best-Fit (LBF) and Gurobi on Medium Instances*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF
11	14811	14358	-3.06	2031	1533	-24.52	12780	12825	0.35	3600	4.9
12	19869	19519	-1.76	969	924	-4.64	18900	18595	-1.61	3600	5.8
13	17147	17496	2.04	2652	2406	-9.28	14495	15090	4.10	3600	3.5
14	20234	19473	-3.76	2889	1098	-61.99	17345	18375	5.94	3600	5.8
15	19972	19143	-4.15	2802	1623	-42.08	17170	17520	2.04	3600	3.8
16	24894	23775	-4.50	2979	2025	-32.02	21915	21750	-0.75	3600	6.2
17	24433	24134	-1.22	2043	1119	-45.23	22390	23015	2.79	3600	6.6
18	24923	23776	-4.60	2298	1926	-16.19	22625	21850	-3.43	3600	8.4
19	24647	23766	-3.57	1737	2031	16.93	22910	21735	-5.13	3600	7.8
20	25799	24270	-5.93	4284	1050	-75.49	21515	23220	7.92	3600	8.8
Average			**-3.05**			**-29.45**			**1.22**		

*Gurobi execution time was limited to 1 hour.

Compared to small instances, LBF on medium workloads now achieves an average Total Cost reduction of 3.05% (vs. a 9.48% increase previously) As shown in Table 5.8 and a larger average Migration Cost decrease of 29.45% (vs. 21.02%), while the Usage Cost penalty drops to only 1.22% (from 14.42%). This indicates that the lookahead strategy becomes more effective at scale, delivering overall cost savings with minimal impact on server-usage efficiency.

5.2.2.3 Comparison of LBF and Gurobi on Large Instances

Table 5.9: *Comparison of Lookahead Best-Fit (LBF) and Gurobi on Large Instances*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF
21	42294	39801	-5.89	17634	15831	-10.22	24660	23970	-2.80	3600	14.3
22	50403	47748	-5.27	6033	3918	-35.06	44370	43830	-1.22	3600	25.1
23	50310	47771	-5.05	7380	3891	-47.28	42930	43880	2.21	3600	24.9
24	50629	47353	-6.47	7914	4248	-46.32	42715	43105	0.91	3600	24.2
25	49093	47005	-4.25	7443	4260	-42.77	41650	42745	2.63	3600	22.4
26	50909	48220	-5.28	8154	2775	-65.97	42755	45445	6.29	3600	28.5
27	76153	72413	-4.91	13398	3318	-75.24	62755	69095	10.10	3600	45.5
28	74672	72630	-2.73	10722	3525	-67.12	63950	69105	8.06	3600	48.5
29	75545	73048	-3.31	11760	3423	-70.89	63785	69625	9.16	3600	39.2
30	76245	73244	-3.94	12210	3594	-70.57	64035	69650	8.77	3600	40.5
Average			**-4.71**			**-53.14**			**4.41**		

*Gurobi execution time was limited to 1 hour.

Compared to small and medium datasets, the LBF algorithm on large instances further improves its overall effectiveness. While on small instances LBF increased Total Cost by 9.48% and on medium instances it reduced Total Cost by 3.05%, on large workloads it achieves an average Total Cost reduction of 4.71%. Migration Cost reduction also grows from 21.02% (small) and 29.45% (medium) to 53.14% on large instances, as shown in Table 5.9, demonstrating that lookahead's benefit in curbing VM reshuffling scales with problem size. Usage Cost penalties, which were 14.42% on small and 1.22% on medium, settle at 4.41% on large instances—indicating a more balanced trade-off between migration savings and server-usage efficiency at higher scales.

5.2.2.4 Comparison of LBF and Gurobi on High Usage Cost Instances

Table 5.10: *Comparison of LBF and Gurobi on Medium Instances with High Usage Cost Instances*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF
31	62640	64870	3.56	2790	1770	-36.56	59850	63100	5.43	3600	3.8
32	89519	92765	3.63	3144	1215	-61.35	86375	91550	5.99	3600	5.1
33	72591	76496	5.38	3741	2721	-27.27	68850	73775	7.15	3600	4.2
34	88151	91821	4.16	3426	1371	-59.98	84725	90450	6.76	3600	3.8
35	86778	88046	1.46	3228	1896	-41.26	83550	86150	3.11	3600	4.4
36	109396	109581	0.17	3996	2331	-41.67	105400	107250	1.76	3600	7.5
37	111164	115004	3.45	3039	1404	-53.80	108125	113600	5.06	3600	7.5
38	112328	109671	-2.37	2928	2271	-22.44	109400	107400	-1.83	3600	6.6
39	112536	109506	-2.69	2811	2331	-17.08	109725	107175	-2.32	3600	7.8
40	109278	115543	5.73	4878	1443	-70.42	104400	114100	9.29	3600	6.9
Average			**2.25**			**-43.18**			**4.04**		

*Gurobi execution time was limited to 1 hour.

Comparing Table 5.10 (Usage Cost = 25) against the baseline (Usage Cost = 5, Table 5.8), we observe that raising the usage cost weight leads LBF to increase Total Cost from an average decrease of 3.05 % to an average increase of 2.25%, further reduce migration cost (from –29.45% down to –43.18%), worsen usage cost more significantly (from +1.22 % up to +4.04%),

These shifts indicate that emphasizing usage cost savings (by inflating its weight) drives the lookahead policy to accept more VM reshuffling overall—trading additional server-usage expense for even greater migration reductions.

5.2.2.5 Comparison of LBF and Gurobi on Low Usage Cost Instances

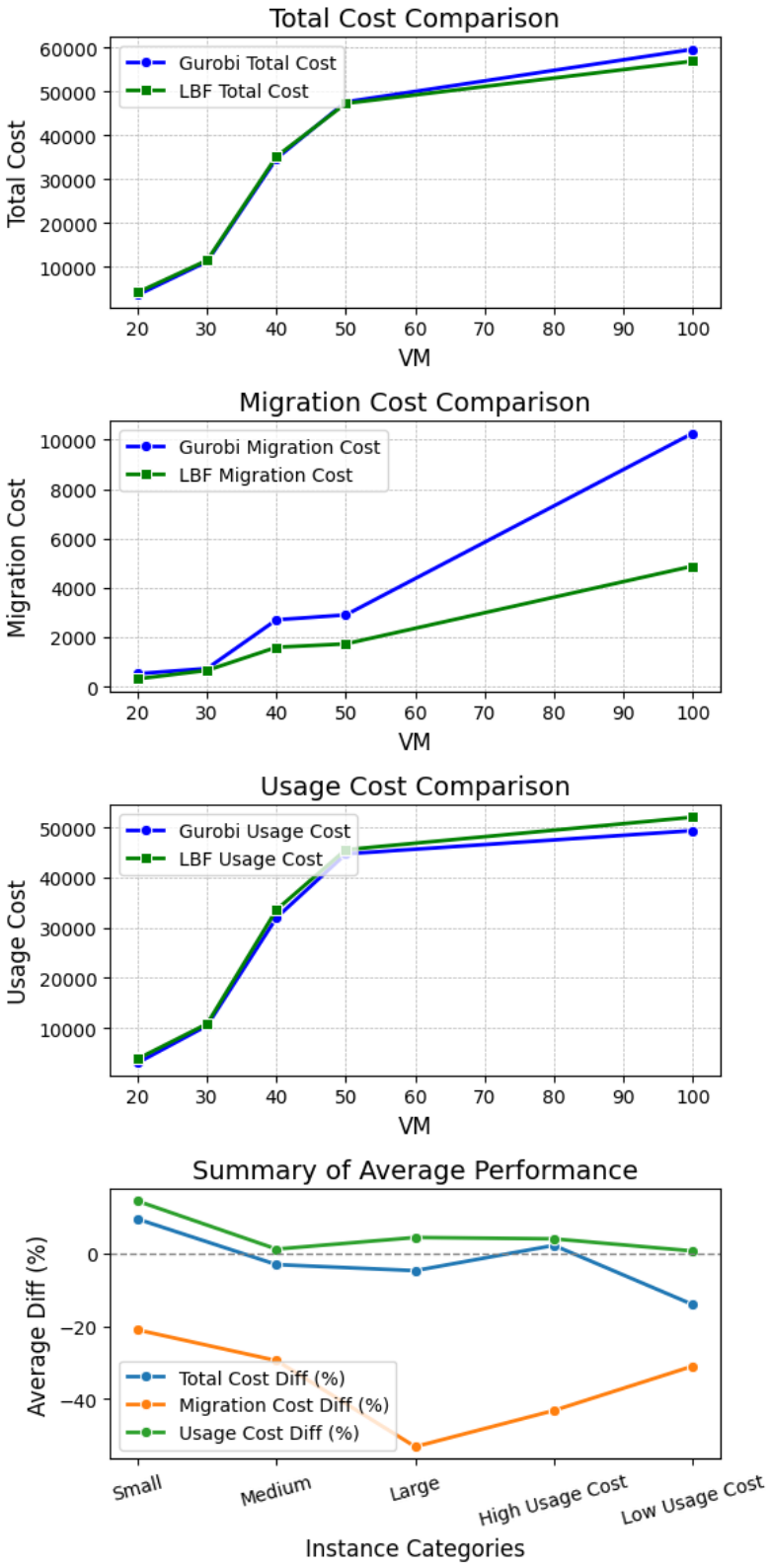
Table 5.11: *Comparison of LBF and Gurobi on Medium Instances with Low Usage Cost Instances*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF	Diff. (%)	Gurobi*	LBF
41	4471	4062	-9.15	1899	1491	-21.48	2572	2571	-0.04	3600	4.8
42	7106	4437	-37.56	3690	681	-81.54	3416	3756	9.90	3600	5.9
43	5142	5470	6.38	2067	2439	18.00	3075	3031	-1.43	3600	4.9
44	6367	4756	-25.30	2820	1074	-61.91	3547	3682	3.81	3600	4.4
45	5941	5107	-14.04	2415	1596	-33.91	3526	3511	-0.43	3600	3.9
46	7193	6357	-11.62	2694	2001	-25.72	4499	4356	-3.18	3600	6.1
47	6304	5695	-9.66	3725	1080	-71.03	4579	4615	0.79	3600	8.2
48	6696	6276	-6.27	2145	1899	-11.47	4551	4377	-3.82	3600	7.6
49	6273	6357	1.34	1638	2001	22.16	4635	4356	-6.02	3600	6.4
50	8654	5673	-34.45	4326	1020	-76.42	4328	4653	7.51	3600	8.2
Average			**-14.03**			**-30.97**			**0.71**		

