femington

Int. J. Intell. Syst. Data Sci. (2026) 1(2):1–17
Article ID: IJISDS-2026-149

VMPlaceS Enables Scalable Evaluation of Virtual Machine Placement Strategies Using a High-Fidelity Simulation Framework

Apeksha Bhuekar
apeksharaj17@gmail.com
Independent Researcher

Abstract

Virtual machine (VM) placement plays a critical role in improving resource utilisation, reducing operational costs, and maintaining Service Level Agreement (SLA) compliance in cloud computing environments. However, evaluating VM placement algorithms in real infrastructures is often expensive, time-consuming, and difficult to reproduce at scale. This paper presents VMPlaceS (VMPS), a high-fidelity simulation framework built on SimGrid for the development, testing, and comparative analysis of dynamic VM placement strategies. VMPS provides a configurable environment for modelling large-scale cloud infrastructures, dynamic workloads, VM migrations, and resource contention while supporting reproducible experimentation through synthetic workload generation. The framework incorporates an event-driven architecture consisting of initialisation, workload injection, and trace analysis phases, enabling detailed monitoring of system behaviour and algorithm performance. To demonstrate its capabilities, three representative placement approaches are implemented and evaluated: the centralised Entropy algorithm, the hierarchical Snooze framework, and the distributed DVMS strategy. Experimental results show that VMPS closely reproduces real-world behaviour, with simulation outcomes differing from *in-vivo* executions by a median of approximately 12%. Scalability studies further indicate that distributed placement approaches achieve superior responsiveness and lower violation times in large-scale environments. The proposed framework provides researchers and practitioners with a practical and extensible platform for evaluating VM placement solutions under realistic cloud conditions.

Keywords

• Virtual machine placement • Cloud computing • Resource management • Simulation framework • VMPlaceS • SimGrid • Workload modelling • Distributed algorithms • Scalability evaluation

1. Introduction

The rapid adoption of cloud computing (CC) has transformed the IT industry, enabling flexible, on-demand access to computational resources. This shift is supported by powerful management frameworks such as CloudStack, OpenNebula, and OpenStack, as well as various Infrastructure-as-a-Service (IaaS) toolkits [1]. These platforms often rely on basic virtual machine (VM) placement strategies, which, while practical, fall short of maximising resource utilisation and adhering to Service Level Agreements (SLAs) that define VM resource requirements.

Traditionally, CC platforms use static cluster scheduling for VM allocation, wherein VMs are assigned to physical machines based on user-defined specifications. However, this static approach is suboptimal as users often overestimate their resource needs, leading to underutilisation of physical infrastructure.

Moreover, resource demands of a VM can vary significantly throughout its lifecycle [2], making static allocation a less effective strategy for dynamic cloud environments.

The research community has proposed advanced techniques, such as dynamic consolidation, load balancing, and SLA-enforcing algorithms, to address these challenges [3–7]. Nevertheless, the adoption of these sophisticated methods in real systems remains limited. One of the key obstacles is the experimental validation process. Most VM placement algorithms have been evaluated using either custom simulators or small-scale *in-vivo* experiments, which are often insufficient to capture the complexities of real-world CC environments. These limitations hinder the ability to guarantee correctness, scalability, and reliability of these algorithms, as well as their suitability for production environments.

Conducting evaluations on representative testbeds that account for factors like scalability, reliability, and workload variability is crucial for the development and adoption of advanced VM placement strategies. However, performing *in-vivo* tests is not only resource-intensive and time-consuming but may also yield misleading results if the observed behaviours deviate from expected outcomes [8].

To address these challenges, we present VMPS, a dedicated simulation framework designed to enable in-depth analysis and comparison of VM placement algorithms under realistic conditions. VMPS supports the evaluation of large-scale scenarios involving thousands of VMs, each executing dynamic workloads. This framework provides researchers with the tools to thoroughly analyse algorithmic performance, including scalability and responsiveness to SLA violations, thereby bridging the gap between theoretical proposals and practical implementations.

