*computers*

MDPI

*Article*

# Optimizing Scheduled Virtual Machine Requests Placement in Cloud Environments: A Tabu Search Approach

**Mohamed Koubàa [1,*,†], Abdullah S. Karar [2,†] and Faouzi Bahloul [3,†]**

[1] SYSCOM Laboratory, National Engineering School of Tunis (ENIT), University of Tunis El Manar, Tunis 1002, Tunisia
[2] College of Engineering and Technology, American University of the Middle East, Egaila 15453, Kuwait; abdullah.karar@aum.edu.kw
[3] SERCOM Laboratory, Tunisia Polytechnic School (EPT), University of Carthage, Tunis 2078, Tunisia; faouzi.bahloul@enit.utm.tn
* Correspondence: mohamed.koubaa@enit.utm.tn
† These authors contributed equally to this work.

**Abstract:** This paper introduces a novel model for virtual machine (VM) requests with predefined start and end times, referred to as scheduled virtual machine demands (SVMs). In cloud computing environments, SVMs represent anticipated resource requirements derived from historical data, usage trends, and predictive analytics, allowing cloud providers to optimize resource allocation for maximum efficiency. Unlike traditional VMs, SVMs are not active concurrently. This allows providers to reuse physical resources such as CPU, RAM, and storage for time-disjoint requests, opening new avenues for optimizing resource distribution in data centers. To leverage this opportunity, we propose an advanced VM placement algorithm designed to maximize the number of hosted SVMs in cloud data centers. We formulate the SVM placement problem (SVMPP) as a combinatorial optimization challenge and introduce a tailored Tabu Search (TS) meta-heuristic to provide an effective solution. Our algorithm demonstrates significant improvements over existing placement methods, achieving up to a 15% increase in resource efficiency compared to baseline approaches. This advancement highlights the TS algorithm's potential to deliver substantial scalability and optimization benefits, particularly for high-demand scenarios, albeit with a necessary consideration for computational cost.

**Keywords:** cloud computing; scheduled virtual machines requests; placement problem; combinatorial optimization; integer linear programming; Tabu Search; Ant Colony Optimization; Particle Swarm Optimization



**Citation:** Koubàa, M.; Karar, A.S.; Bahloul, F. Optimizing Scheduled Virtual Machine Requests Placement in Cloud Environments: A Tabu Search Approach. *Computers* **2024**, *13*, 321. https://doi.org/10.3390/computers13120321

Academic Editor: Pedro Alonso Jordá

Received: 21 October 2024
Revised: 21 November 2024
Accepted: 26 November 2024
Published: 2 December 2024



## 1. Introduction

Cloud computing has revolutionized how organizations deploy and manage IT resources, offering scalable, flexible, and cost-effective solutions via on-demand access to shared computing infrastructures. This paradigm shift has enabled businesses to support a wide range of applications, spanning traditional enterprise systems to cutting-edge cloud-native, serverless, and edge computing environments [1]. The adaptability and breadth of these applications are underpinned by distributed architectures and microservices, which necessitate rapid provisioning and de-provisioning of virtual machines (VMs) to maintain optimal service performance, responsiveness, and availability [2].

As cloud-native applications continue to gain traction, they bring about new challenges and demands in efficiently managing resources within cloud data centers. A key challenge in this context is the virtual machine placement problem (VMPP), an essential optimization task that maps VMs onto physical machines (PMs) to achieve multiple objectives: maximizing resource utilization, minimizing operational costs, and ensuring compliance with strict service quality requirements [3,4]. Each VM comes with specific

resource demands—such as CPU, memory, and storage—which must be allocated within the capacity constraints of available PMs.

However, traditional approaches to solving the VMPP, often based on static or random VM request patterns, fall short in addressing the complexities of modern cloud environments. Today's workloads are increasingly characterized by predictable and time-bound patterns that necessitate dynamic and adaptive placement strategies to ensure both efficiency and performance [5]. Addressing these evolving demands requires innovative solutions that account for the temporal and operational intricacies of cloud-native ecosystems.

This paper introduces a novel VM request model, termed scheduled virtual machine requests (SVMs). SVMs are a distinct class of VM requests defined by predictable start and end times derived from historical usage data. This predictability allows cloud providers to proactively allocate resources, reducing idle capacity and improving overall resource utilization. Unlike conventional VM requests, SVMs enable operators to better anticipate demand, offering an optimized approach to managing resources for time-sensitive applications—a dimension not fully explored in current models.

SVMs bring distinct benefits to time-sensitive cloud applications, especially in serverless and edge computing environments. In serverless systems, where users are abstracted from infrastructure management, SVM predictability enhances response times and reduces latency. For edge computing, SVM-based resource allocation enables efficient handling of latency-sensitive workloads across distributed locations, ensuring timely processing and resource availability. The ability to forecast and allocate resources for SVMs aligns with service-level agreements (SLAs) and supports dynamic applications with minimal idle resource costs.

To address the scheduled virtual machine placement problem (SVMPP), this paper proposes a Tabu Search (TS)-based meta-heuristic specifically designed for SVM placement. TS is an advanced optimization technique widely recognized for its effectiveness in solving complex optimization problems [6]. It enhances local search methods through a memory-based approach, enabling it to escape local optima. By systematically exploring the solution space, it allows moves that may temporarily worsen the objective while maintaining a tabu list—a short-term memory structure that prevents revisiting recently explored solutions or moves. This ensures diversification and guides the search toward high-quality solutions [7,8].

The primary contributions of this work are as follows:

1. Formalization of the scheduled VM model:
    - The concept of SVMs is introduced and formalized, with each request defined by its start and end times and specific resource requirements.
    - This model bridges a critical gap in VM scheduling research by enabling precise resource allocation for time-bound and predictable workloads.
2. Development of a TS-based meta-heuristic for SVMPP:
    - A novel Tabu Search algorithm is tailored to solve the SVMPP, leveraging its adaptive local search capabilities and memory-based mechanisms to avoid local optima.
    - The algorithm incorporates temporal constraints and resource requirements, demonstrating scalability and adaptability for dynamic cloud resource allocation.
    - The performance of the TS algorithm is rigorously evaluated through a comprehensive comparison with existing placement methods from the literature, highlighting its efficiency and effectiveness.
3. Advancing optimization in VM placement:
    - The proposed TS-based approach demonstrates clear superiority over traditional VM placement methods, particularly in terms of solution quality.
    - This work represents, to the best of our knowledge, one of the first applications of TS in the context of VM placement, laying a strong foundation for future research and positioning TS as a robust tool for tackling large-scale optimization challenges in cloud computing.

The remainder of this paper is organized as follows: Section 2 provides an in-depth description of SVMPP. Section 3 reviews the relevant literature. Section 4 presents the mathematical model for SVMPP. Section 5 details the TS-based solution, and Section 6 evaluates its effectiveness. Section 7 concludes the study and discusses future work.

## 2. Problem Description

The SVMPP involves efficiently assigning CPU, RAM, and storage resources on a set of PMs to accommodate incoming SVM requests. Each SVM request has defined CPU, memory, and storage needs as well as a specific start and end time, posing the challenge of ensuring that PM resources are allocated without exceeding capacity and are available precisely when required.

To maximize resource utilization and minimize operational costs, the goal of SVMPP is to optimize the placement strategy so that the maximum number of SVMs can be accommodated within the available resources. Each PM has finite capacities, requiring careful allocation of CPU, memory, and storage for each SVM without overcommitting. Moreover, each SVM's timing constraints must be respected, ensuring that resources are available for each request only within its specified time window.

An SVM request is defined by a tuple $(c, r, s, \alpha, \beta)$, where $c$, $r$, and $s$ represent the CPU, memory, and storage requirements, respectively, and $\alpha$ and $\beta$ denote the start and end times. Table 1 provides an example set of SVMs, highlighting the timing constraints central to this model. The SVMPP accounts for both simultaneous and time-disjoint SVM requests. For instance, SVM $v_1$ and SVM $v_2$ do not overlap and can share the same PM resources at different times, whereas SVMs $v_1$ and $v_3$ overlap in time, preventing them from using the same PM resources concurrently.

**Table 1.** Example of four scheduled VM requests (SVMs).

| SVM | CPU (Cores) | RAM (GB) | Storage (GB) | Start Time | End Time | SVMs with Time Overlap |
|---|---|---|---|---|---|---|
| $v_1$ | 4 | 16 | 100 | 08:00 | 12:00 | $v_3$, $v_4$ |
| $v_2$ | 8 | 24 | 200 | 13:00 | 17:00 | $v_4$ |
| $v_3$ | 2 | 8 | 50 | 10:00 | 13:00 | $v_1$, $v_4$ |
| $v_4$ | 16 | 64 | 300 | 09:00 | 17:00 | $v_1$, $v_2$, $v_3$ |

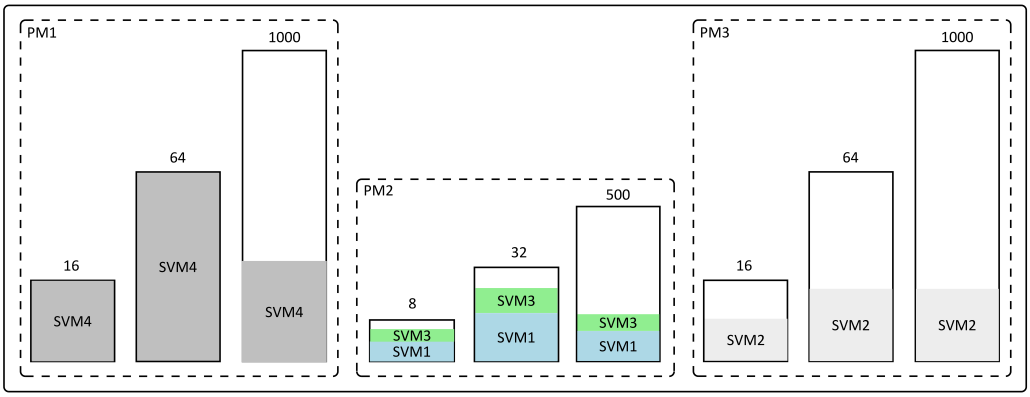
The time disjointness property of SVMs refers to their non-overlapping time windows, enabling cloud providers to optimize PM usage by reusing resources across different time slots. This strategy maximizes resource utilization and reduces the number of active PMs required, thereby improving data center (DC) efficiency. Figure 1 illustrates this optimization through a comparison of two placement strategies for the set of SVMs described in Table 1. In this example, three PMs in the DC are considered, with their characteristics summarized in Table 2 (provided for illustration and not based on actual server configurations):

- Strategy 1: Hosts SVM $v_4$ on PM1, SVMs $v_1$ and $v_3$ on PM2, and SVM $v_2$ on PM3. This approach results in three active PMs.
- Strategy 2: Enhances resource efficiency by hosting SVM $v_4$ on PM1, while reusing resources on PM2, initially used by SVMs $v_1$ and $v_3$, to service SVM $v_2$. Since SVMs $v_1$ and $v_3$ are time-overlapping, they cannot share the same resources. However, SVM $v_2$, being time-disjoint from $v_1$ and $v_3$, can use the resources released by these two SVMs at different time slots. This approach leaves PM3 idle, demonstrating how resource reuse across non-overlapping time-slot windows reduces the number of active PMs.