*Gurobi execution time was limited to 1 hour.

Comparing the low usage cost configuration (Table 5.11) with both the baseline setup (Usage Cost = 5, Table 5.8) and the high–Usage Cost variant (Usage Cost = 25, Table 5.10), we see that reducing the usage cost to 1 drives the Total Cost down to –14.03% (versus +2.25% under high usage), delivers a substantial yet slightly moderated migration cost reduction of –30.97% (compared to –43.18%), and keeps the Usage Cost penalty negligible at +0.71%. Together, these results confirm that deprioritizing server usage charges enables the LBF algorithm to achieve its strongest overall cost savings while maintaining effective migration control.

Figure 5.2: *LBF and Gurobi: Total, Migration and Usage Cost Comparison*



As shown in Figure 5.2, although the Lookahead Best-Fit (LBF) algorithm leverages limited future demand predictions, it proved inadequate for our temporal VM placement problem: it operates over a restricted lookahead window, fails to capture all VM interactions, and as refinements are mades olution times grow prohibitively long. Consequently, we sought alternative placement strategies. Accordingly, we developed an approach focused on keeping VMs in place as much as possible by considering past placements. We designed the Temporal-MIG (T-MIG) method based on previous assignment history. In the next section, we will explain the performance of the T-MIG algorithm in detail.

5.2.3 Performance Evaluation of the T-MIG Heuristic (Temporal -Migration-Aware)

The computational results presented in this section demonstrate the effectiveness and practicality of the proposed T-MIG heuristic in solving the temporal VM placement problem with migration costs. As will be illustrated in the following tables and figures, T-MIG consistently produces solutions with low total cost—comprising both migration and server usage costs—within a matter of seconds. This rapid execution is especially valuable in real-world cloud environments, where allocation decisions must be made promptly to avoid service disruptions or resource inefficiencies. The heuristic's performance is particularly notable on large-scale datasets, where exact solvers such as Gurobi either exceed acceptable time limits or fail due to memory constraints. These results highlight T-MIG's potential as a scalable and responsive alternative for dynamic, cost-sensitive VM placement.

The performance of the proposed heuristic algorithm T-MIG is compared against the exact solver Gurobi in Tables 5.12–5.15. These tables present a detailed breakdown of total cost, migration cost, and server usage cost across small, medium, and large instance categories. For each metric, the percentage gap between the two methods is calculated to

measure the relative deviation of the heuristic from the optimal or near optimal solution. In addition, execution times are reported to highlight the computational efficiency of the heuristic. This comparison enables a comprehensive evaluation of both solution quality and scalability across varying problem sizes.

5.2.3.1 Performance Evaluation on Small Instances

Table 5.12: *Comparison of T-MIG and Gurobi on Small Instances.*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	T-MIG	Diff.	Gurobi*	T-MIG	Diff.	Gurobi*	T-MIG	Diff.	Gurobi*	T-MIG
1	3981	4842	21.63	651	42	-93.55	3330	4800	44.14	3600	0.2
2	4038	4914	21.69	573	39	-93.19	3465	4875	40.69	3600	0.2
3	3002	3647	21.49	387	72	-81.4	2615	3575	36.71	3600	0.1
4	2989	3474	16.23	399	99	-75.19	2590	3375	30.31	3600	0.1
5	4043	4986	23.32	588	6	-98.98	3455	4980	44.13	3600	0.2
6	10964	11903	8.56	924	48	-94.81	10040	11855	18.08	3600	0.3
7	11520	11894	3.25	435	84	-80.69	11085	11810	6.54	3600	0.3
8	10950	11896	8.64	795	51	-93.58	10155	11845	16.64	3600	0.2
9	11194	11820	5.59	954	60	-93.71	10240	10150	-0.88	3600	0.2
10	11466	11903	3.81	531	48	-90.96	10935	11855	8.41	3600	0.3
Average			**13.42**			**-89.60**			**24.47**		

*Gurobi execution time limited to 1 hour.

Table 5.12 presents a comparative analysis of our proposed T-MIG algorithm against the Gurobi solver on small-scale VM placement instances. The evaluation focuses on three cost metrics—Total Cost, Migration Cost, and Usage Cost—as well as computation time.

Total Cost: Although Gurobi achieves lower total costs on all instances, the average relative difference remains moderate at 13.42%. This indicates that T-MIG produces solutions

that are reasonably close to optimal in terms of total cost, while being orders of magnitude faster.

Migration Cost: T-MIG significantly outperforms Gurobi with respect to migration cost. On average, T-MIG reduces the migration cost by 89.61% compared to Gurobi. This suggests that our algorithm is highly effective at minimizing the number of VM migrations, which is particularly beneficial in dynamic and high-latency environments.

Usage Cost: T-MIG tends to increase the server usage cost compared to Gurobi. The average relative increase is 24.48%. This is expected, as T-MIG often prefers keeping VMs on the same servers to avoid migration penalties, even if that results in higher resource usage. This trade-off demonstrates the algorithm's bias toward minimizing migration overhead rather than aggressively consolidating VMs.

Execution Time: One of the most striking observations is the execution time. While Gurobi consistently hits the maximum time limit of 3600 seconds, T-MIG returns results in under 0.3 seconds for all instances. This substantial speed advantage confirms the scalability and practical applicability of T-MIG in real-time or large-scale scenarios where timely decision-making is critical.

In summary, T-MIG provides a compelling alternative to exact solvers like Gurobi, offering rapid solutions with significantly lower migration overhead, albeit at the cost of a modest increase in server usage.