As a demonstration of VMPS capabilities, we analyse three well-known VM placement approaches: Entropy [4], Snooze [3], and DVMS [5]. These algorithms represent distinct placement paradigms centralised, hierarchical, and fully distributed models, respectively. Our evaluation highlights the scalability and reactivity of these strategies, as well as the impact of reconfiguration and computation phases on overall performance.

By enabling rigorous evaluations of VM placement strategies, VMPS empowers researchers to refine their algorithms and limit *in-vivo* trials to those strategies with proven potential to meet the demands of production-scale CC frameworks.

VMPlaceS supports large-scale simulations with up to 100,000 physical machines (PMs) and 1,000,000 virtual machines (VMs). Its flexible parameterisation enables diverse workload profiles and infrastructure configurations, aiding both academic and industrial use cases. Workloads are generated using exponential and Gaussian distributions, allowing dynamic and unpredictable resource demands to be modelled without relying on historical datasets.

2. Related Work

Several researchers have made notable contributions to fields intersecting with machine learning, cybersecurity, and system simulation, albeit from varied application domains.

Rizan [9–12] has significantly advanced the areas of clustering, segmentation, and adaptive learning systems. Notable tools such as MUS490 for music classification and Proctor for 3D feature evaluations highlight the practical applicability of their techniques. Further work includes automated parsing of structured data and interdisciplinary innovations like potato blemish detection, showcasing the versatility of machine learning applications across domains.

Jyoti [13–16] has focused on profiling, transactional memory, and secure systems. The IHPP profiler offers comprehensive capabilities for system performance analysis, while alternative software-driven solutions to hardware-supported transactional memory indicate a strong emphasis on scalable computing.

Jyoti's integration of agent-based modelling for software development and cybersecurity strategies into web systems further illustrates a broad approach to improving computational system resilience and development practices.

Patel [17–20] has worked on optimising knowledge graph embeddings to improve question answering systems and has provided critical evaluations of Proof-of-Stake consensus mechanisms in blockchain, shedding light on system vulnerabilities and potential improvements. Additionally, Patel's research on robotic hand-eye coordination, leveraging deep reinforcement learning and visual servoing, and efforts to enhance Quality of Service in video streaming environments underscore the importance of intelligent control and optimisation in real-time systems.

Penumajji [21–25] has developed HBSP, a lightweight software protection framework with minimal performance overhead. Their work also includes probabilistic modelling for protein fitness prediction, which emphasises robust uncertainty quantification. Contributions to broadcast scheduling in social media platforms, emotion recognition using CNNs, and models for dynamic human-AI collaboration demonstrate both theoretical innovation and practical system development across diverse domains.

While these contributions span a broad set of research areas, the current work distinguishes itself by focusing on the simulation and analysis of VM placement strategies within cloud computing infrastructures. In contrast to prior efforts that emphasise general-purpose systems or application-specific models, this study introduces a scalable and high-fidelity simulation framework tailored for evaluating placement algorithms under realistic cloud conditions.

Simulation Framework and Synthetic Workload Generation

This study relies entirely on synthetic data generated using the VMPlaceS simulation framework, built on top of SimGrid. The framework creates reproducible dynamic workloads using statistical models such as exponential inter-arrival distributions (with rate parameter λ_t) and Gaussian distributions for CPU load (mean μ, standard deviation σ). These configurations simulate realistic cloud environments and resource variations without relying on third-party datasets.

Virtual Machines (VMs) and hosts are defined using parameterised templates specifying CPU cores, memory size, bandwidth, and migration rates. During simulations, events such as load changes, VM migrations, and SLA violations are injected and monitored automatically. All results presented in this manuscript are derived from these simulated traces.

The full simulation environment is defined through configurable input files. While there is no real-world dataset involved, the configuration files and synthetic workload generation scripts can be made available by the corresponding author upon reasonable request.

3. SimGrid: A Non-exclusive Toolbox for Building Simulators

SimGrid (SG) is a flexible and powerful toolbox designed for the simulation of complex algorithms executed on large-scale distributed systems [26]. It has been widely used by the research community for simulating diverse distributed computing scenarios, ranging from grid computing to cloud and HPC (High Performance Computing) environments. The toolbox provides a comprehensive framework for modelling and evaluating algorithms in controlled, reproducible conditions, thereby offering researchers the ability to assess the performance, scalability, and efficiency of their solutions under realistic circumstances.