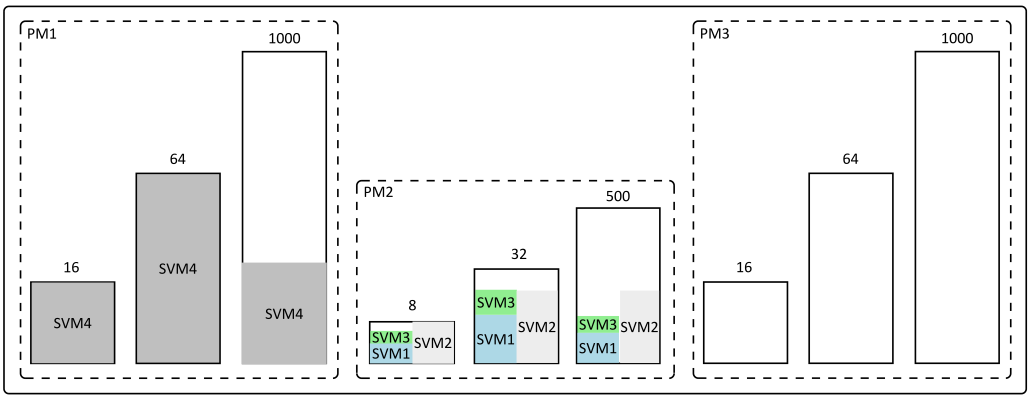
By leveraging time-disjoint SVM requests, the SVMPP allows cloud providers to meet service demands with fewer resources, enhancing operational efficiency and enabling better resource planning for scheduled requests. This approach optimizes the DC workload by increasing the acceptance rate of SVM requests, and creating flexibility for handling unpredictable demand spikes.

**Table 2.** PM resource characteristics.

| | CPU (Cores) | RAM (GB) | Storage (GB) |
|---|---|---|---|
| PM1 | 16 | 64 | 1000 |
| PM2 | 8 | 32 | 500 |
| PM3 | 16 | 64 | 1000 |




(**a**) Placement strategy 1: Without considering SVMs' time disjointness




(**b**) Placement strategy 2: Exploiting SVMs' time disjointness

**Figure 1.** Enhancing VM placement efficiency by exploiting SVMs' time disjointness.

## 3. Related Work

In this section, we review related work to provide context and background for our study, focusing first on the objectives of VM placement and the underlying workload models, before addressing the methods employed to solve the VMPP.

The VMPP has been the subject of extensive research, with studies addressing diverse objectives to meet the requirements of cloud providers:

- Resource utilization: Maximizing the efficient use of physical resources such as CPU, memory, storage, and network bandwidth while avoiding both under-utilization and contention, which can lead to inefficiency and increased operational costs [9–14].
- Energy efficiency: Reducing power consumption through strategies such as VM consolidation, which minimizes the number of active PMs by co-locating VMs, and switching off or putting idle PMs into low-power states, thereby lowering energy costs and environmental impact [15–18].
- Quality of service (QoS): Ensuring high QoS by minimizing latency, reducing performance degradation, and avoiding SLA violations, all of which are critical for user satisfaction [19–21].
- Load balancing: Preventing resource bottlenecks by distributing workloads evenly across PMs, thereby improving overall system performance and response times [22–25].

- Cost minimization: Optimizing resource allocation to reduce both capital (CAPEX) and operational (OPEX) expenditures, including network overheads and VM migration costs [26–28].
- Fault tolerance and reliability: Enhancing system robustness by accounting for PM failures, creating redundancy, and enabling seamless recovery mechanisms to minimize service disruptions [29–31].

Achieving these objectives depends on the underlying workload models, which define the nature and variability of VM requests. These models provide valuable insights into resource demand patterns and inform the development of placement strategies:

- Static request models: These models involve requesting VMs with predetermined and unchanging resource specifications, such as specific amounts of CPU, memory, and storage. They simplify resource allocation but lack flexibility in handling varying workload demands.
- Dynamic request models: These models account for changes in VM requests over time, reflecting the dynamic nature of workloads that fluctuate based on time, user activity, or external factors. They require adaptability in resource management to respond to varying demands effectively.
- Probabilistic workloads: Requests are generated based on probabilistic distributions (such as Poisson or Gaussian), capturing the uncertainty and variability inherent in demand. This model enables more robust resource allocation strategies that can withstand fluctuations in workload.
- Multi-tier request models: VM requests are made with specific performance and availability requirements outlined in SLAs. This model ensures that resource allocation aligns with contractual obligations to users, emphasizing the need for reliability and quality of service.

These diverse objectives and workload models underscore the complexity and multi-faceted nature of the VMPP, requiring advanced, often hybrid, approaches to achieve an optimal balance between performance, cost, and reliability. Table 3 presents a summary of the various methods, highlighting the specific findings and research gaps identified.

**Table 3.** VM placement methods, findings, and research gaps.

| Method | Specific Findings | Research Gaps | References |
| --- | --- | --- | --- |
| Deterministic methods | Techniques such as linear programming, integer programming, and constraint programming yield precise, optimal solutions; effective for small-scale problems with limited resources. | High computational cost and scalability limitations make them impractical for large-scale DCs; require simplifications or hybridization for broader applicability. | [32–38] |
| Heuristic methods | Methods like best-fit, first-fit, and worst-fit offer fast, near-optimal solutions; practical for real-time deployment where speed is prioritized over absolute optimality. | Solution quality varies widely based on the problem instance; lacks adaptivity to dynamic workloads, which limits their flexibility. | [39–42] |
| Meta-heuristic methods | Approaches such as Genetic Algorithms, Simulated Annealing, Particle Swarm Optimization, and Ant Colony Optimization effectively explore large solution spaces, yielding near-optimal results. | Risk of local optimality and high computational costs; scalability remains challenging for extensive DCs due to prolonged runtime. | [17,36,43–62] |
| Machine learning-based methods | Machine learning techniques (e.g., reinforcement learning, clustering, predictive modeling) enable adaptive placement based on historical data, improving allocation accuracy over time. | Dependence on large datasets and model generalization limits their effectiveness in heterogeneous and highly variable environments; can struggle with real-time adaptability. | [14,63–67] |

**Table 3.** *Cont.*

| Method | Specific Findings | Research Gaps | References |
| --- | --- | --- | --- |
| Game theory and auction-based methods | Game-theoretic approaches model VM placement as a resource allocation market, allowing structured bidding mechanisms that optimize resource usage. | High complexity and scalability issues in dynamic cloud environments; may not be flexible enough to adapt to fluctuating resource demands in real time. | [20,68–71] |
| Hybrid methods | Hybrid methods combine multiple strategies to address multi-objective optimization, resulting in robust, scalable solutions that handle complex VMPP scenarios effectively. | Increased complexity and computational overhead hinder real-time practicality; tuning parameters for efficiency remains challenging. | [72–76] |

## 4. Formalization of the Problem

### *4.1. Formulating the Model*

We formalize the considered SVM problem as a combinatorial optimization problem. Below is the notation used in the model.

- Set of PMs:
  Let $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M\}$ denote the set of available PMs in the DC. Each PM $\mathcal{P}_j$ is defined by its resources:
  $$\mathcal{P}_j = \{C_j, R_j, S_j\}$$
  where
  - $C_j$: Initial CPU capacity of $\mathcal{P}_j$;
  - $R_j$: Initial memory capacity of $\mathcal{P}_j$;
  - $S_j$: Initial storage capacity of $\mathcal{P}_j$.
- Set of SVM requests:
  Let $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ represent the set of all SVM requests received by the DC. Each SVM $v_i$ is defined by
  $$v_i = \{c_i, r_i, s_i, \alpha_i, \beta_i\}$$
  where
  - $c_i$: CPU requirements of SVM $v_i$;
  - $r_i$: Memory requirements of SVM $v_i$;
  - $s_i$: Storage requirements of SVM $v_i$;
  - $\alpha_i$: Start time of SVM $v_i$;
  - $\beta_i$: End time of SVM $v_i$.
- Number of PMs and SVMs:
  Let $M = |\mathcal{P}|$ be the number of PMs available in the DC and $N = |\mathcal{V}|$ be the number of SVM requests.
- Temporal relationship between SVMs:
  Let $\theta = (\theta_{ij})$ be a $\{0,1\}^{M \times N}$ lower triangular matrix; $\theta_{ij}$ indicates whether the SVMs $v_i$ and $v_j$ overlap in time ($\theta_{ij} = 1$) or not ($\theta_{ij} = 0$). By defining $\theta_{ii} = 1$, $1 \leq i \leq N$, and $\theta_{ij} = 0$ for $i < j$.
- $(\mathcal{P}, \mathcal{V})$ is a pair representing an instance of the SVMPP.
- $P_{j,i}$, $1 \leq j \leq M$, $1 \leq i \leq N$ represents the j-th PM in $\mathcal{P}$ hosting SVM $v_i$.
- $\pi_{\kappa,\mathcal{V}} = \{P_{\kappa_1,1}, P_{\kappa_2,2}, \ldots, P_{\kappa_N,N}\}$, $\kappa \in \{1, \ldots, M\}^N$ is called an admissible SVM placement solution for $\mathcal{V}$. Here, $\kappa$ is an N-dimensional vector whose elements can take values between 1 and $M$. An admissible SVM placement solution is fully characterized by $\kappa$. The $\kappa$ for the solution of Figure 1a is (2, 3, 2, 1) and (2, 2, 2, 1) for the solution of Figure 1b.
- $\Pi_{\mathcal{V}} = \{\pi_{\kappa,\mathcal{V}}, \kappa \in \{1, \ldots, M\}^N\}$ is the set of admissible solutions for $\mathcal{V}$. There are $|\Pi_{\mathcal{V}}| = M^M$ possible admissible SVM placement solutions.

- $\mathcal{C} : \Pi_{\mathcal{V}} \to \mathbb{N}$ is the cost function that computes the number of hosted SVMs by an admissible SVM placement solution $\pi_{\kappa,\mathcal{V}}$. In such a solution, each SVM in $\mathcal{V}$ is hosted by at most one PM in $\mathcal{P}$. The same resources on a PM can be used by two or more SVMs that do not overlap in time.

Table 4 provides a summary of the notation used in the SVMPP model, offering a quick reference for each parameter and its function within the optimization model. The corresponding combinatorial optimization problem is as follows:

$$\textbf{Maximize} \quad \mathcal{C}(\pi_{\kappa,\mathcal{V}}) \tag{1}$$

**subject to**

$$\pi_{\kappa,\mathcal{V}} \in \Pi_{\mathcal{V}} \tag{2}$$

that is, we are interested in finding an admissible SVM placement solution of maximum cost for the set of SVM requests $\mathcal{V}$. To formalize the cost function $\mathcal{C} : \Pi_{\mathcal{P}} \to \mathbb{N}$, we define the following additional notation:

- Let $x_i$ be a binary decision variable that indicates whether SVM $v_i$ is hosted. $x_i = 1$ if SVM $v_i$ is assigned to any PM $\mathcal{P}_j$, and $x_i = 0$ otherwise.
- Let $\phi_{ji}$ be a binary decision variable that indicates whether SVM $v_i$ is hosted on PM $\mathcal{P}_j$. $\phi_{ji} = 1$ if SVM $v_i$ is hosted by PM $\mathcal{P}_j$, and $\phi_{ji} = 0$ otherwise.