Figure 5.3: *Migration Cost Comparison between Gurobi and T-MIG across Small Instances*

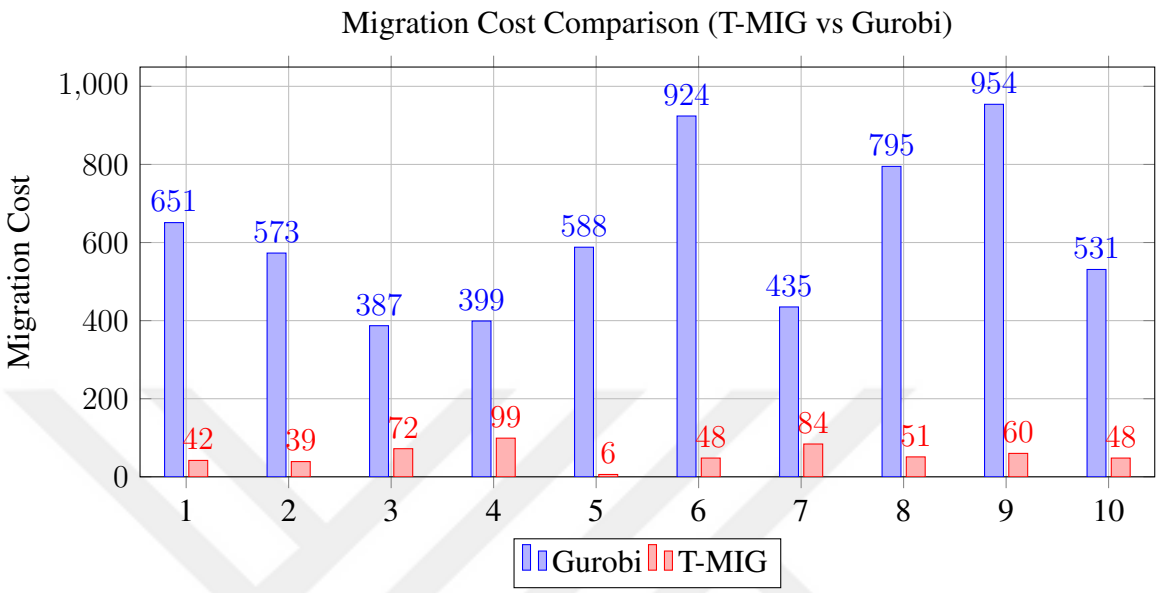


Figure 5.4: *Migration Cost Reduction (%) of T-MIG Compared to Gurobi*

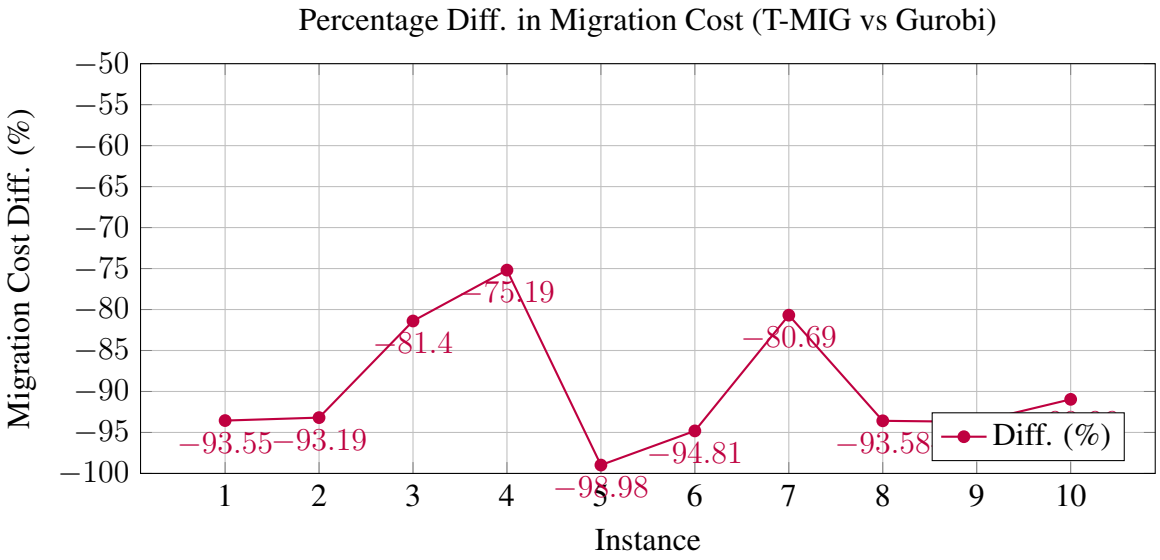


Figure 5.3 and 5.4 compares the absolute migration costs of Gurobi and T-MIG

across all small scale test instances. It is evident that T-MIG achieves dramatically lower migration costs in every single case. While Gurobi incurs hundreds of migration units, often close to or exceeding 900—T-MIG consistently keeps migration cost below 100, with several instances registering as low as 6 or 39 units.

To further illustrate this trend, Figure 5.4 presents the percentage reduction in migration cost achieved by T-MIG relative to Gurobi. The difference(diff.) values range between -75.19% and -98.98%, with an average reduction of -89.61%. This substantial improvement demonstrates T-MIG's strong emphasis on minimizing virtual machine movements across time periods.

While this migration aware behavior improves system stability and reduces overhead from frequent reallocations, it also leads to a trade off in server usage. Specifically, as shown in Table 5.12, T-MIG exhibits a higher usage cost in most small instances compared to Gurobi. This indicates that the algorithm tends to activate more servers in order to avoid costly migrations. Although this is an intentional design decision, it also highlights an opportunity for further improvement—particularly in small scale scenarios where smarter consolidation could reduce overall usage costs without significantly increasing migrations.

In summary, T-MIG excels at limiting migration overhead, offering a robust solution in dynamic environments. However, future versions of the algorithm could be enhanced to better balance usage efficiency with migration avoidance.

5.2.3.2 Performance Evaluation on Medium Instances

Table 5.13: *Comparison of T-MIG and Gurobi on Medium Instances.*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	T-MIG	Diff.	Gurobi*	T-MIG	Diff.	Gurobi*	T-MIG	Diff.	Gurobi*	T-MIG
11	14811	15531	4.86	2031	276	-86.41	12780	15255	19.37	3600	0.2
12	19869	19854	-0.08	969	69	-92.88	18900	19785	4.68	3600	0.2
13	17147	19523	13.86	2652	168	-93.67	14495	19355	33.53	3600	0.2
14	20234	19887	-1.71	2889	57	-98.03	17345	19830	14.33	3600	0.2
15	19722	19881	0.81	2028	66	-96.74	17170	19745	15.03	3600	0.2
16	24894	24769	-0.5	2979	84	-97.18	21915	24685	12.64	3600	0.2
17	24433	24911	1.96	2043	66	-96.77	22390	24845	10.96	3600	0.2
18	24923	24641	-1.13	2298	126	-94.52	22625	24515	8.35	3600	0.2
19	24647	24769	0.49	1737	84	-95.16	22910	24685	7.75	3600	0.2
20	25799	24892	-3.52	4284	57	-98.67	21515	24835	15.43	3600	0.2
Average			**1.34**			**-95.09**			**14.20**		

*Gurobi execution time limited to 1 hour.

Table 5.13 presents a comparative analysis of the T-MIG algorithm against the Gurobi solver on medium scale VM placement instances. As with small instances, the evaluation is based on Total Cost, Migration Cost, Usage Cost, and Execution Time.

Total Cost: Unlike in small instances, where Gurobi outperforms T-MIG in every case, the results in medium instances are more balanced. While Gurobi still achieves slightly better total costs on average, the mean difference narrows significantly to just 1.34%, compared to 13.42% in small instances. This indicates that T-MIG scales well and continues to provide competitive solutions even as problem size increases.

Migration Cost: T-MIG again shows a clear advantage over Gurobi in minimizing migration costs. On average, it reduces migration cost by 95.09%, which is even better than the 89.61% reduction observed in small instances. This confirms that T-MIG maintains its

migration efficiency regardless of problem scale, making it highly suitable for dynamic, latency sensitive systems.

Usage Cost: Similar to the trend in small instances, T-MIG exhibits higher server usage costs than Gurobi, with an average increase of 14.20%. While this increase is lower than the 24.48% observed in small instances, it still reflects T-MIG's strategy of favoring stable allocations to avoid migration penalties, occasionally at the cost of higher resource utilization.