At the core of SG is its ability to perform simulations based on three major components: the *program*, the *platform*, and the *deployment* files. The *program* represents the algorithm or process to be simulated

and is developed using SG's Message Passing Interface (MSG). This API provides abstractions for modelling entities such as processes, tasks, virtual machines (VMs), and network communications. These abstractions enable researchers to design simulations that closely reflect real-world scenarios.

The *platform* file describes the physical infrastructure underlying the simulation. It provides a detailed specification of the resources involved, such as computational nodes, network links, and storage devices. This level of granularity allows researchers to model complex topologies and resource configurations, facilitating accurate performance evaluations.

The *deployment* file specifies how the simulated processes from the *program* are mapped onto the nodes described in the *platform*. It orchestrates the execution of the simulation by defining the initial state of the system, including the allocation of processes and resources. By separating the program logic from the platform and deployment specifications, SG ensures flexibility and modularity in simulation design.

Simulations are executed by the SG engine, which uses an internal constraint solver to manage the allocation and scheduling of CPU and network resources throughout the simulation. This approach ensures that the dynamics of resource contention and sharing are accurately modelled, enabling precise evaluations of algorithmic performance. For a detailed overview of SG's architecture and capabilities, we refer readers to [26].

SG was chosen as the foundation for VMPS for several compelling reasons. First, SG has a reputation for its performance and realism within the research community. Its thorough modelling capabilities and powerful simulation engine make it an ideal choice for studying large-scale distributed systems. Second, SG has recently been enhanced to include VM abstractions and a live migration model [27]. These features are particularly valuable for our work, as they allow researchers to manipulate VMs in a manner consistent with real-world operations. This includes creating, destroying, starting, shutting down, suspending, resuming, and migrating VMs.

The live migration model provided by SG is a fundamental component that distinguishes it from other simulation systems. It accurately estimates the time and network traffic associated with VM relocations by accounting for resource contention and memory refresh rates during migration events. This level of precision is crucial for evaluating the performance of VM placement algorithms, as it reflects the complexities of real-world scenarios involving resource sharing and dynamic workloads.

By leveraging these capabilities, we developed VMPS to facilitate the simulation and analysis of VM placement strategies at scale. SG's support for modelling VM behaviour and live migration was instrumental in achieving this objective. With VMPS, researchers can conduct in-depth analyses of placement algorithms under realistic conditions, such as fluctuating workload patterns, resource contention, and large-scale infrastructure configurations. The overall workflow of VMPS is illustrated in Figure 1.

The adoption of SG as the foundational framework for VMPS ensures that our simulation tool is not only robust and accurate but also extensible and adaptable to the evolving research demands in the field of cloud computing.

4. VM Placement Simulator

The **VM Placement Simulator (VMPS)** is designed with two primary objectives:

1. To free researchers from the complex details of handling VM creation and dynamic workload variations while evaluating novel VM placement algorithms.