The complete model can be summarized as
**Objective**

$$\text{Maximize} \quad \sum_{i=1}^{N} x_i \tag{3}$$

**Constraints**

- Single-host constraint: Ensure that each SVM $v_i$ is hosted by at most one PM, $\mathcal{P}_j \in \mathcal{P}$ or not hosted:

$$\sum_{j=1}^{M} \phi_{ji} \leq x_i, \quad \forall 1 \leq i \leq N \tag{4}$$

- Resource capacity constraints: For each PM $\mathcal{P}_j$, the total resource demand of SVMs hosted by that PM must not exceed its available resources. If two SVMs overlap in time ($\theta_{ik} = 1$), they cannot share resources on the same PM.
  - CPU constraint:

$$\sum_{i=1}^{N} \left( c_i \cdot \phi_{j,i} + \sum_{k=1}^{i-1} \theta_{i,k} \cdot c_k \cdot \phi_{j,k} \right) \leq C_j, \quad \forall 1 \leq j \leq M \tag{5}$$

  - Memory (RAM) constraint:

$$\sum_{i=1}^{N} \left( r_i \cdot \phi_{j,i} + \sum_{k=1}^{i-1} \theta_{i,k} \cdot r_k \cdot \phi_{j,k} \right) \leq R_j, \quad \forall 1 \leq j \leq M \tag{6}$$

  - Storage constraint:

$$\sum_{i=1}^{N} \left( s_i \cdot \phi_{j,i} + \sum_{k=1}^{i-1} \theta_{i,k} \cdot s_k \cdot \phi_{j,k} \right) \leq S_j, \quad \forall 1 \leq j \leq M \tag{7}$$

- Non-overlapping SVMs sharing resources: If two SVMs $v_i$ and $v_k$ do not overlap in time ($\theta_{i,k} = 0$), they can share resources on the same PM:

$$c_i \cdot \phi_{ji} + c_k \cdot \phi_{jk} \leq C_j, \quad \forall 1 \leq j \leq M, \forall 1 \leq i,k \leq N \text{ such that } \theta_{ik} = 0 \tag{8}$$

$$r_i \cdot \phi_{ji} + r_k \cdot \phi_{jk} \leq R_j, \quad \forall 1 \leq j \leq M, \forall 1 \leq i,k \leq N \text{ such that } \theta_{ik} = 0 \tag{9}$$

$$s_i \cdot \phi_{ji} + s_k \cdot \phi_{jk} \leq S_j, \quad \forall 1 \leq j \leq M, \forall 1 \leq i,k \leq N \text{ such that } \theta_{ik} = 0 \tag{10}$$

- SVM hosting indicator: Ensure that if SVM $v_i$ is hosted by any PM, then $x_i = 1$:

$$x_i \leq \max_{\forall 1 \leq j \leq M} \phi_{ji}, \quad \forall 1 \leq i \leq N \tag{11}$$

**Table 4.** Notation used in the SVMPP.

| Notation | Description |
|---|---|
| $\mathcal{P}$ | Set of PMs in the DC |
| $\mathcal{P}_j$ | A PM characterized by $\mathcal{P}_j = \{C_j, R_j, S_j\}$ |
| $C_j$, $R_j$, $S_j$ | Initial CPU, memory, and storage capacities of PM $\mathcal{P}_j$ |
| $M = \lvert\mathcal{P}\rvert$ | Number of available PMs |
| $\mathcal{V}$ | Set of SVMs |
| $v_i$ | An SVM request defined by $v_i = \{c_i, r_i, s_i, \alpha_i, \beta_i\}$ |
| $c_i$, $r_i$, $s_i$ | CPU, memory, and storage requirements of SVM $v_i$ |
| $\alpha_i$, $\beta_i$ | Start and end times of SVM $v_i$ |
| $N = \lvert\mathcal{V}\rvert$ | Number of SVM requests |
| $P_{j,i}$ | Indicates if PM $\mathcal{P}_j$ is hosting SVM $v_i$ |
| $\theta = (\theta_{ij})$ | Binary lower triangular matrix indicating temporal overlap of SVMs |
| $x_i$ | Binary decision variable: 1 if SVM $v_i$ is hosted, 0 otherwise |
| $\phi_{ji}$ | Binary decision variable: 1 if SVM $v_i$ is hosted on PM $\mathcal{P}_j$ |

### *4.2. Characterization of Problem Instances*

We can characterize the problem instances according to their time correlation using a normalized time correlation measure. The normalized time correlation $\tau(\mathcal{V})$ for a set of SVMs $\mathcal{V}$ is defined as

$$\tau(\mathcal{V}) = \frac{2 \cdot \sum_{i=1}^{N} \sum_{j=1}^{i-1} \theta_{ij}}{N(N-1)} \tag{12}$$

The numerator $\sum_{i=1}^{N} \sum_{j=1}^{i-1} \theta_{ij}$ counts the total number of overlapping SVM pairs. The denominator $\frac{N(N-1)}{2}$ represents the total number of possible pairs of SVMs, ensuring that $\tau(\mathcal{V})$ is normalized to the interval $[0, 1]$.

- $\tau(\mathcal{V}) = 0$ indicates no overlap (no time correlation).
- $\tau(\mathcal{V}) = 1$ indicates complete overlap (all SVMs overlap in time).

Thus, $\tau(\mathcal{V})$ gives a measure of how time-correlated the SVMs are in terms of their execution periods.

## 5. Tabu Search Algorithm

Tabu search (TS) is an iterative meta-heuristic algorithm designed for solving combinatorial optimization problems. The algorithm explores the solution space until either a predefined number of iterations is reached or a specific cost criterion is met. The process begins with an initial solution, which can be generated by another algorithm (e.g., a random solution if no better alternative is available). The initial solution becomes the current solution at the start of the algorithm. At each iteration, TS computes a set of neighboring solutions by applying perturbations to the current solution. The best solution from this neighborhood with the best cost is selected as the new current solution. To avoid the

algorithm revisiting the same solutions repeatedly, a tabu list is maintained. This list stores a limited number of recently visited solutions, which are temporarily excluded from being selected again as they belong to the list. Each new current solution is added to the tabu list and remains there for a given number of iterations. This mechanism helps the algorithm escape local minima during the search process. Although there are no mathematical proofs guaranteeing that TS will converge to a global optimum, it is widely used in practice due to its ability to find solutions that are often close to optimal. Additionally, TS is well suited for handling combinatorial optimization problems of practical scale. For further details on TS, refer to [77].

In order to design a TS algorithm, four problem-specific elements must be defined: an initial solution, a cost function to evaluate the solutions produced by the algorithm, a neighborhood exploration procedure to generate new solutions from the current one, and a tabu list in order to prevent the algorithm from cycling.

**Initial solution:** The algorithm starts with an initial solution where no SVMs are placed on any PMs. This is represented as an empty placement configuration, where each SVM is unassigned.

**Neighborhood exploration procedure:** At each iteration, the algorithm explores a neighborhood of the current solution by generating possible moves. A move consists of assigning an SVM to a PM, subject to the resource constraints. Each potential assignment is evaluated by checking if the CPU, memory, and storage capacities of the PM are sufficient to host the SVM, considering other SVMs already placed on the same PM within overlapping time periods.

**Cost function:** The cost function evaluates the quality of a solution based on the number of successfully placed SVMs. The goal is to maximize the number of SVMs that are placed on PMs while respecting resource constraints.

**Tabu list and aspiration criterion:** To prevent cycling and encourage exploration of the search space, the algorithm maintains a tabu list, which records recently made moves (i.e., assignments of SVMs to PMs). Moves stored in the tabu list are prohibited for a certain number of iterations (referred to as the tabu tenure).

However, an aspiration criterion allows the algorithm to override the tabu status of a move if it leads to a better solution than the current best-known solution. This flexibility ensures that the algorithm does not miss promising solutions.

**Solution update:** At each iteration, the algorithm selects the best neighbor (the best possible assignment of an SVM to a PM) from the neighborhood that is not tabu, or which satisfies the aspiration criterion. The current solution is updated to this best neighbor, and the tabu list is updated accordingly.

The proposed TS algorithm is capable of efficiently navigating the large and complex solution space of the SVMPP. By maintaining a balance between exploration and exploitation, the algorithm avoids becoming trapped in local optima and is able to find high-quality solutions in a reasonable amount of time. The principle steps of the algorithm are as follows:

1. Initialization
    - Start with an initial solution where no SVMs are placed.
    - Initialize the best solution to the initial solution.
    - Initialize an empty tabu list.
2. Neighborhood search
    - For each SVM, generate possible placements on each PM.
    - For each possible placement (neighbor), check resource feasibility (CPU, memory, storage).
3. Cost evaluation
    - Evaluate the cost of each neighbor by counting the number of successfully placed SVMs.

4. Move selection
   - Select the best non-tabu neighbor, or select a tabu move if it improves the best solution (aspiration criterion).
5. Tabu list update
   - Add the current move to the tabu list and manage its size based on the tabu tenure.
6. Update solution
   - Update the current solution to the best selected neighbor.
   - Update the best solution if the current solution improves it.
7. Termination
   - Repeat steps 2–6 for a predefined number of iterations.
8. Output
   - Return the best solution found, which is the configuration that maximizes the number of successfully placed SVMs.

The pseudo-code for the TS algorithm is provided in Algorithm 1. The TS relies on several supporting functions to determine optimal SVM placement. Specifically, CostFunction evaluates each potential solution by counting the number of SVMs successfully assigned to PMs, guiding the search toward maximizing the number of accepted SVMs. The IsTimeOverlap function checks if two SVMs have overlapping active times, ensuring that only time-disjoint SVMs share resources on the same PM, which is crucial for meeting temporal constraints. Lastly, IsValidPlacement determines if a PM has sufficient resources (CPU, memory, and storage) to host a particular SVM, accounting for any overlapping SVMs already assigned to that PM.

**Algorithm 1:** Tabu Search Algorithm for SVM Placement

```
Input: Set of PMs, Set of SVMs, max_iterations, tabu_tenure
Output: Best placement of SVMs on PMs, Best cost
Initialize best_solution with no placements (all SVMs set to None);
current_solution ← best_solution;
best_cost ← CostFunction(best_solution, SVMs);
tabu_list ← empty list;
for iteration ← 1 to max_iterations do
    neighborhood ← empty list;
    for each SVM in SVMs do
        for each PM in PMs do
            if IsValidPlacement(PM, overlapping SVMs on PM, SVM) then
                Create neighbor by moving SVM to PM;
                Add neighbor to neighborhood;
            end
        end
    end
    Select best_neighbor from neighborhood not in tabu_list or satisfying
      aspiration criterion;
    Update best_solution if best_neighbor improves the best cost;
    Update tabu_list with new moves, remove oldest if list exceeds tabu_tenure;
end
return best_solution, best_cost;
```

## 6. Performance Evaluation

In the following, we present the simulation results used to evaluate the performance of the TS algorithm in solving the SVMPP. First, we analyze the performance of the TS algo-

rithm compared to an integer linear programming (ILP) model that considers permanent VM (PVM) requests (i.e., VMs without specific start and end times) to highlight the gap between the optimal solution obtained by the ILP and the sub-optimal solution produced by the TS algorithm. After demonstrating the TS algorithm's performance, we compare it against two meta-heuristic methods from the literature: the Ant Colony Optimization (ACO) approach described in [17] and the Particle Swarm Optimization (PSO) method presented in [52].

### 6.1. Simulation Parameters

For benchmarking purposes, the evaluation tests were carried out in a heterogeneous DC comprising two types of PMs. The detailed configurations of these PMs are presented in Table 5. The DC is assumed to have $M$ PMs in total, with an equal distribution between type 1 and type 2 PMs.

**Table 5.** PM configurations.

| PM Type | CPU (Cores) | RAM (GB) | Storage (GB) |
|---|---|---|---|
| HPE ProLiant Gen 10 | 28 | 3000 | 48,000 |
| HPE ProLiant Gen 11 | 96 | 8000 | 60,000 |

The SVM requests are generated based on four distinct types: small (S), medium (M), large (L), and extra large (XL), with specifications detailed in Table 6. These types—S, M, L, and XL—are crucial for accurately simulating a range of real-world workloads in virtualized environments. Each type corresponds to a different level of resource demand, from lightweight (S) to resource-intensive (XL), allowing the placement algorithm to handle a broad spectrum of scenarios. The start and end times for each SVM are drawn from a uniform random distribution over the interval [0, 1440], where 1440 represents the total number of minutes in a day. Although the SVMs are known in advance, the random generation of their start and end times helps simulate a realistic and diverse range of temporal resource demands. For a given number of SVMs, $N$, to be hosted in the DC, the distribution of SVM types is determined using a uniform random distribution, ensuring an equal representation across the four types.

**Table 6.** SVM types and resource requirements.

| SVM Type | CPU (Cores) | RAM (GB) | Storage (GB) |
|---|---|---|---|
| S | 2 | 4 | 40 |
| M | 8 | 16 | 100 |
| L | 32 | 64 | 500 |
| XL | 64 | 128 | 1000 |

The ILP model is implemented using OPL and solved with CPLEX. The TS, as well as the comparative algorithms, are implemented in python and run on an Intel Core i7 (2.6 GHz) processor with 16 GB of RAM.

### 6.2. Performance Comparison: ILP Model and TS

Figures 2 and 3 provide a comprehensive comparison of the ILP model and the TS meta-heuristic in terms of the percentage of hosted PVMs and their relative performance deviation as the number of arriving PVMs increases, for a DC size of $M = 250$ PMs.