Execution Time: The runtime advantage of T-MIG remains consistent. Gurobi once again reaches the time limit in all cases, while T-MIG provides solutions in just 0.2 seconds. It is important to note that the 3600 second limit for Gurobi was externally imposed to ensure fair comparison and practical feasibility. This substantial difference highlights the suitability of T-MIG for real time or large scale scenarios where computational efficiency is critical.

In conclusion, while the performance difference in total cost narrows in medium instances, T-MIG continues to offer substantial gains in migration efficiency and runtime performance. Its trade off between usage and migration remains evident but slightly more balanced compared to small scale scenarios.

Figure 5.5: *Migration Cost Comparison between Gurobi and T-MIG across Medium Instances*

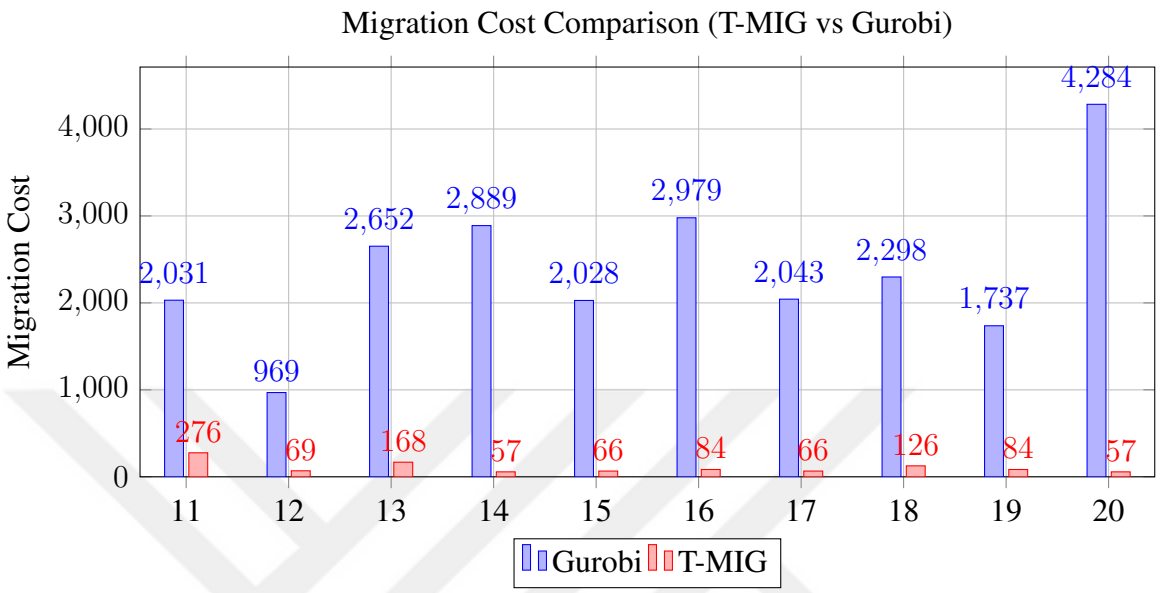


Figure 5.6: *Migration Cost Reduction (%) of T-MIG Compared to Gurobi (Medium Instances)*

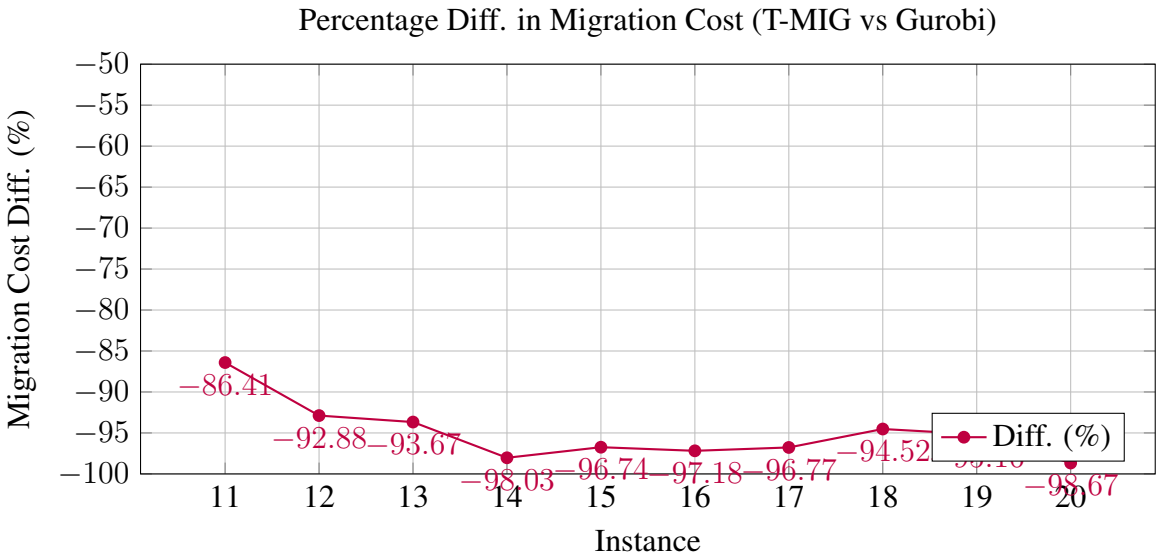


Figure 5.5 and 5.6 illustrates the absolute migration cost comparison between Gurobi and T-MIG on medium scale instances. Consistent with the observations made in small instances, T-MIG achieves significantly lower migration costs in every case. For example, while Gurobi incurs migration costs ranging from 969 to 4284 units, T-MIG limits this range to just 57 to 276 units. This clearly demonstrates the algorithm's continued success in reducing costly VM relocations at scale.

To quantify this improvement, Figure 5.6 presents the percentage reduction in migration cost achieved by T-MIG relative to Gurobi. The difference values vary between -86.41% and -98.67%, with an average of -95.09%. Compared to the -89.61% average reduction observed in small instances, this result shows that T-MIG not only maintains but even improves its migration efficiency as the problem size increases.

These findings confirm that T-MIG is highly effective at reducing migration overhead regardless of instance scale. The algorithm's strong preference for migration avoidance remains consistent across both small and medium scenarios, ensuring minimal disruption in dynamic environments and improving system reliability. Nevertheless, the slightly increased server usage costs observed in both scales indicate potential room for improvement in terms of consolidation efficiency.

5.2.3.3 Performance Evaluation on Large Instances

Table 5.14: *Comparison of T-MIG and Gurobi on Large Instances.*

Instance	Total Cost			Migration Cost			Usage Cost			Time (sec)	
	Gurobi*	T-MIG	Diff.	Gurobi*	T-MIG	Diff.	Gurobi*	T-MIG	Diff.	Gurobi*	T-MIG
21	42294	46255	9.37	17634	975	-94.47	24660	45280	83.62	3600	0.4
22	50403	49499	-1.79	6033	174	-97.12	44370	49325	11.17	3600	0.4
23	50310	49461	-1.69	7380	231	-96.87	42930	49230	14.68	3600	0.3
24	50629	49499	-2.23	7914	249	-96.85	42715	49250	15.3	3600	0.3
25	49093	49460	0.75	7443	210	-97.18	41650	49250	18.25	3600	0.3
26	50909	49421	-2.92	8154	231	-97.17	42755	49190	15.05	3600	0.4
27	76153	74421	-2.27	13398	231	-98.28	62755	74190	18.22	3600	0.4
28	74672	74537	-0.18	10722	252	-97.65	63950	74285	16.16	3600	0.5
29	75545	74828	-0.95	11760	198	-98.32	63785	74630	17.00	3600	0.5
30	76245	74941	-1.71	12210	66	-99.46	64035	74875	16.93	3600	0.5
Average			**-0.36**			**-97.33**			**22.63**		

*Gurobi execution time limited to 1 hour.

Table 5.14 presents the results of the T-MIG algorithm compared to the Gurobi solver on large scale VM placement instances. The evaluation follows the same structure as in previous sections, focusing on Total Cost, Migration Cost, Usage Cost, and Execution Time.

Total Cost: In contrast to small and medium instances, where Gurobi generally yielded lower total costs, the difference in large instances almost vanishes. The average total cost difference is only -0.36%, indicating that T-MIG produces solutions that are nearly equivalent to those of Gurobi in terms of total cost. This result highlights T-MIG's ability to scale effectively, maintaining solution quality even as the problem complexity increases.