2. To facilitate a robust evaluation of these algorithms under consistent and reproducible conditions.

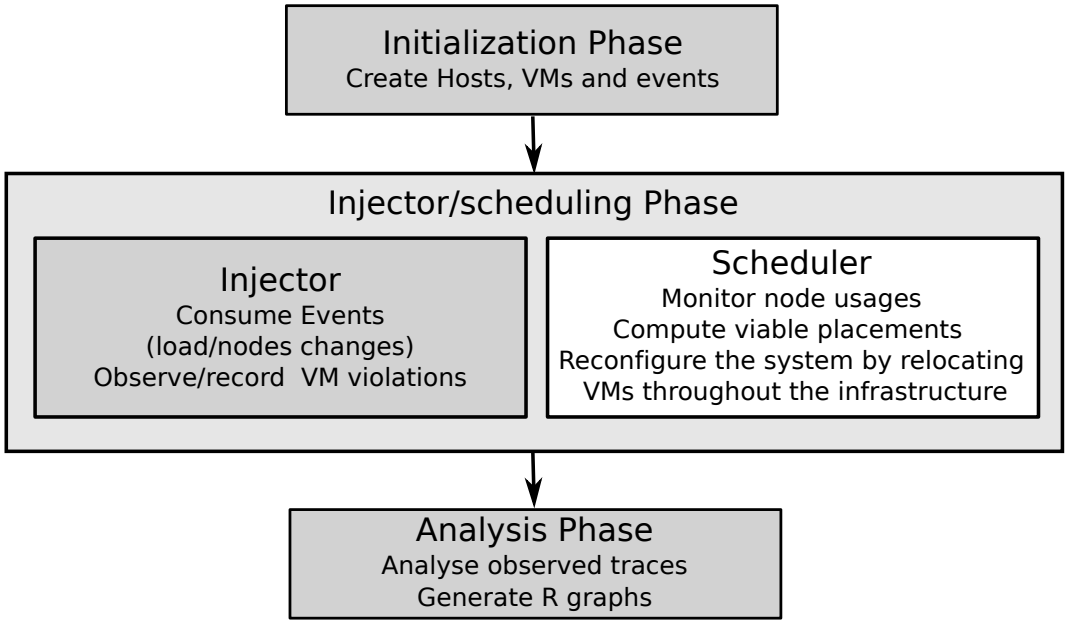


Figure 1: Overall workflow of the VMPlaceS simulation framework. The three main phases are: Initialisation (creating VMs and event queue), Injection (dynamic workload changes), and Trace Analysis (collecting and visualising metrics).

4.1 Overview

VMPS is implemented in **Java**, using the **SimGrid (SG) MSG API** to build its core functionalities. Although Java introduces a slight overhead to SG's performance, its choice is deliberate. Java significantly simplifies the development of advanced scheduling policies by providing a flexible and developer-friendly programming environment. For example, complex proposals such as *Snooze* were implemented in Java, while the *DVMS* approach was implemented using both *Scala* and Java. This demonstrates the versatility and ease of Java for implementing complex algorithms.

VMPS operates through a structured three-stage simulation process, as outlined in Figure 1:

1. **Initialisation Phase:** Setting up the simulation environment, creating VMs, and generating the event queue.
2. **Injection Phase:** Dynamically managing workload changes and events during the simulation.
3. **Trace Analysis Phase:** Analysing simulation data and visualising performance metrics.

The simulator uses at least two SG processes to perform its operations:

1. The **Injector Process**, which handles event injection during the simulation.
2. The process running the **scheduling algorithm** under evaluation.

Researchers can implement their scheduling algorithms using the SG MSG API together with a set of specific interfaces provided by VMPS. These interfaces include key classes such as:

- **XHost:** An extension of SG's `Host`, representing the simulated hosts.
- **XVM:** An extension of SG's `VM`, encapsulating VM-specific attributes.

- **SimulatorManager:** A class that manages interactions between simulator components, enabling users to monitor the infrastructure's state, including metrics such as host load, the number of hosted VMs, and overload conditions.

4.2 Initialisation Phase

The **Initialisation Phase** begins by creating n VMs, which are distributed across the first p hosts listed in the platform configuration file using an initial co-located placement. The platform file defines the cluster topology, specifying h hosting nodes and s service nodes. These parameters n, h, and s are input by users via a configuration file.

The default platform configuration simulates a cluster, but users can define more complex topologies, such as centralised data centres, to suit their research scenarios.

Each VM is launched based on pre-defined templates called **VM classes**, which specify essential attributes:

- **Number of Cores:** `nb_cpus`
- **Memory Size:** `ramsize`
- **Network Bandwidth:** `net_bw`
- **Migration Bandwidth:** `mig_speed`
- **Memory Update Speed:** `mem_speed`

The `net_bw` parameter defines the maximum bandwidth available to a VM. However, when multiple VMs on the same physical host communicate concurrently, the available bandwidth is shared according to SimGrid's flow-level network model, which accurately captures contention and link sharing. This ensures that migration traffic and normal VM communication interfere realistically.

These attributes govern the VM's behaviour, especially during migrations. For instance, the memory update speed (`mem_speed`) directly impacts migration time and the volume of transferred data. Users can customise these templates to simulate different workloads, including memory-intensive tasks.