Figure 2 shows that both approaches perform optimally at lower arrival rates (up to around 569 PVMs), hosting nearly 100% of the incoming PVMs. However, beyond 569 PVMs, the performance of TS begins to decline, hosting a smaller percentage of PVMs compared to ILP. This gap becomes more pronounced as the number of PVMs increases, with ILP consistently outperforming TS, especially when the PVM count reaches 1000.

At first glance, this difference arises because TS, being an approximate method, computes sub-optimal solutions that become less efficient as the problem size grows. In contrast, ILP, as an exact optimization method, consistently finds the optimal solution, enabling it to host a higher percentage of PVMs, even under high-load conditions.

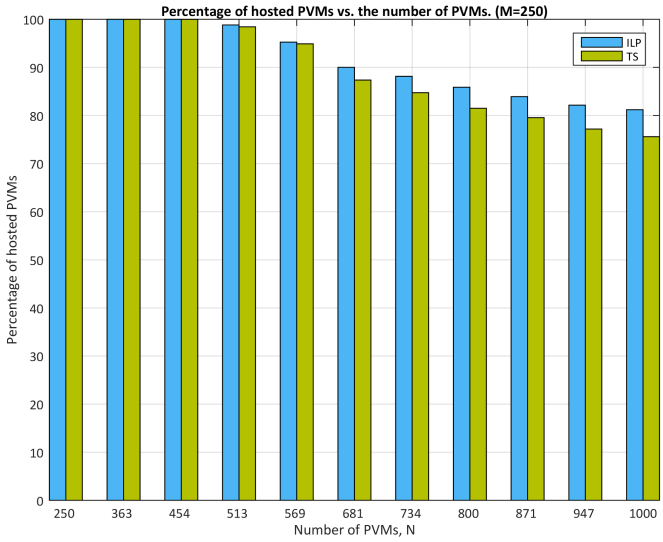



**Figure 2.** Percentage of accepted PVMs vs. the number of arriving PVMs.

To quantify this difference, Figure 3 illustrates the relative deviation between the ILP and TS results. The relative deviation is defined as the percentage difference between the number of accepted PVMs by the ILP and TS, relative to the number of accepted PVMs by the ILP. The figure shows that this deviation remains minimal when the number of arriving PVMs is below 500, indicating similar performance under lighter loads. However, as the PVM count grows, the relative deviation becomes more noticeable, reaching approximately 7% for 1000 arriving PVMs.

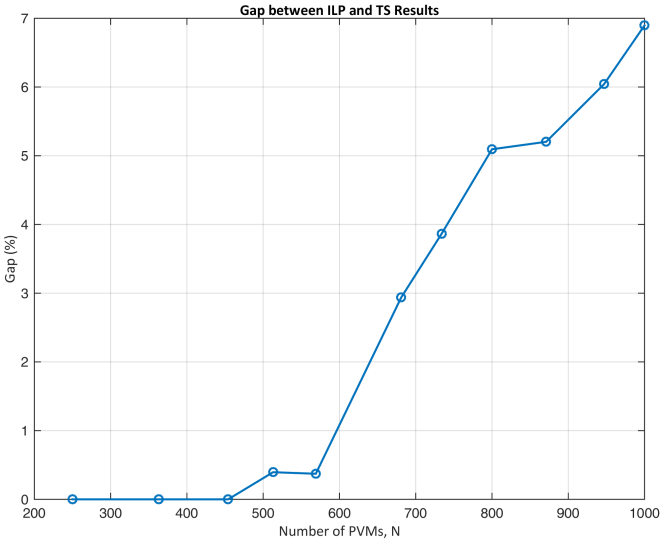



**Figure 3.** ILP model vs. TS: Relative deviation in hosted PVMs.

To further analyze the growing gap between the results computed by the ILP and the TS algorithm, we present the number of hosted PVMs for the different types—S, M, L, and XL—in the DC. Figures 4, 5, 6, and 7, respectively, display the results for each PVM type. In these figures, each group of three bars represents the total number of arriving PVMs to be hosted (first bar from the left), the number of PVMs of type T (T being S, M,

L, or XL) hosted by the ILP (second bar), and the number of PVMs of the same type T accepted by the TS algorithm (third bar).

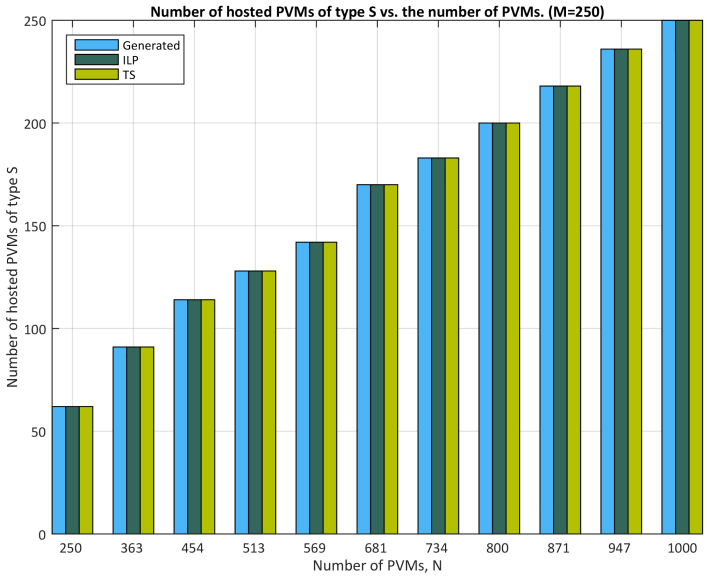



**Figure 4.** Number of hosted PVMs of type S.

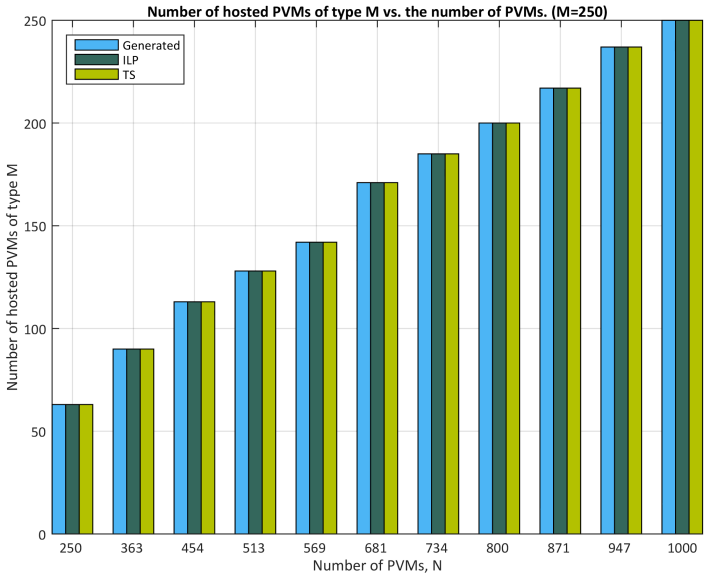



**Figure 5.** Number of hosted PVMs of type M.

The figures indicate that all PVMs of types S and M are successfully hosted by the PMs, regardless of the number of incoming PVMs. However, some PVMs of types L and XL cannot be hosted (rejected), especially under heavy load (the DC capacity is insufficient to accept all PVMs). The difference in the number of PVMs accepted by the ILP model and the TS algorithm can be further elucidated by the results displayed in Figures 6 and 7. The ILP model consistently accepts a larger number of PVMs of type L while rejecting more PVMs of type XL. In contrast, the TS algorithm manages to accept a reasonable number of both L and XL types, with a slight preference for type L. Since PVMs of type XL consume more resources on the PMs than those of type L, this leads to a higher rejection rate in the TS algorithm. Overall, these findings demonstrate that the TS performs comparably to the ILP model.

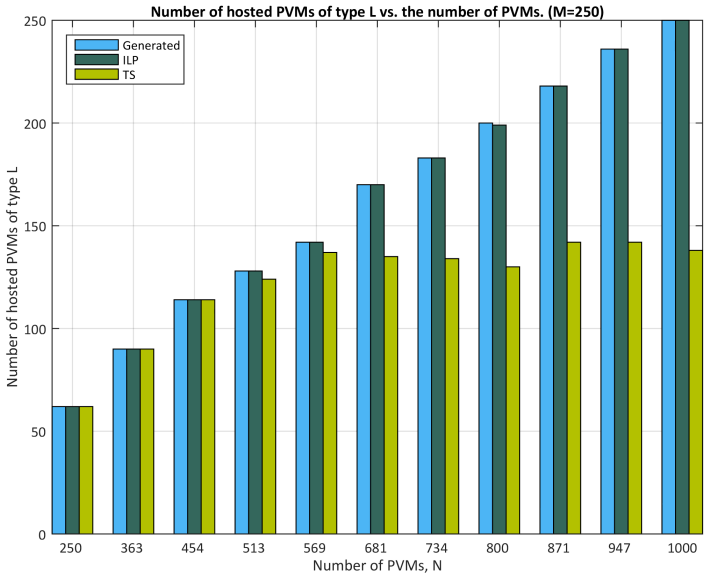



**Figure 6.** Number of hosted PVMs of type L.

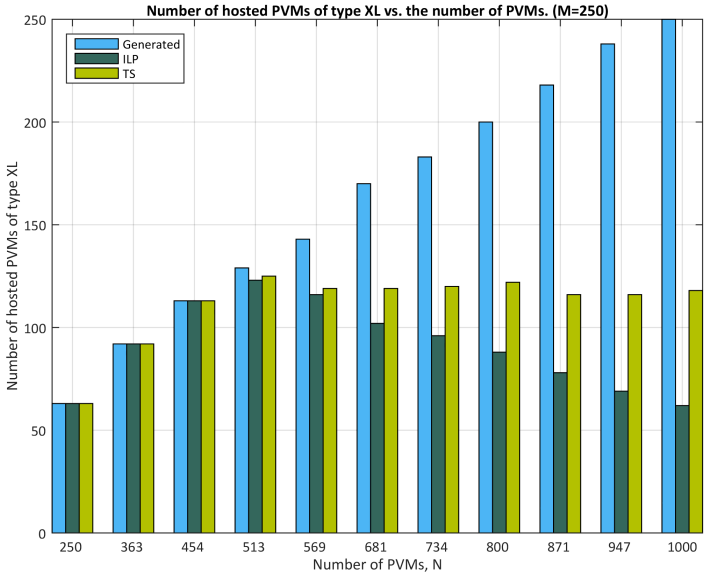



**Figure 7.** Number of hosted PVMs of type XL.

In Figures 8–10, we compare the CPU, RAM, and storage normalized residual capacity, respectively, for each PM, assuming $N$ = 1000 PVMs arriving at the DC for the two optimization approaches. The normalized residual capacity is computed as the ratio between the total available capacity minus the consumed capacity and the total available capacity. The ILP method shows high variance in CPU, RAM, and storage utilization, with some PMs having almost full residual capacity while others are nearly fully utilized. This suggests that ILP results in uneven workload distribution, leaving many machines underutilized and potentially leading to inefficiencies in the DC. In contrast, the TS method results in consistently low residual capacity across most PMs, indicating near-total resource utilization and a more balanced workload allocation. TS appears to maximize resource use, ensuring that almost all machines operate at or near full capacity, minimizing wasted resources. Consequently, TS would be a more efficient approach when the goal is to maximize resource utilization, while ILP may prioritize maximizing the number of hosted PVMs, potentially at the cost of under-utilizing available resources.