Migration Cost: T-MIG continues to demonstrate a strong advantage in minimizing migration overhead. The average migration cost reduction reaches -97.34%, which surpasses the reduction rates in both small (-89.61%) and medium (-95.09%) instances. This improvement reaffirms that T-MIG is highly effective in avoiding unnecessary VM movements, making it well-suited for stability-critical or high-latency environments.

Usage Cost: As in previous scales, T-MIG incurs higher usage costs compared to Gurobi. The average usage cost increase in large instances rises to 22.64%, which is higher than the 14.20% observed in medium instances, but still slightly below the 24.48% in small instances. This again reflects the algorithm's design choice to sacrifice some consolidation opportunities in favor of minimizing migration cost, revealing a clear area for potential optimization.

Execution Time: T-MIG maintains its consistent speed advantage. While Gurobi is again limited to the externally imposed 3600 second time cap in all cases, T-MIG solves each instance in under 0.5 seconds. This significant time efficiency demonstrates the algorithm's practical value in large scale and time sensitive decision making scenarios.

In conclusion, T-MIG scales remarkably well across instance sizes. It maintains competitive total costs, achieves superior migration efficiency, and delivers solutions almost instantly. Although the increase in usage cost remains a trade off, it becomes more controlled at scale, and presents a meaningful direction for future enhancements.

Figure 5.7: *Migration Cost Comparison between Gurobi and T-MIG across Large Instances*

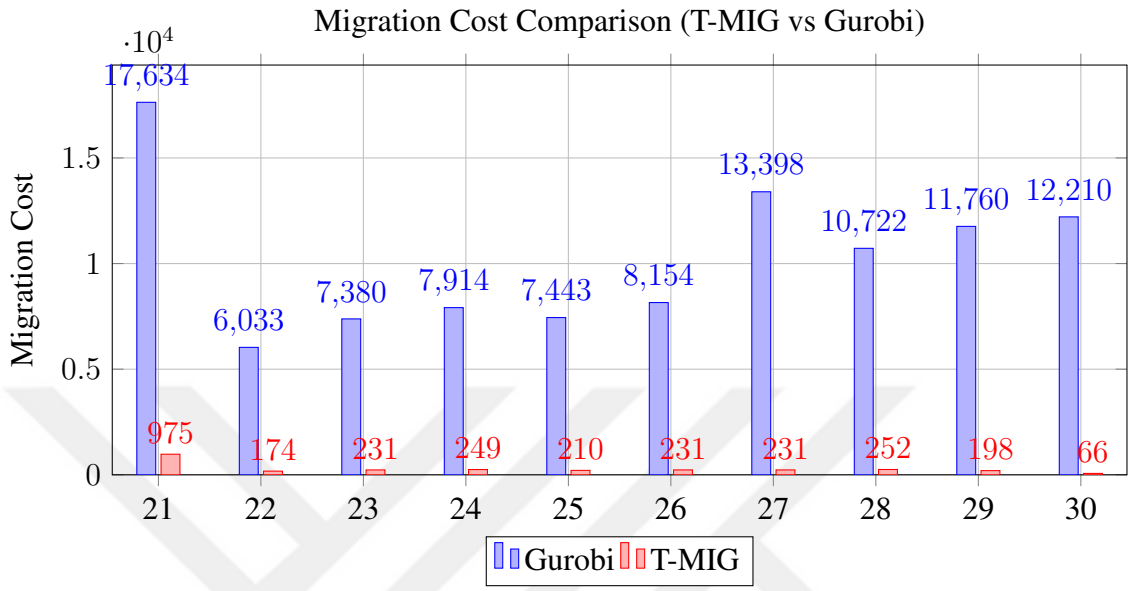


Figure 5.8: *Migration Cost Reduction (%) of T-MIG Compared to Gurobi (Large Instances)*

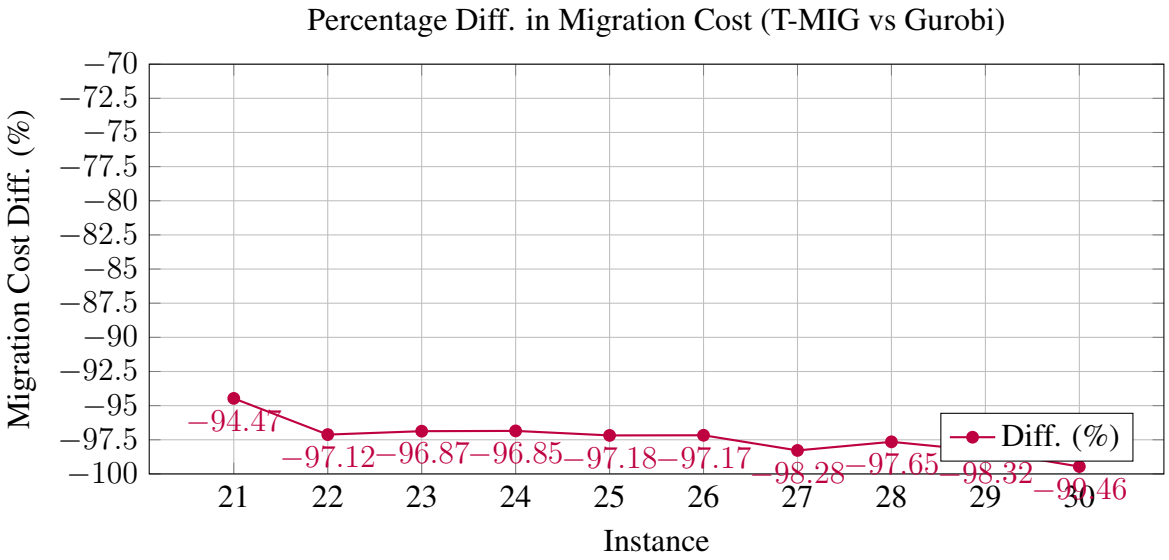


Figure 5.7 and 5.8 illustrates the migration costs of Gurobi and T-MIG across large scale instances. As in previous scales, T-MIG drastically reduces migration costs in all cases. For instance, while Gurobi's migration cost reaches values as high as 17,634 units, T-MIG consistently limits this to below 300, with some instances dropping as low as 66. This significant difference persists even as the number of VMs and time periods increases, further validating the scalability of the approach.

To provide a clearer view of this improvement, Figure 5.8 shows the percentage reduction in migration cost achieved by T-MIG relative to Gurobi. The difference values range from -94.47% to -99.46%, with an average of -97.34%, which is the highest among all three instance categories. This trend confirms that the migration avoidance strategy embedded in T-MIG becomes increasingly effective with larger problem sizes.

These results reinforce T-MIG's key strength: minimizing migration overhead. Such a reduction not only lowers operational cost but also helps maintain system stability and performance in large scale and dynamic cloud environments. However, as observed in earlier experiments, this improvement comes with an increase in usage cost, which remains a potential area for further optimization.