During initialisation, all VMs start with a CPU load of zero. Their workloads evolve dynamically throughout the simulation, driven by events injected in subsequent phases. Once the VMs are created and assigned, VMPS generates two basic SG processes:

1. The **Injector Process**, which produces the event queue.
2. The process executing the scheduling algorithm under evaluation.

The event queue is populated with **CPU load change events**, generated at intervals following an exponential distribution with rate parameter λ_t. The load for each event is drawn from a Gaussian distribution defined by a mean (μ) and standard deviation (σ). These parameters λ_t, μ, and σ are user-defined, ensuring flexibility in simulation scenarios.

To guarantee reproducibility, all random processes in VMPS are initialised with a seed specified in the configuration file. Researchers can extend the event framework by creating new event classes implementing the `InjectorEvent` interface and adding the corresponding event generation logic. For instance, future releases of VMPS will include *node addition/removal events* to simulate scenarios like crashes or dynamic scaling.

4.3 Injector Phase

Once the initialisation is finished and the global event queue is ready, the **Injector Phase** begins. The injector process iteratively consumes events from the queue, dynamically modifying the VM workloads by creating and assigning new SG tasks to the VMs. These tasks alter the CPU load, which directly affects migration times and memory update speeds. In other words, the injector dynamically oversees the resource demands of each VM throughout the simulation.

Based on decisions made by the scheduler, VMs may undergo actions such as suspension, resumption, or migration across available hosts. Researchers must implement both the scheduling algorithm to address the VM placement problem and the logic to execute reconfiguration plans. These plans, managed through the `SimulatorManager` class, are critical as reconfiguration costs are integral to the performance of dynamic placement systems.

Critically, VMPS does not simulate the computational time of the scheduler. Instead, it invokes the scheduler execution directly, capturing the actual computation time. However, all workload changes and reconfiguration actions (e.g., suspend, resume, migrate) are simulated using SG.

4.4 Trace Analysis

The **Trace Analysis Phase** processes the data collected during the simulation. VMPS extends the SG Trace module to record comprehensive infrastructure metrics, such as VM and host loads, the frequency and duration of SLA violations, the number of migrations, and the scheduler's invocation success rate.

This data is stored in two formats:

1. **Visualisation Files:** Compatible with community tools like **PajeNG** for graphical analysis.
2. **JSON Trace Files:** Used for generating detailed resource usage visualisations in the **R** statistical environment.

The Trace module is fully extensible, allowing researchers to define and record custom variables specific to their algorithms. For example, additional metrics can be recorded to evaluate specific scheduling behaviours or workload characteristics.

By enabling fine-grained trace analysis and visualisation, VMPS provides researchers with valuable insights into the performance and efficiency of their VM placement algorithms, paving the way for robust and informed comparisons.

5. Dynamic Virtual Machine Placement Problem (VMPP) Algorithms

To demonstrate the applicability and effectiveness of VMPS in addressing the complexities of dynamic virtual machine placement, we implemented three distinct algorithms: a centralised approach inspired by the Entropy proposal [4], a hierarchical arrangement based on the Snooze framework [3], and a fully distributed strategy using the principles of DVMS [5]. Each approach represents a unique methodology to resolve resource allocation violations arising from overloaded nodes. These algorithms are designed to maintain system performance and ensure efficient utilisation of resources in a simulated environment.

The issue of overloaded nodes arises when the virtual machines (VMs) hosted on a physical server collectively demand computational resources exceeding the server's capacity, especially CPU power. Such scenarios disrupt system equilibrium and require prompt reconfiguration. All three algorithms implemented here rely on a common solver component that computes and applies reconfiguration plans to

restore system stability. In particular, we used the solver developed within the Entropy framework [28] as the core reconfiguration engine for all approaches. This choice ensures that any performance differences come from the placement logic itself, not from solver efficiency. The solver iteratively evaluates potential configurations within a predefined timeout. If the timeout is reached, it applies the best solution found, initiating live migrations to execute the reconfiguration in the simulated environment.

In the sections below, we provide a comprehensive overview of each approach, elaborating on their unique models, operational strategies, and key benefits.