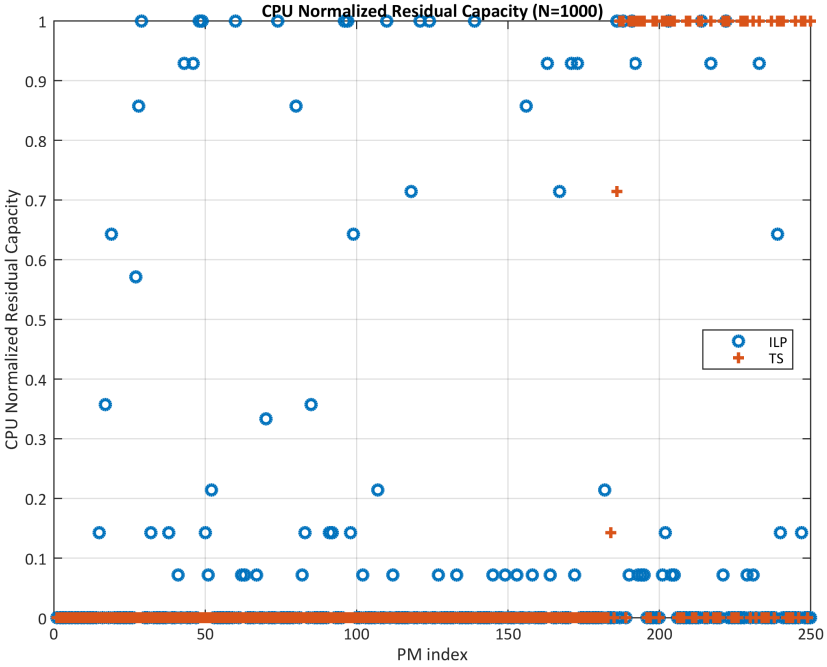



**Figure 8.** CPU normalized residual capacity.

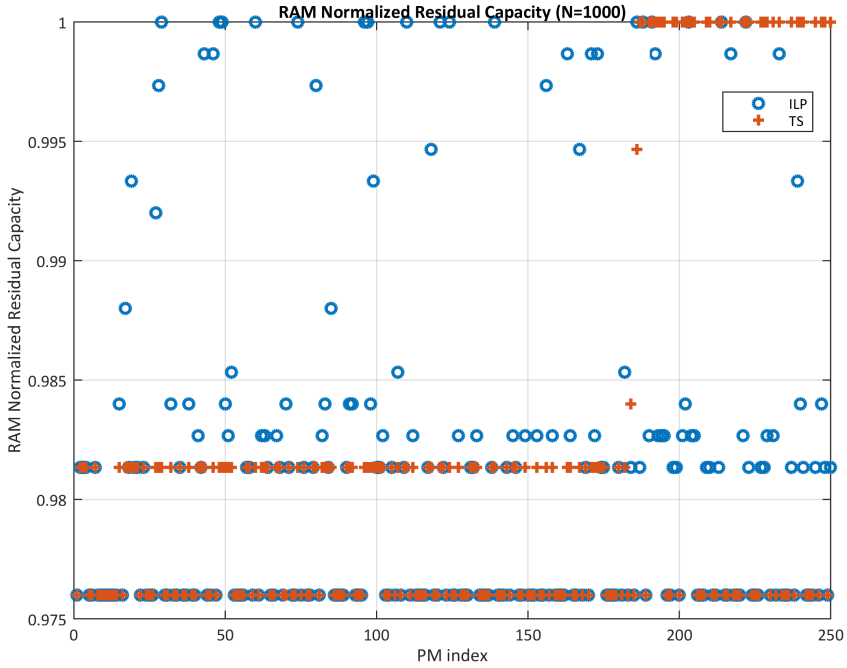



**Figure 9.** RAM normalized residual capacity.

Figure 11 draws the CPU execution time for the two methods. The execution time for the ILP model starts at around $5 \times 10^5$ seconds for $N = 250$ PVMs and increases slightly as the number of PVMs grows, showing that ILP is computationally expensive and does not scale well as the number of PVMs increases. In contrast, TS remains near zero for the entire range, indicating a much faster execution time regardless of the number of PVMs. The graph demonstrates that TS vastly outperforms ILP in terms of CPU execution time when handling increasing numbers of PVMs.

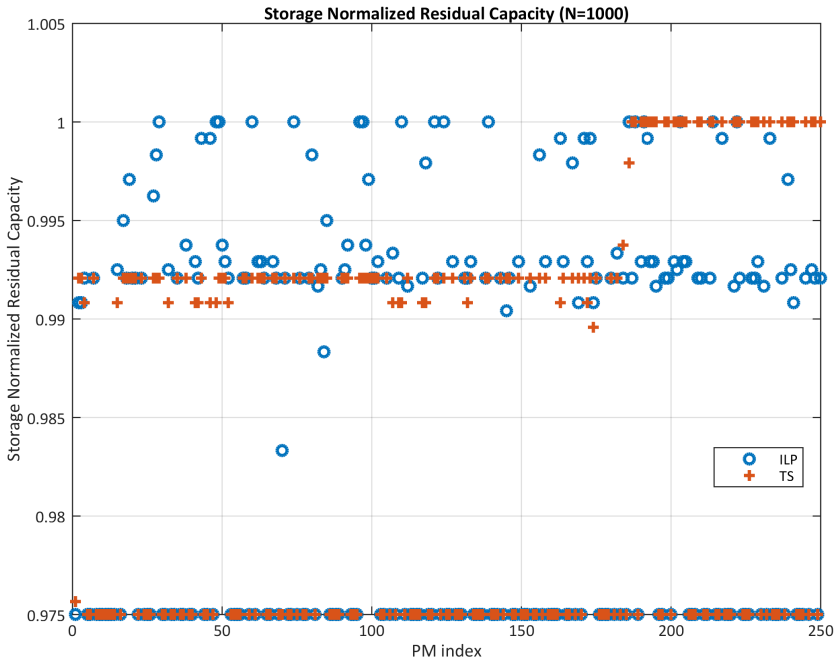



**Figure 10.** Storage normalized residual capacity.

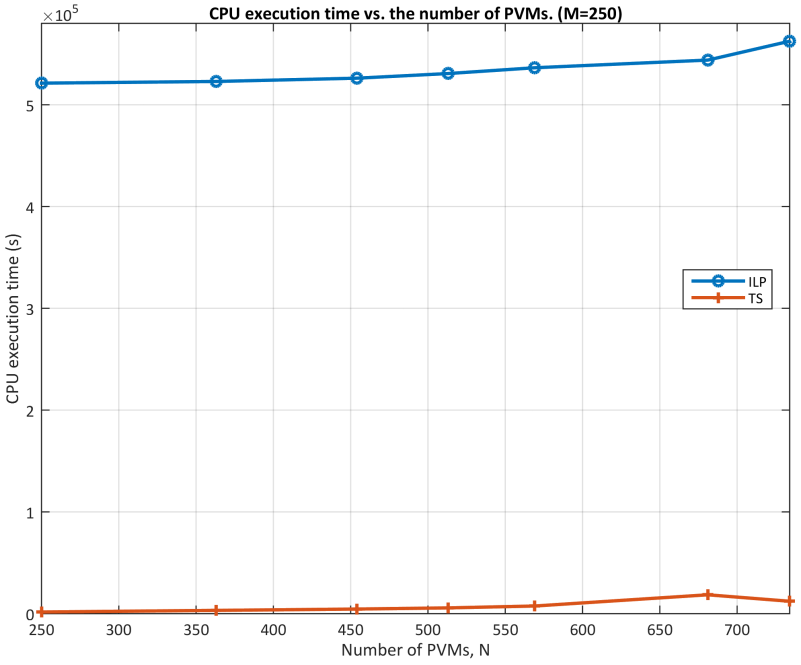



**Figure 11.** CPU execution time.

### *6.3. Performance Comparison: Tabu Search, ACO, and PSO*

In this section, we present a comparative performance analysis of the proposed TS algorithm against two well-established meta-heuristic algorithms from the literature: ACO and PSO. The goal of this comparison is to evaluate the effectiveness of the TS algorithm in solving the SVMPP, particularly in terms of solution quality, convergence speed, and robustness. By benchmarking TS against ACO and PSO, we can highlight its strengths and advantages, as well as identify any potential limitations.

Figure 12 shows the percentage of hosted SVMs computed by TS, ACO, and PSO as the number of arriving SVMs, $N$, increases. The total number of available machines is kept constant at $M = 250$. Key observations include the following:

- The TS algorithm consistently achieves the highest percentage of hosted SVMs, maintaining nearly 100% across all scenarios. ACO performs slightly below TS, while PSO shows the lowest performance, especially as the number of SVMs increases.
- As the number of arriving SVMs grows, PSO experiences a sharper decline in hosted SVMs compared to TS and ACO. For instance, at $N = 250$ arriving SVMs, all three algorithms perform similarly ($\sim$100% hosted SVMs). However, as $N$ increases further, PSO hosts only around 60–70%, while TS and ACO maintain near-full capacity.
- These results suggest that TS and ACO scale more effectively with increasing SVMs than PSO, which struggles to maintain hosting percentages as $N$ rises. TS and ACO are better at exploiting the time disjointness of the SVMs to optimize resource utilization on each PM, likely due to more efficient packing of multiple SVMs on the same PM by considering their time windows. In contrast, PSO may not explore the solution space as effectively, possibly leading to premature convergence or under-utilization of available PM resources over time.

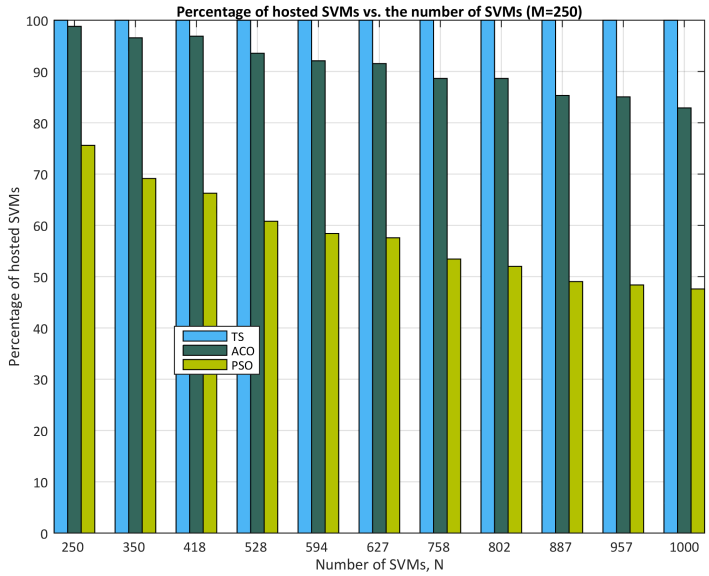



**Figure 12.** Percentage of hosted SVMs vs. the number of arriving SVMs.

The percentage gain of the TS algorithm over ACO and PSO is given in Figure 13. The gain of the TS over ACO remains relatively small but steadily increases as $N$ grows, suggesting that TS outperforms ACO by a modest margin. The gain appears to be most significant for larger $N$ values, reaching approximately 15%. This indicates that while both algorithms handle smaller numbers of SVMs similarly, TS demonstrates a greater advantage as the number of SVMs increases, suggesting better scalability and resource utilization under high load conditions. The strength of TS in high-load scenarios lies in its local search mechanism, which allows it to efficiently explore the neighborhood of the current solution and escape local optima, even when faced with resource constraints. This local search capability is crucial when resources are scarce, as TS is able to adapt to the specific constraints of the problem through its memory structure (the tabu list), which prevents cycling and encourages exploration of promising regions in the solution space. The achieved gain of TS over PSO is significantly higher than the gain over ACO. The TS algorithm consistently outperforms PSO by a substantial margin across all values of $N$. The gain starts at around 25% and increases linearly with $N$, reaching over 45%. This sharp rise in performance differential highlights PSO's struggles to deal with the SVMPP efficiently.