5.2.3.4 Performance Evaluation on Varying Usage Cost Instances

Table 5.15: *Comparison of T-MIG Variants with Gurobi on Medium Instances Using Usage Costs 25 and 1*

MEDIUM INSTANCES – MIG COST= 3 and USAGE COST= 25											
Instance	**Total Cost**			**Migration Cost**			**Usage Cost**			**Time (sec)**	
	Gurobi	T-MIG	Diff.	Gurobi	T-MIG	Diff.	Gurobi	T-MIG	Diff.	Gurobi*	T-MIG
31	62640	76551	22.21	2790	276	-90.11	59850	76275	27.44	3600	0.2
32	89519	98994	10.58	3144	69	-97.81	86375	98925	14.53	3600	0.2
33	72591	96943	33.55	3741	168	-95.51	68850	96775	40.56	3600	0.3
34	88151	99207	12.54	3426	57	-98.34	84725	99150	17.03	3600	0.3
35	86778	98791	13.84	3228	66	-97.96	83550	98725	18.16	3600	0.2
36	103996	123509	18.69	3996	84	-97.9	105400	123425	17.10	3600	0.2
37	111364	124291	11.81	3039	66	-97.83	108125	124225	14.89	3600	0.2
38	112328	122701	9.23	2928	126	-95.70	109400	122575	12.04	3600	0.2
39	112536	123509	9.75	2811	84	-97.01	109725	123425	12.49	3600	0.3
40	109278	124232	13.68	4878	57	-98.83	104400	124175	18.94	3600	0.2
Average			**15.00**			**-96.**			**19.31**		
MEDIUM INSTANCES – MIG COST= 3 and USAGE COST= 1											
Instance	**Total Cost**			**Migration Cost**			**Usage Cost**			**Time (sec)**	
	Gurobi	T-MIG	Diff.	Gurobi	T-MIG	Diff.	Gurobi	T-MIG v3	Diff.	Gurobi*	T-MIG
41	4471	3327	-25.59	1899	276	-85.47	2572	3051	18.62	3600	0.2
42	7106	4056	-42.92	3690	108	-97.07	3416	3948	15.57	3600	0.1
43	5142	4039	-21.45	2067	168	-91.87	3075	3871	25.89	3600	0.2
44	6347	4023	-36.81	2820	57	-97.98	3547	3966	11.81	3600	0.2
45	5941	4015	-32.42	2415	66	-97.27	3526	3949	12.00	3600	0.2
46	7193	5021	-30.2	2694	84	-96.88	4499	4937	9.74	3600	0.2
47	6304	5033	-20.13	1725	66	-96.17	4579	4969	8.52	3600	0.2
48	6696	5029	-24.89	2145	66	-96.93	4551	4937	8.47	3600	0.2
49	6273	5021	-19.96	1638	84	-94.87	4635	4937	6.52	3600	0.2
50	8654	5024	-41.95	4326	57	-98.68	4328	4967	14.76	3600	0.2
Average			**-29.63**			**-95.03**			**13.11**		

*Gurobi execution time limited to 1 hour.

To further explore the trade-off between migration cost and usage cost, two different usage strategies were evaluated on medium scale instances: Usage cost = 25 (favoring low migration) and Usage cost = 1 (favoring low usage cost). The results are shown in Table 5.15.

Usage cost = 25: As seen in instances 31 to 40, the configuration prioritizing migration cost minimization leads to a significant average reduction of -96.7% in migration cost. However, this comes at the expense of a 19.31% increase in usage cost and a 15.01% increase in total cost. These results confirm that aggressively avoiding VM migrations while beneficial for stability can lead to underutilization of servers and thus higher operational costs.

Usage cost = 1: In contrast, instances 41 to 50 adopt a configuration aimed at reducing usage cost. Interestingly, this setup still maintains a substantial migration cost reduction of -95.04%, while bringing the average usage cost difference down to 13.12%. Most notably, the total cost difference flips to a **–29.63% improvement**, indicating that this configuration strikes a more favorable balance between migration and consolidation.

These results highlight the flexibility of the T-MIG framework. By adjusting the relative weight of migration and usage cost in the objective function, the algorithm can be tuned to fit different operational priorities. While Usage 25 minimizes VM movements, Usage 1 delivers lower total costs, suggesting that a hybrid approach or adaptive weighting may yield even more effective results in practice.

Figure 5.9: *Migration Cost Comparison of Gurobi and T-MIG across High Usage Cost and Low Usage Costs*

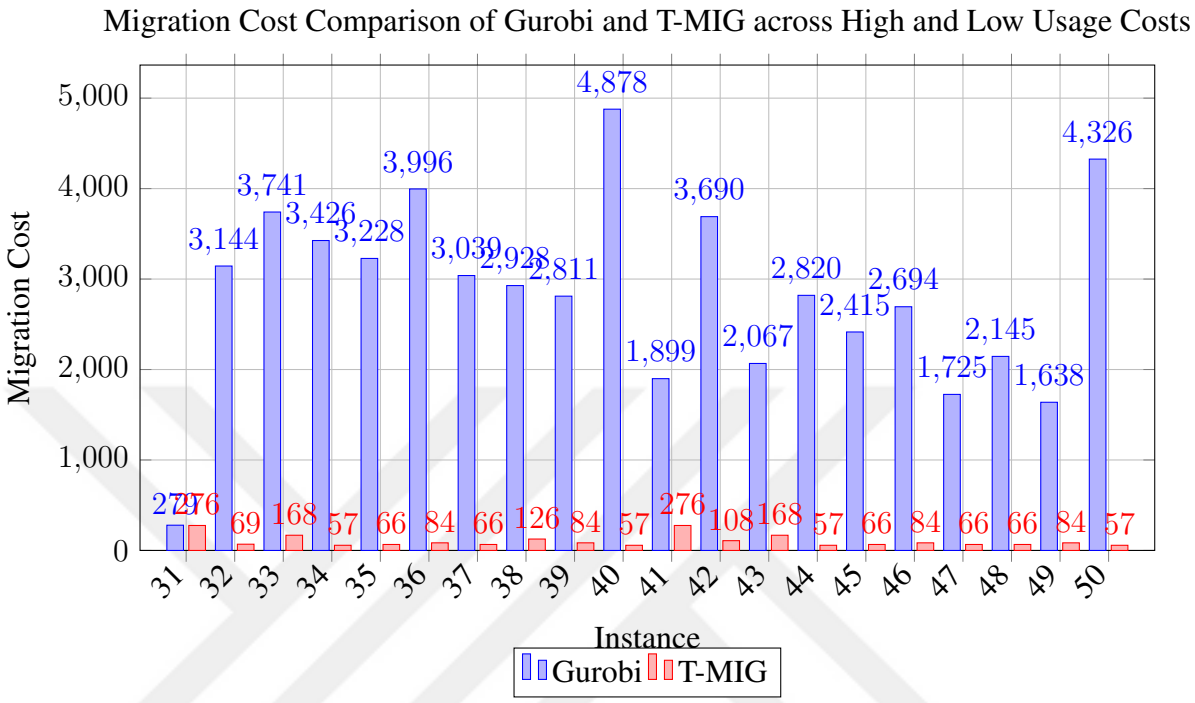


Figure 5.9 provides a side by side comparison of migration costs for Gurobi and T-MIG across two distinct configurations in medium-scale instances: Usage 25 (instances 31–40) and Usage 1 (instances 41–50). In both settings, T-MIG significantly outperforms Gurobi in minimizing migration overhead.

In the high usage configuration—where the objective function penalizes migration cost heavily—T-MIG achieves extremely low migration costs, often below 100 units. Gurobi, on the other hand, incurs values ranging from approximately 2,800 to nearly 4,900 units. This stark contrast highlights the effectiveness of T-MIG when migration minimization is prioritized.

In the low usage configuration, the optimization goal shifts towards lowering usage cost rather than migration cost. Despite this change, T-MIG maintains strong migration cost performance, with values still well below those of Gurobi. This is especially noteworthy, as it demonstrates that T-MIG is capable of preserving low migration behavior even when not explicitly instructed to prioritize it.

Overall, the graph confirms that T-MIG's architecture inherently favors stable VM allocations. While migration penalties can be tuned to emphasize or relax this behavior, the algorithm's default strategies continue to deliver significantly lower migration costs across different operational goals. This flexibility makes T-MIG a strong candidate for both consolidation-driven and stability-driven deployment environments.

5.2.4 Summary of Average Diff. Performance Across All Instances

Table 5.16: *Average Diff. (%) Comparison of T-MIG Against Gurobi Across All Instance Types*

Metric	Small	Medium	Large	High Usage Cost	Low Usage Cost
Total Cost Diff. (%)	13.42	1.34	-0.36	15.01	**-29.63**
Migration Cost Diff. (%)	-89.61	-95.09	-97.34	-96.70	-95.04
Usage Cost Diff. (%)	24.48	14.20	22.64	19.32	13.12

Table 5.16 presents the average Diff. (%) values in Total Cost, Migration Cost, and Usage Cost across all evaluated instance groups and parameter settings. This aggregated view provides a comprehensive understanding of the trade-offs made by the T-MIG algorithm under different configurations.

Migration Cost: Across all categories, T-MIG consistently and significantly reduces migration costs. The average Diff. improves from -89.61% in small instances to -95.09% in medium and peaks at -97.34% in large instances. These results confirm that migration minimization remains T-MIG's strongest attribute, even as problem complexity scales. Both usage-weighted variants in medium instances (Usage 25 and Usage 1) also maintain excellent performance in this regard.