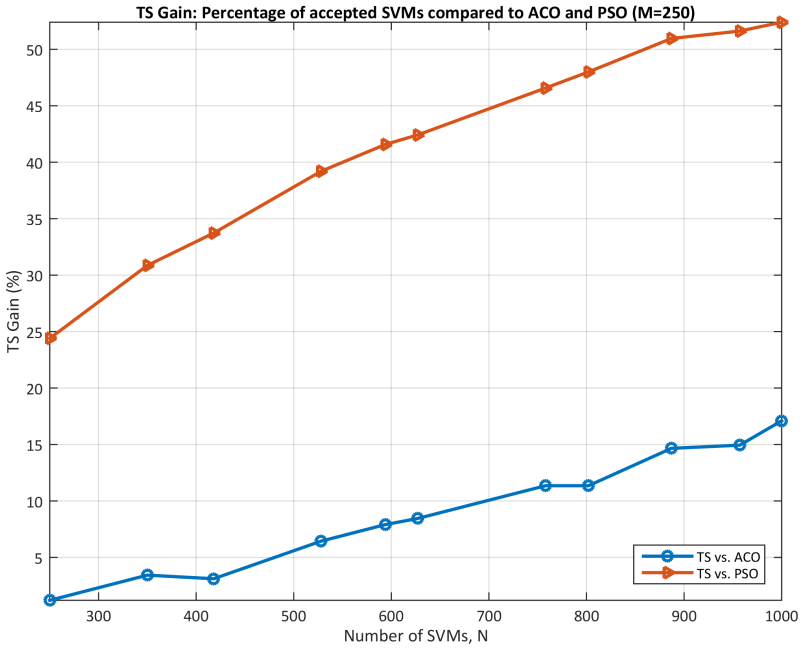



**Figure 13.** TS gain in terms of percentage of accepted SVMs compared to ACO and PSO.

In Figures 14–17, we plot the number of accepted SVMs of types S, M, L, and XL, respectively, with respect to the number of SVM requests to be hosted. Each group of four bars represents the number of SVM requests of a particular type (S, M, L, or XL) to be hosted (first bar from the left), followed by the number of SVMs of that type hosted by the TS algorithm (second bar), the ACO algorithm (third bar), and the PSO algorithm (fourth bar). These figures confirm the previous results, showing that TS consistently hosts a number of SVMs very close to the generated optimal values, outperforming ACO and PSO as $N$ increases. ACO performs slightly below TS but still hosts a high number of SVMs, while PSO consistently hosts the fewest, with the gap widening as $N$ increases, indicating poorer scalability compared to TS and ACO.

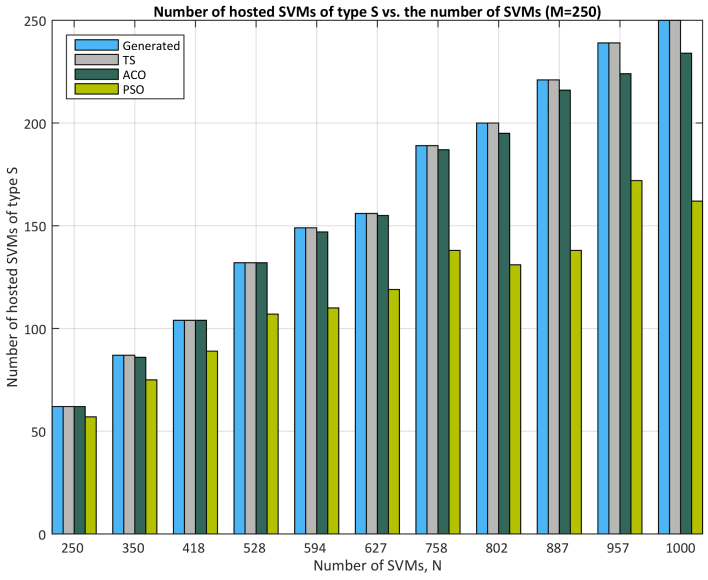



**Figure 14.** Number of hosted SVMs of type S.

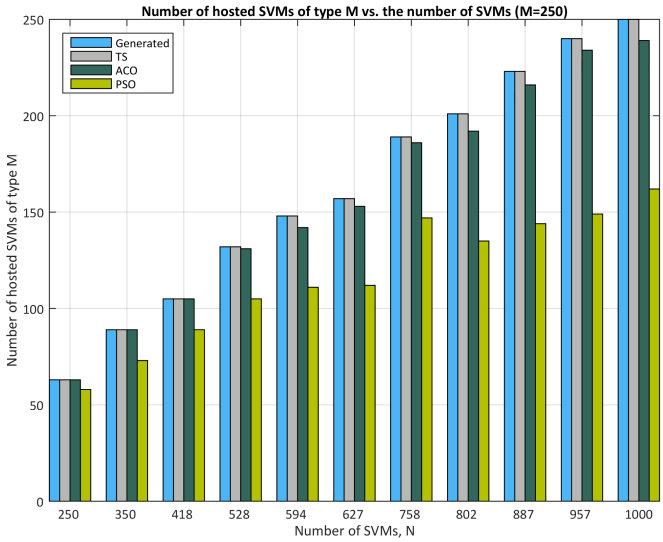



**Figure 15.** Number of hosted SVMs of type M.

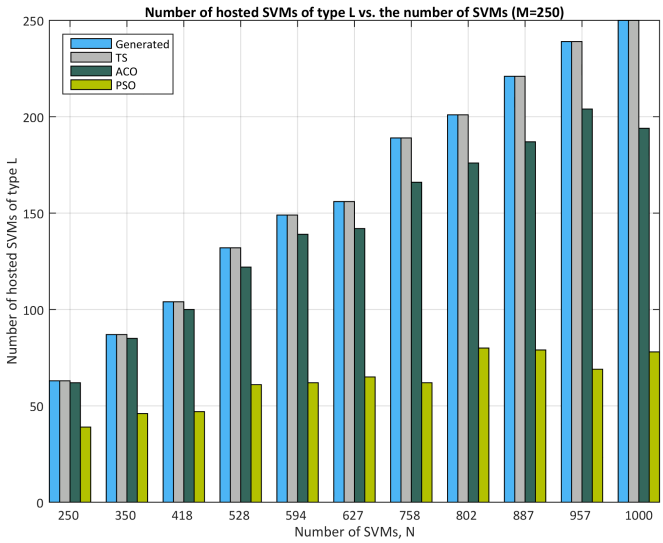



**Figure 16.** Number of hosted SVMs of type L.

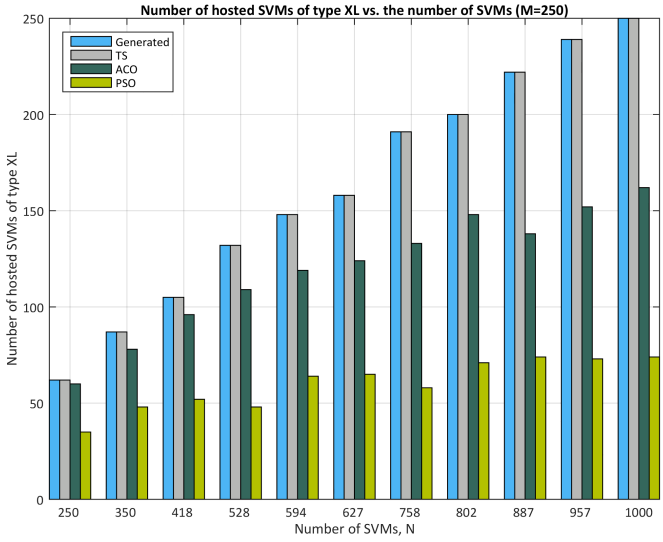



**Figure 17.** Number of hosted SVMs of type XL.

Figure 18, comparing CPU execution time versus the number of arriving SVMs, illustrates a clear trade-off between computational efficiency and solution quality across the three algorithms. The TS algorithm, while offering the best performance in terms of hosted SVMs and scalability, incurs a significantly higher computational cost, with execution time increasing steeply as the number of SVMs grows. In contrast, ACO maintains a more moderate execution time, making it a balanced choice when both performance and efficiency are important. PSO, although the fastest in terms of computation, sacrifices solution quality, particularly under higher loads, making it suitable for time-sensitive scenarios but less ideal for more demanding problems.

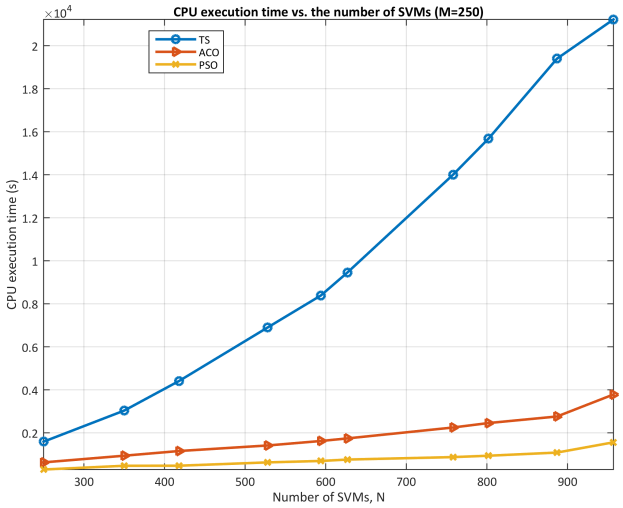



**Figure 18.** CPU execution time.

### *6.4. Impact of the Normalized Time Correlation on the Percentage of Accepted SVMs*

In Figure 19, we draw the relationship between the normalized time correlation and the percentage of accepted SVMs for the three considered algorithms. The key findings are that TS consistently outperforms the others, maintaining nearly 100% acceptance of SVMs across all time correlations, demonstrating its stability and reliability. ACO performs well, with acceptance rates generally between 85% and 100%, but shows some variability as time correlation increases. PSO, however, performs the worst, with acceptance percentages ranging from 50% to 75%, and exhibits significant performance degradation as time correlation rises, particularly beyond 0.49. Overall, TS is the most stable and effective, while PSO struggles most, especially at higher time correlations.

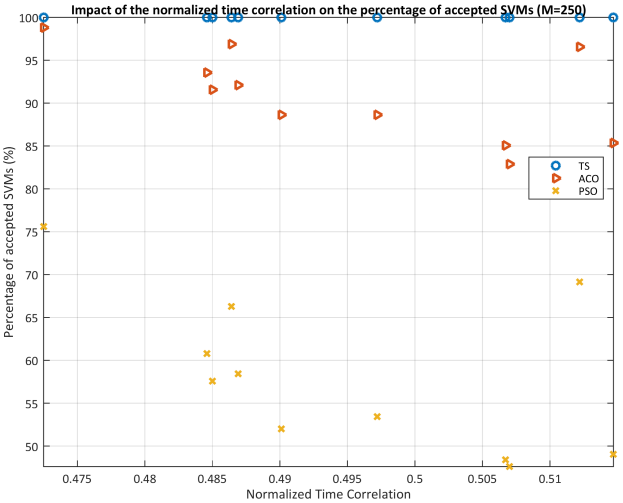



**Figure 19.** Impact of time correlation.

### *6.5. Discussion*

The performance comparison of the proposed TS algorithm with ACO and PSO highlights both its strengths and limitations in solving the SVMPP. TS consistently outperforms ACO and PSO, particularly as the number of SVMs increases, achieving up to 15% increase in resource efficiency over ACO and more than 45% better performance than PSO. TS's success can be attributed to its effective local search mechanism, which avoids local optima and ensures optimal resource utilization under increasing load, as well as its tabu list, which prevents cycling and supports scalability.

However, TS incurs higher computational costs, especially in managing the frequently accessed tabu list, as the number of SVMs grows [78]. It also faces challenges related to parameter tuning and computational complexity. Despite these issues, recent improvements, including hybridization with other meta-heuristics and parallel computing, have enhanced TS's scalability and performance, offering valuable insights for further optimization in VM placement problems [79].

## 7. Conclusions and Future Work

In this study, we introduced the concept of SVMs, a novel VM model that considers both resource needs and specific time windows for each VM. This model allows for optimized resource sharing by enabling multiple SVMs to occupy the same PM when their operational times do not overlap, improving efficiency over traditional static resource models.

We developed a TS algorithm to solve the SVMPP and compared its performance to an ILP model and two meta-heuristics, ACO and PSO. The TS algorithm consistently achieved near-optimal solutions, outperforming ACO and PSO in scalability and the number of hosted SVMs, though with higher computational costs. While ACO was competitive, PSO lagged behind in both solution quality and scalability.

Future work will focus on adaptive, dynamic algorithms for real-time workload variations, potentially leveraging machine learning for predictive management and decision making. Further, we aim to address challenges in hybrid cloud environments and enhance fault tolerance to ensure resilience in VM placement and resource allocation.

**Author Contributions:** Conceptualization, M.K.; methodology, M.K., A.S.K. and F.B.; software, M.K.; validation, M.K., A.S.K. and F.B.; formal analysis, M.K., A.S.K. and F.B.; investigation, M.K.; resources, M.K., A.S.K. and F.B.; data curation, M.K.; writing—original draft preparation, M.K.; writing—review and editing, M.K., A.S.K. and F.B.; visualization, M.K.; supervision, M.K., A.S.K. and F.B.; project administration, M.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** The data supporting the findings of this study are available from the corresponding author upon reasonable request.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Varghese, B.; Buyya, R. Next Generation Cloud Computing: New Trends and Research Directions. *Future Gener. Comput. Syst.* **2018**, *79*, 849–861. [CrossRef]
2. Buyya, R.; Srirama, S.N.; Casale, G.; Calheiros, R.; Simmhan, Y.; Varghese, B.; Gelenbe, E.; Javadi, B.; Vaquero, L.M.; Netto, M.A.S.; et al. A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade. *ACM Comput. Surv.* **2018**, *51*, 105. [CrossRef]
3. Wang, J.; Zhao, G.; Xu, H.; Zhai, Y.; Zhang, Q.; Huang, H.; Yang, Y. A Robust Service Mapping Scheme for Multi-Tenant Clouds. *IEEE/ACM Trans. Netw.* **2022**, *30*, 1146–1161. [CrossRef]
4. Ali Jabber, S.; Hashem, S.H.; Jafer, S.H. Task Scheduling and Resource Allocation in Cloud Computing: A Review and Analysis. In Proceedings of the 2023 3rd International Conference on Emerging Smart Technologies and Applications (eSmarTA), Taiz, Yemen, 10–11 October 2023; pp. 1–8. [CrossRef]
5. Agrawal, S.; Singh, D. Study Containerization Technologies like Docker and Kubernetes and their Role in Modern Cloud Deployments. In Proceedings of the 2024 IEEE 9th International Conference for Convergence in Technology (I2CT), Pune, India, 5–7 April 2024; pp. 1–5. [CrossRef]

6. Cordeau, J.F.; Laporte, G. Tabu Search Heuristics for the Vehicle Routing Problem. In *Les Cahiers du GERAD G-2002-15, Groupe D'études et de Recherche en Analyse des Décisions*; GERAD: Montréal, QC, Canada, 2002.
7. Glover, F.W. Tabu Search—Part I. *Orsa J. Comput.* **1989**, *1*, 190–206. [CrossRef]
8. Glover, F.W. Tabu Search—Part II. *Orsa J. Comput.* **1990**, *2*, 4–32. [CrossRef]
9. Natarajan, P.; Panjatcharam, V.G.; Rajkumaran, T. Comparative Analysis of Techniques for Efficient Resource Utilization in Cloud Environments Through VM Placement Optimization. In Proceedings of the 2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS), Erode, India, 18–20 October 2023; pp. 1015–1018. [CrossRef]
10. Nagadevi; Vidhya; Jeya. Two Dimensional Balanced Resource Utilization Using Vector Heuristics for Multi-Core Aware Virtual Machine Placement Algorithms in a Cloud Environment. In Proceedings of the 2023 2nd International Conference on Edge Computing and Applications (ICECAA), Namakkal, India, 19–21 July 2023; pp. 112–116. [CrossRef]
11. Choudhury, A.; Nath, K.K.; Ghose, M.; Thakran, Y. Memory and CPU utilization-aware Energy-Efficient VM Placement and Consolidation in Cloud Data Centers. In Proceedings of the 2023 IEEE Guwahati Subsection Conference (GCON), Guwahati, India, 23–25 June 2023; pp. 1–6. [CrossRef]
12. Gupta, M.K.; Amgoth, T. Resource-aware algorithm for virtual machine placement in cloud environment. In Proceedings of the 2016 Ninth International Conference on Contemporary Computing (IC3), Noida, India, 11–13 August 2016; pp. 1–6. [CrossRef]
13. Mosa, A.; Sakellariou, R. Dynamic Virtual Machine Placement Considering CPU and Memory Resource Requirements. In Proceedings of the 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), Milan, Italy, 8–13 July 2019; pp. 196–198. [CrossRef]
14. De, U.C.; Satapathy, R.; Patra, S.S. Optimizing Resource Allocation using Proactive Predictive Analytics and ML-Driven Dynamic VM Placement. In Proceedings of the 2023 4th IEEE Global Conference for Advancement in Technology (GCAT), Bangalore, India, 6–8 October 2023; pp. 1–5. [CrossRef]
15. Khoshkholghi, M.A.; Derahman, M.N.; Abdullah, A.; Subramaniam, S.; Othman, M. Energy-Efficient Algorithms for Dynamic Virtual Machine Consolidation in Cloud Data Centers. *IEEE Access* **2017**, *5*, 10709–10722. [CrossRef]
16. Li, L.; Dong, J.; Zuo, D.; Wu, J. SLA-Aware and Energy-Efficient VM Consolidation in Cloud Data Centers Using Robust Linear Regression Prediction Model. *IEEE Access* **2019**, *7*, 9490–9500. [CrossRef]
17. Liu, X.F.; Zhan, Z.H.; Deng, J.D.; Li, Y.; Gu, T.; Zhang, J. An Energy Efficient Ant Colony System for Virtual Machine Placement in Cloud Computing. *IEEE Trans. Evol. Comput.* **2018**, *22*, 113–128. [CrossRef]
18. Vakilinia, S.; Heidarpour, B.; Cheriet, M. Energy Efficient Resource Allocation in Cloud Computing Environments. *IEEE Access* **2016**, *4*, 8544–8557. [CrossRef]
19. Wang, S.; Zhou, A.; Hsu, C.H.; Xiao, X.; Yang, F. Provision of Data-Intensive Services Through Energy- and QoS-Aware Virtual Machine Placement in National Cloud Data Centers. *IEEE Trans. Emerg. Top. Comput.* **2016**, *4*, 290–300. [CrossRef]
20. Li, Z.; Yu, X.; Zhao, L. A Strategy Game System for QoS-Efficient Dynamic Virtual Machine Consolidation in Data Centers. *IEEE Access* **2019**, *7*, 104315–104329. [CrossRef]
21. Hadadi, A.; Shameli-Sendi, A. A Quality of Service Aware VM Placement for User Applications in Cloud Data Center. In Proceedings of the 2022 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI), Dalian, China, 17–19 October 2022; pp. 1–6. [CrossRef]
22. Patel, K.K.; Desai, M.R.; Soni, D.R. Dynamic priority based load balancing technique for VM placement in cloud computing. In Proceedings of the 2017 International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 8–19 July 2017; pp. 78–83. [CrossRef]
23. Chhabra, S.; Singh, A.K. Optimal VM Placement Model for Load Balancing in Cloud Data Centers. In Proceedings of the 2019 7th International Conference on Smart Computing & Communications (ICSCC), Sarawak, Malaysia, 28–30 June 2019; pp. 1–5. [CrossRef]
24. Zhao, H.; Wang, Q.; Wang, J.; Wan, B.; Li, S. VM Performance Maximization and PM Load Balancing Virtual Machine Placement in Cloud. In Proceedings of the 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, VIC, Australia, 11–14 May 2020; pp. 857–864. [CrossRef]
25. Liu, X.; Mashayekhy, L. Joint Load-Balancing and Energy-Aware Virtual Machine Placement for Network-on-Chip Systems. In Proceedings of the 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), Zurich, Switzerland, 17–20 December 2018; pp. 124–132. [CrossRef]
26. Duong-Ba, T.; Tran, T.; Nguyen, T.; Bose, B. A Dynamic Virtual Machine Placement and Migration Scheme for Data Centers. *IEEE Trans. Serv. Comput.* **2021**, *14*, 329–341. [CrossRef]
27. Li, K.W.; Huang, P.H.; Wen, C.H.P. Reducing network cost of minimal-migration based VM management in cloud datacenters. In Proceedings of the 2016 7th International Conference on the Network of the Future (NOF), Buzios, Brazil, 16–18 November 2016; pp. 1–3. [CrossRef]
28. Duong-Ba, T.; Nguyen, T.; Bose, B.; Tran, T. Joint virtual machine placement and migration scheme for datacenters. In Proceedings of the 2014 IEEE Global Communications Conference, Austin, TX, USA, 8–12 December 2014; pp. 2320–2325. [CrossRef]
29. Zhou, A.; Wang, S.; Cheng, B.; Zheng, Z.; Yang, F.; Chang, R.N.; Lyu, M.R.; Buyya, R. Cloud Service Reliability Enhancement via Virtual Machine Placement Optimization. *IEEE Trans. Serv. Comput.* **2017**, *10*, 902–913. [CrossRef]
30. Zhou, A.; Wang, S.; Hsu, C.H.; Kim, M.H.; Wong, K.s. Virtual machine placement with (m, n)-fault tolerance in cloud data center. *Clust. Comput.* **2019**, *22*, 11619–11631. [CrossRef]