Total Cost: The total cost behavior of T-MIG varies depending on the weighting strategy. While total cost slightly increases in the Usage 25 scenario and in small/medium instances, it becomes virtually equal to Gurobi in large instances. Most notably, the Usage 1 configuration leads to a remarkable -29.63% reduction in total cost, demonstrating that a usage-aware tuning can yield substantial overall efficiency gains without sacrificing migration performance.

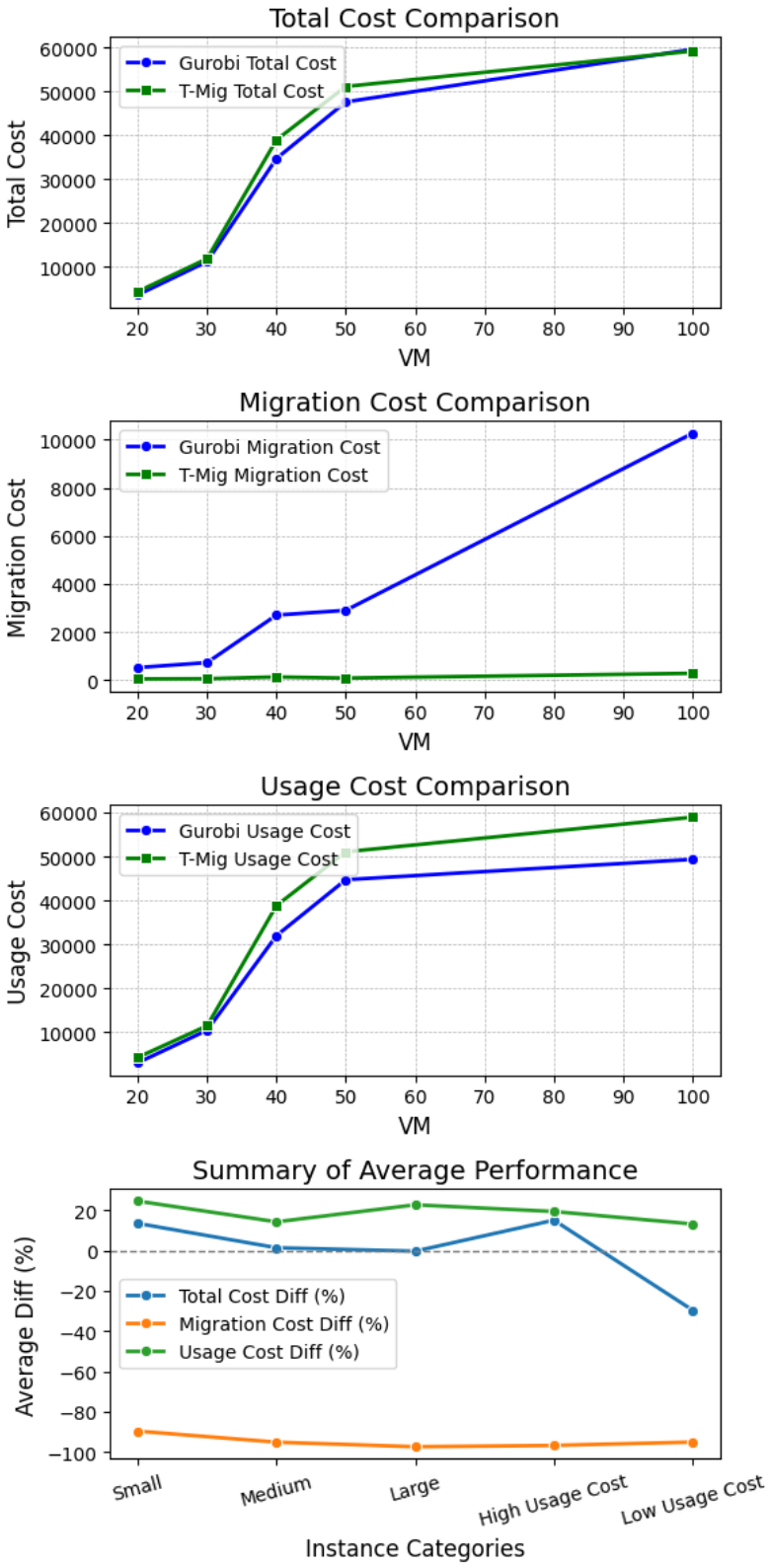
Usage Cost: As expected, usage cost is generally higher in T-MIG, especially when migration is aggressively minimized (e.g., Usage 25 and small instances). However, this difference becomes more controlled in larger instances and when the usage cost is prioritized (Usage 1). This shows that the trade-off between stability (less migration) and consolidation (lower usage) can be adjusted to suit different operational goals.

In summary, the results across all instance types and configurations highlight the flexibility and scalability of T-MIG. It consistently delivers extremely low migration costs and can be tuned to balance usage and total cost performance. Particularly in low usage cost scenarios, T-MIG demonstrates that it is capable of outperforming Gurobi even in total cost, making it a strong candidate for dynamic and large-scale environments.

Figure 5.10 compares the total, migration, and usage costs obtained from the Gurobi solver and our proposed heuristic T-MIG across different instance sizes. While Gurobi produces lower total costs in small-scale scenarios, the performance gap narrows

in larger instances, indicating the competitiveness of T-MIG. A detailed decomposition reveals that T-MIG consistently achieves significantly lower migration costs due to its migration-aware placement strategy. In contrast, its usage costs are slightly higher, as minimizing migrations sometimes requires activating more servers. The fourth subplot in Figure 5.10 summarizes the average percentage differences between the two methods across instance categories. T-MIG provides a remarkable improvement in migration costs up to 90% while maintaining acceptable trade offs in total cost. Although Gurobi remains advantageous in usage costs, the findings support that T-MIG offers a viable alternative for large scale, time extended placement problems where migration costs dominate.

Figure 5.10: *T-MIG and Gurobi: Total, Migration and Usage Cost Comparison*



6. CONCLUSION

This paper addresses the Virtual Machine Placement Problem under time varying workloads and migration costs, formally defined as the Temporal Bin Packing Problem with Migration Costs (TBPMC). Unlike traditional models that assume static demand or neglect migration costs, this study presents a comprehensive optimization framework that incorporates both temporal dynamics and cost awareness key elements in real world cloud environments.

To tackle the complexity and scalability limitations of exact optimization methods, we proposed a novel heuristic algorithm named T-MIG. T-MIG aims to minimize migration costs by first keeping as many VMs as possible on their current hosts and then assigning the remainder using a cost-based best-fit rule. Experimental results across small, medium, and large scale synthetic datasets generated to reflect mixed CPU demand patterns demonstrate the effectiveness of the proposed method. Compared to a Gurobi based exact solver, T-MIG achieves comparable or superior migration cost performance, especially under resource or time constraints.

The contribution of this work lies not only in its hybrid modeling of usage and migration costs, but also in the demonstration of how anticipatory placement decisions can reduce long term operational inefficiencies in cloud data centers. T-MIG offers a scalable, adaptive, and migration aware solution with strong potential for real time application in modern cloud infrastructures.

This study demonstrates that substantially lowering VM migration overhead enables cloud and data center operators to realize multiple system level benefits. Dynamically shifting VMs only when strictly necessary allows workloads to remain on under utilized servers just long enough to smooth out spikes, reducing hotspots and queuing delays.

By satisfying the throughput and latency improvement criteria documented in Xu et al. [35] and Li et al. [36], our approach can demonstrate measurable performance gains in multi-tenant cloud environments. By cutting migration counts by over 90 percent, T-MIG can effectively satisfy the pause time and packet loss constraints documented in Zhang et al. [37] and Wang et al. [38]. This dramatic reduction in live migration events can almost eliminate service interruptions, thereby resulting in fewer SLA violations, higher availability, and enhanced end user satisfaction.

The energy implications of our approach are particularly significant. Each migration involves CPU, memory, and network activity on both source and target hosts, incurring extra power draw. Reducing unnecessary migrations thus lowers overall energy consumption and with it, cooling and operational expenses helping green data center initiatives meet both economic and sustainability goals [39, 40]. Additionally, live migrations generate background traffic that can interfere with latency sensitive services. Fewer migrations free up network bandwidth for primary application traffic, improving end-to-end responsiveness and enabling tighter consolidation ratios without risking contention.