31. Gonzalez, C.; Tang, B. FT-VMP: Fault-Tolerant Virtual Machine Placement in Cloud Data Centers. In Proceedings of the 2020 29th International Conference on Computer Communications and Networks (ICCCN), Honolulu, HI, USA, 3–6 August 2020; pp. 1–9. [CrossRef]
32. Regaieg, R.; Koubaa, M.; Osei-Opoku, E.; Aguili, T. Multi-Objective Mixed Integer Linear Programming Model for VM Placement to Minimize Resource Wastage in a Heterogeneous Cloud Provider Data Center. In Proceedings of the 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN), Prague, Czech Republic, 3–6 July 2018; pp. 401–406. [CrossRef]
33. Wei, C.; Hu, Z.H.; Wang, Y.G. Exact algorithms for energy-efficient virtual machine placement in data centers. *Future Gener. Comput. Syst.* **2020**, *106*, 77–91. [CrossRef]
34. Patra, S.S.; Bhusan Dash, B.; Barik, L.; Jena, J.R.; Nanda, S.; Barik, R.K. Improving VM Placement in fog Center by Multi-objective optimization. In Proceedings of the 2022 IEEE 2nd International Symposium on Sustainable Energy, Signal Processing and Cyber Security (iSSSC), Gunupur, India, 15–17 December 2022; pp. 1–6. [CrossRef]
35. Zhang, J.; Wang, W.; Xiao, Y.; Hu, L.; Li, Y.; Zhao, Y.; Zhang, J. Carbon-efficient Virtual Machine Placement in Cloud Datacenters over Optical Networks. In Proceedings of the 2023 Opto-Electronics and Communications Conference (OECC), Shanghai, China, 2–6 July 2023; pp. 1–4. [CrossRef]
36. Shirvani, M.H.; Seifhosseini, S. Power Management of Cloud Datacenter in Infrastructure Level via Efficinet Virtual Machine Placement by Utilizing Hybrid Genetic Algorithm. In Proceedings of the 2023 International Symposium on Signals, Circuits and Systems (ISSCS), Iasi, Romania, 13–14 July 2023; pp. 1–4. [CrossRef]
37. Zhao, G.; Liu, J.; Zhai, Y.; Xu, H.; He, H. Alleviating the Impact of Abnormal Events Through Multi-Constrained VM Placement. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 1508–1523. [CrossRef]
38. Wang, W.; Tornatore, M.; Zhao, Y.; Chen, H.; Li, Y.; Gupta, A.; Zhang, J.; Mukherjee, B. Infrastructure-efficient Virtual-Machine Placement and Workload Assignment in Cooperative Edge-Cloud Computing Over Backhaul Networks. *IEEE Trans. Cloud Comput.* **2023**, *11*, 653–665. [CrossRef]
39. Benbrahim, S.E.; Quintero, A.; Bellaïche, M. Live Placement of Interdependent Virtual Machines to Optimize Cloud Service Profits and Penalties on SLAs. *IEEE Trans. Cloud Comput.* **2019**, *7*, 237–249. [CrossRef]
40. Azizi, S.; Shojafar, M.; Abawajy, J.; Buyya, R. GRVMP: A Greedy Randomized Algorithm for Virtual Machine Placement in Cloud Data Centers. *IEEE Syst. J.* **2021**, *15*, 2571–2582. [CrossRef]
41. Zhou, Z.; Shojafar, M.; Alazab, M.; Abawajy, J.; Li, F. AFED-EF: An Energy-Efficient VM Allocation Algorithm for IoT Applications in a Cloud Data Center. *IEEE Trans. Green Commun. Netw.* **2021**, *5*, 658–669. [CrossRef]
42. Alsaidy, S.A.; Al-Chalabi, A.M.; Alnooh, A.H.; Sahib, M.A. Multi-resource Power Efficient Virtual Machine Placement in Cloud Computing. In Proceedings of the 2021 International Conference on Computing and Communications Applications and Technologies (I3CAT), Ipswich, UK, 15 September 2021; pp. 93–97. [CrossRef]
43. Zhang, R.; Chen, Y.; Dong, B.; Tian, F.; Zheng, Q. A Genetic Algorithm-Based Energy-Efficient Container Placement Strategy in CaaS. *IEEE Access* **2019**, *7*, 121360–121373. [CrossRef]
44. Naeen, H.M. Virtual machine consolidation using SLA-aware genetic algorithm placement for data centers with non-stationary workloads. In Proceedings of the 2021 11th International Conference on Computer Engineering and Knowledge (ICCKE), Mashhad, Iran, 28–29 October 2021; pp. 150–156. [CrossRef]
45. Wu, X. A GA-Based Energy Aware Virtual Machine Placement Algorithm for Cloud Data Centers. In Proceedings of the 2021 12th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), Xi'an, China, 10–12 December 2021; pp. 42–46. [CrossRef]
46. Seyyedsalehi, S.M.; Khansari, M. Virtual Machine Placement Optimization for Big Data Applications in Cloud Computing. *IEEE Access* **2022**, *10*, 96112–96127. [CrossRef]
47. Sajadinia, A.; Yari, A. Virtual Machine Placement Strategy Using Clustering and Genetic Algorithm for increasing cloud performance and power saving. In Proceedings of the 2023 28th International Computer Conference, Computer Society of Iran (CSICC), Tehran, Iran, 25–26 January 2023; pp. 1–5. [CrossRef]
48. Wu, Y.; Tang, M.; Fraser, W. A simulated annealing algorithm for energy efficient virtual machine placement. In Proceedings of the 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Seoul, Republic of Korea, 14–17 October 2012; pp. 1245–1250. [CrossRef]
49. Gilesh, M.; Satheesh, S.; Chandran, A.; Madhu Kumar, S.; Jacob, L. Parallel Schedule of Live Migrations for Virtual Machine Placements. In Proceedings of the 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), Philadelphia, PA, USA, 18–20 October 2018; pp. 64–70. [CrossRef]
50. Zhao, D.M.; Zhou, J.T.; Li, K. An Energy-Aware Algorithm for Virtual Machine Placement in Cloud Computing. *IEEE Access* **2019**, *7*, 55659–55668. [CrossRef]
51. Dubey, K.; Sharma, S.C.; Nasr, A.A. A Simulated Annealing based Energy-Efficient VM Placement Policy in Cloud Computing. In Proceedings of the 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, India, 24–25 February 2020; pp. 1–5. [CrossRef]
52. Wang, S.; Liu, Z.; Zheng, Z.; Sun, Q.; Yang, F. Particle Swarm Optimization for Energy-Aware Virtual Machine Placement Optimization in Virtualized Data Centers. In Proceedings of the 2013 International Conference on Parallel and Distributed Systems, Seoul, Republic of Korea, 15–18 December 2013; pp. 102–109. [CrossRef]

53. Kumar, D.; Raza, Z. A PSO Based VM Resource Scheduling Model for Cloud Computing. In Proceedings of the 2015 IEEE International Conference on Computational Intelligence & Communication Technology, Ghaziabad, India, 13–14 February 2015; pp. 213–219. [CrossRef]
54. Abdessamia, F.; Tai, Y.; Zhang, W.Z.; Shafiq, M. An Improved Particle Swarm Optimization for Energy-Efficiency Virtual Machine Placement. In Proceedings of the 2017 International Conference on Cloud Computing Research and Innovation (ICCCRI), Singapore, 1–12 April 2017; pp. 7–13. [CrossRef]
55. BRAIKI, K.; YOUSSEF, H. Multi-Objective Virtual Machine Placement Algorithm Based on Particle Swarm Optimization. In Proceedings of the 2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC), Limassol, Cyprus, 25–29 June 2018; pp. 279–284. [CrossRef]
56. Ibrahim, A.; Noshy, M.; Ali, H.A.; Badawy, M. PAPSO: A Power-Aware VM Placement Technique Based on Particle Swarm Optimization. *IEEE Access* **2020**, *8*, 81747–81764. [CrossRef]
57. Shao, Y.; Yang, Q.; Gu, Y.; Pan, Y.; Zhou, Y.; Zhou, Z. A Dynamic Virtual Machine Resource Consolidation Strategy Based on a Gray Model and Improved Discrete Particle Swarm Optimization. *IEEE Access* **2020**, *8*, 228639–228654. [CrossRef]
58. Farahnakian, F.; Ashraf, A.; Liljeberg, P.; Pahikkala, T.; Plosila, J.; Porres, I.; Tenhunen, H. Energy-Aware Dynamic VM Consolidation in Cloud Data Centers Using Ant Colony System. In Proceedings of the 2014 IEEE 7th International Conference on Cloud Computing, Anchorage, AK, USA, 27 Jun–2 July 2014; pp. 104–111. [CrossRef]
59. Gao, C.; Wang, H.; Zhai, L.; Gao, Y.; Yi, S. An Energy-Aware Ant Colony Algorithm for Network-Aware Virtual Machine Placement in Cloud Computing. In Proceedings of the 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), Wuhan, China, 13–16 December 2016; pp. 669–676. [CrossRef]
60. Qin, Y.; Wang, H.; Zhu, F.; Zhai, L. A Multi-Objective Ant Colony System Algorithm for Virtual Machine Placement in Traffic Intense Data Centers. *IEEE Access* **2018**, *6*, 58912–58923. [CrossRef]
61. Wei, W.; Gu, H.; Lu, W.; Zhou, T.; Liu, X. Energy Efficient Virtual Machine Placement With an Improved Ant Colony Optimization Over Data Center Networks. *IEEE Access* **2019**, *7*, 60617–60625. [CrossRef]
62. Suliman, S.I.; Abdul Mutalib, A.R.I.; Mohamad Yusof, Y.W.; Yasmin Abdul Rahman, F.; Shahbudin, S. Energy-Efficient Virtual Machine Placement in Data Centers by Ant Colony Optimization Algorithm (ACO). In Proceedings of the 2024 IEEE 6th Symposium on Computers & Informatics (ISCI), Kuala Lumpur, Malaysia, 10 August 2024; pp. 146–151. [CrossRef]
63. Masoumzadeh, S.S.; Hlavacs, H. A Cooperative Multi Agent Learning Approach to Manage Physical Host Nodes for Dynamic Consolidation of Virtual Machines. In Proceedings of the 2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA), Munich, Germany, 11–12 June 2015; pp. 43–50. [CrossRef]
64. Jumnal, A.; Dilip Kumar, S.M. Optimal VM Placement Approach Using Fuzzy Reinforcement Learning for Cloud Data Centers. In Proceedings of the 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), Tirunelveli, India, 4–6 February 2021; pp. 29–35. [CrossRef]
65. Kim, S.; Pham, N.; Baek, W.; Choi, Y.r. Holistic VM Placement for Distributed Parallel Applications in Heterogeneous Clusters. *IEEE Trans. Serv. Comput.* **2021**, *14*, 1411–1425. [CrossRef]
66. Long, S.; Li, Z.; Xing, Y.; Tian, S.; Li, D.; Yu, R. A Reinforcement Learning-Based Virtual Machine Placement Strategy in Cloud Data Centers. In Proceedings of the 2020 IEEE 22nd International Conference on High Performance Computing and Communications, Yanuca Island, Fiji, 14–16 December 2020; pp. 223–230. [CrossRef]
67. Kamat, S.; Naik, S.; Kanamadi, S.; Alur, S.; G, N.D.; Patil, S. Compute and Network Aware VM Scheduling using Reinforcement Learning in Cloud. In Proceedings of the 2023 IEEE 8th International Conference for Convergence in Technology (I2CT), Lonavla, India, 7–9 April 2023; pp. 1–7. [CrossRef]
68. Xiao, Z.; Jiang, J.; Zhu, Y.; Ming, Z.; Zhong, S.; Cai, S. A Solution of Dynamic VMs Placement Problem for Energy Consumption Optimization Based on Evolutionary Game Theory. *J. Syst. Softw.* **2015**, *101*, 260–272. [CrossRef]
69. Wang, W.; Jiang, Y.; Wu, W. Multiagent-Based Resource Allocation for Energy Minimization in Cloud Computing Systems. *IEEE Trans. Syst. Man Cybern. Syst.* **2017**, *47*, 205–220. [CrossRef]
70. Homsi, S.; Quan, G.; Wen, W.; Chapparo-Baquero, G.A.; Njilla, L. Game Theoretic-Based Approaches for Cybersecurity-Aware Virtual Machine Placement in Public Cloud Clusters. In Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Larnaca, Cyprus, 14–17 May 2019; pp. 272–281. [CrossRef]
71. Banerjee, S.; Roy, S.; Khatua, S. Game Theory Based Energy-aware Virtual Machine Placement towards Improving Resource Efficiency in Homogeneous Cloud Data Center. In Proceedings of the 2022 IEEE Calcutta Conference (CALCON), Kolkata, India, 10–11 December 2022; pp. 293–298. [CrossRef]
72. Suseela, J.; Venugopal, J. A Multi-Objective Hybrid ACO-PSO Optimization Algorithm for Virtual Machine Placement in Cloud Computing. *Int. J. Res. Eng. Technol.* **2014**, 474–476.
73. Hong, L.; Yufei, G. GACA-VMP: Virtual Machine Placement Scheduling in Cloud Computing Based on Genetic Ant Colony Algorithm Approach. In Proceedings of the 2015 IEEE 12th International Conference on Ubiquitous Intelligence and Computing and 2015 IEEE 12th International Conference on Autonomic and Trusted Computing and 2015 IEEE 15th International Conference on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), Beijing, China, 10–14 August 2015; pp. 1008–1015. [CrossRef]

74. Bharathi, P.D.; Prakash, P.; Kiran, M.V.K. Energy efficient strategy for task allocation and VM placement in cloud environment. In Proceedings of the 2017 Innovations in Power and Advanced Computing Technologies (i-PACT), Vellore, India, 21–22 April 2017; pp. 1–6. [CrossRef]
75. Wei, W.; Wang, K.; Wang, K.; Guo, S.; Gu, H. A Virtual Machine Placement Algorithm Combining NSGA-II and Bin-Packing Heuristic. In Proceedings of the 2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), Gold Coast, QLD, Australia, 5–7 December 2019; pp. 190–195. [CrossRef]
76. Singh, A.K.; Swain, S.R.; Saxena, D.; Lee, C.N. A Bio-Inspired Virtual Machine Placement Toward Sustainable Cloud Resource Management. *IEEE Syst. J.* **2023**, *17*, 3894–3905. [CrossRef]
77. Glover, F.; Laguna, M. *Tabu Search*; Kluwer Academic Publishers: Boston, MA, USA, 1997.
78. Khalid, A.N.; Rao Baki, N.M.; Phani Varma, T.S.; Khan, A.; Singh, N. Tabu Search Algorithm: Optimizing the Search Runtime. In Proceedings of the 2024 5th International Conference for Emerging Technology (INCET), Belgaum, India, 24–26 May 2024; pp. 1–4. [CrossRef]
79. Dao, T.K.; Nguyen, T.T.; Nguyen, T.D.; Nguyen, T.H.; Nguyen, T.T. A Review of the Tabu Search Algorithm for Industrial Applications. *J. Inf. Hiding Multimed. Signal Process.* **2024**, *15*, 179–191.