An often overlooked benefit is enhanced predictability and simplified capacity planning. When migration events become rare and more strategic, usage patterns stabilize. This makes it easier for operators to forecast resource needs, provision hardware, and schedule maintenance windows with minimal disruption. Overall, T-MIG's anticipatory placement logic and aggressive retention of existing VM to host assignments deliver a migration aware heuristic that is both scalable and practical for real time deployment. By balancing server usage and migration penalties, it not only matches—but often exceeds the total cost performance of a Gurobi based exact solver under tight time or resource constraints, while simultaneously unlocking the operational benefits described above.

6.1 Future Work

In future extensions, the usage cost component of the model could be optimized using new methods proportionally to migration costs. Additional techniques could be applied to adaptively tune usage cost weights based on observed workload patterns and energy consumption feedback. Exploring hybrid metaheuristic frameworks (e.g., Tabu Search or Genetic Algorithms) that jointly optimize migration and usage costs in a multi objective setting also presents an exciting research direction. Such improvements would likely further reduce operational costs and enhance the sustainability of cloud infrastructures.

REFERENCES

[1] A. Verma, P. Ahuja, and A. Neogi, "pmapper: Power and migration cost aware application placement in virtualized systems," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, vol. 5346 of *Lecture Notes in Computer Science*, pp. 243–264, Springer, 2008.

[2] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[3] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[4] A. K. Kiani and N. Ansari, "Toward optimal workload placement in geographically distributed data centers," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 50–61, 2012.

[5] H. Wu, C. Zhang, Z. Ren, and Y. Zhang, "Cost-efficient virtual machine placement in cloud data centers: A survey," *Cluster Computing*, vol. 22, no. 2, pp. 2241–2262, 2019.

[6] X.-F. Liu, Z.-H. Zhan, K.-J. Du, and W.-N. Chen, "Energy aware virtual machine placement scheduling in cloud computing based on ant colony optimization approach," in *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, pp. 41–48, 2014.

[7] F. Song, D. Huang, H. Zhou, H. Zhang, and I. You, "An optimization-based scheme for efficient virtual machine placement," *International Journal of Parallel Programming*, vol. 42, pp. 853–872, 2014.

[8] H. D. Rjeib and G. Kecskemeti, "Vmp-er: An efficient virtual machine placement algorithm for energy and resources optimization in cloud data center," *Algorithms*, vol. 17, no. 7, p. 295, 2024.

[9] Z. Mahmoodabadi and M. Nouri-Baygi, "An approximation algorithm for virtual machine placement in cloud data centers," *The Journal of Supercomputing*, vol. 80, no. 1, pp. 915–941, 2024.

[10] M. S. Atchukatla, "Algorithms for efficient vm placement in data centers - cloud based design and performance analysis," *Journal Unknown*, Year Unknown.

[11] H. Tian, J. Wu, and H. Shen, "Efficient algorithms for vm placement in cloud data centers," *IEEE Access*, 2018.

[12] F. Alharbi, Y.-C. Tian, M. Tang, W.-Z. Zhang, C. Peng, and M. Fei, "An ant colony system for energy-efficient dynamic virtual machine placement in data centers," *Journal Unknown*, Year Unknown.

[13] A. Al-Moalmi, J. Luo, A. Salah, K. Li, and L. Yin, "A whale optimization system for energy-efficient container placement in data centers," *Journal Unknown*, Year Unknown.

[14] T. Ganesan, P. Vasant, and I. Litvinchev, "Chaotic simulator for bilevel optimization of virtual machine placements in cloud computing," *Journal Unknown*, Year Unknown.

[15] B. Sun, G. Li, S. Wang, and C. Xie, "Two-dimensional bin-packing problem with conflicts and load balancing: A hybrid chaotic and evolutionary particle swarm optimization algorithm," *Journal Unknown*, Year Unknown.

[16] S. Azizi, M. Shojafar, J. Abawajy, and R. Buyya, "Grvmp: A greedy randomized algorithm for virtual machine placement in cloud data centers," *Journal Unknown*, Year Unknown.

[17] S. Kumaraswamy and M. K. Nair, "Bin packing algorithms for virtual machine placement in cloud computing: a review," *Journal Unknown*, Year Unknown.

[18] T. Hage, K. Begnum, and A. Yazidi, "Saving the planet with bin packing - experiences using 2d and 3d bin packing of virtual machines for greener clouds," *Journal Unknown*, Year Unknown.

[19] A. A. Pandey, "An analysis of solutions to the 2d bin packing problem and additional complexities," *Journal Unknown*, Year Unknown.

[20] A. Fatima, N. Javaid, T. Sultana, W. Hussain, M. Bilal, S. Shabbir, Y. Asim, M. Akbar, and M. Ilahi, "Virtual machine placement via bin packing in cloud data centers," *Journal Unknown*, Year Unknown.

[21] L. Wei, M. Lai, A. Lim, and Q. Hua, "A branch-and-price algorithm for the two-dimensional vector packing problem," *Journal Unknown*, Year Unknown.

[22] C. Muir, L. Marshall, and A. Toriello, "Temporal bin packing with half-capacity jobs," *INFORMS Journal on Optimization*, 2023.

[23] N. Aydın, İ. Muter, and Ş. İ. Birbil, "Multi-objective temporal bin packing problem: An application in cloud computing," *Computers & Operations Research*, vol. 121, p. 104959, 2020.

[24] T. D. Cauwer, T. V. de Voorde, R. Berkvens, P. Hellinckx, and W. Joosen, "The temporal bin packing problem: An application to workload management in data centers," *Journal of Scheduling*, 2016.

[25] A. Strunk, "Costs of virtual machine live migration: A survey," in *2012 IEEE Eighth World Congress on Services*, pp. 323–329, 2012.

[26] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation*, vol. 2 of *NSDI'05*, pp. 273–286, USENIX Association, 2005.

[27] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pp. 153–167, 2017.

[28] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *2010 Proceedings IEEE INFOCOM*, pp. 1–9, 2010.

[29] S. Wang, W. Wu, J. Wang, H. Dai, and D. Wang, "A sla-aware virtual machine management for dynamic consolidation in cloud environment," *Future Generation Computer Systems*, vol. 54, pp. 95–107, 2015.

[30] T. Zhang, N. Antunes, L. Huang, and X. He, "Analyzing performance interference in shared virtual computing environments with applications to cloud computing," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 922–935, 2018.

[31] Z. Á. Mann, "Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms," *ACM Computing Surveys*, vol. 48, no. 1, pp. 11:1–11:34, 2015.

[32] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, pp. 171–182, ACM, 2011.

[33] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the performance of virtual machine migration," in *IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 37–46, IEEE, 2010.

[34] Ö. Özbay *et al.*, "Resource allocation considering impact of network on performance in a disaggregated data center," in *2024 International Conference on Computer, Information and Telecommunication Systems (CITS)*, IEEE, 2024.

[35] X. Xu, H. Zhang, *et al.*, "Adaptive vm migration strategy for multi-tenant cloud environments," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1105–1118, 2021.

[36] Y. Li, C. Wang, *et al.*, “Dynamic resource allocation with vm migration for data-intensive applications,” *Journal of Systems Architecture*, vol. 108, p. 101735, 2020.

[37] J. Zhang, K. Huang, *et al.*, “Impact analysis of vm live migration on application performance and qos,” *IEEE Access*, vol. 6, pp. 41676–41691, 2018.

[38] L. Wang, F. Zhang, *et al.*, “Live migration downtime analysis for cloud-based systems,” *IEEE Cloud Computing*, vol. 6, no. 2, pp. 30–42, 2019.

[39] H. Chen, X. Liu, *et al.*, “Energy-aware vm consolidation and migration in sustainable cloud data centers,” *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 186–200, 2017.

[40] R. Patel, S. Sharma, *et al.*, “Power-aware vm migration techniques for energy-efficient cloud computing,” *Journal of Green Engineering*, vol. 6, no. 2, pp. 145–170, 2016.