5.1 Centralised Approach: Entropy-Based Algorithm

The centralised placement algorithm, rooted in the Entropy framework, operates as a single, cohesive entity that governs the entire system from a designated service node. This approach involves deploying a single SG process tasked with monitoring resource utilisation and assessing the current configuration's viability. At regular intervals, this process invokes the Entropy VMPP solver to effectively identify potential resource allocation violations and resolve them.

Resource monitoring is performed through direct access to the states of hosts and their respective VMs. Metrics such as CPU usage and memory utilisation are used to evaluate whether any node is operating beyond its capacity. If the solver detects a need for reconfiguration, it determines the optimal migration plan, minimising the number of migrations required while ensuring system stability.

Once a reconfiguration plan is formulated, the framework records several key metrics, including:

- The number of migrations performed during the reconfiguration process.
- The total time taken to execute the reconfiguration.
- Whether the reconfiguration successfully resolved all resource allocation violations or introduced new ones.

This centralised approach is particularly suitable for small-scale environments where the overhead of a single monitoring and decision-making entity does not significantly impact performance. Its simplicity and determinism make it an excellent benchmark for comparing other, more complex strategies.

5.2 Hierarchical Approach: Snooze Based Algorithm

The Snooze framework employs a hierarchical plan to handle the placement and load balancing of VMs efficiently. The architecture is organised into three distinct levels, each with specific responsibilities:

1. **Group Leaders (GLs):** At the highest level, GLs aggregate monitoring information from the intermediate layer and maintain a global view of the system state. This centralised perspective enables them to make informed decisions about resource allocation.
2. **Group Managers (GMs):** Serving as intermediaries, GMs manage groups of nodes and relay data between the GLs and Local Controllers. They are responsible for coordinating the activities of nodes within their domain.
3. **Local Controllers (LCs):** Located at the lowest level, LCs handle individual nodes and manage the VMs assigned to them. They also monitor the nodes' resource utilisation and report it to the higher layers.

Communication between these layers is structured and periodic. Monitoring data flows upward from LCs to GMs and finally to GLs, ensuring that higher layers receive timely updates about system load and other parameters. Conversely, reconfiguration commands and heartbeats are disseminated downward, enabling efficient command dispersal.

In our implementation, Snooze's core functionality was modelled using Simgrid's event-handling primitives. Multicast communication, a critical feature for disseminating updates across the hierarchy, was implemented as a dedicated service. This service ensures concurrent state propagation to multiple receivers, maintaining system responsiveness even under high loads.

Snooze's hierarchical design strikes a balance between scalability and manageability, making it an ideal choice for medium- to large-scale environments. By distributing decision-making responsibilities across multiple layers, the architecture reduces pressure on individual components while retaining a degree of centralised oversight.

5.3 Distributed Approach: DVMS-Based Algorithm

The Distributed Virtual Machine Scheduler (DVMS) [5] represents a fully decentralised approach to VM placement, emphasising collaboration among nodes to achieve scalable and fault-tolerant resource management. Unlike centralised and hierarchical systems, DVMS operates without a single point of control, instead relying on a network of agents deployed across nodes.

Each node in the system hosts a DVMS agent responsible for monitoring its resource usage, managing its VMs, and cooperating with neighbouring agents. Communication between agents is facilitated via an overlay network, which defines inter-node relationships and supports efficient data sharing.

When a node experiences resource constraints, it initiates an *Iterative Planning Method (ISP)*:

1. The node assumes the role of the problem-solving leader and identifies its nearest neighbour in the overlay network.

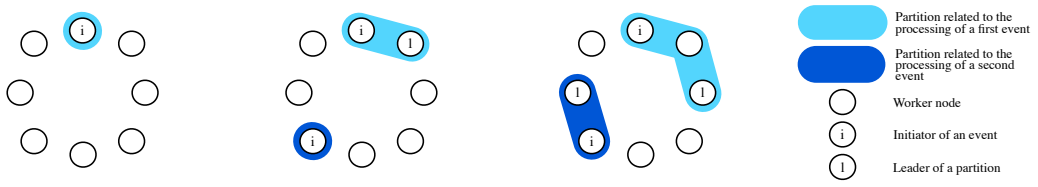
2. If the neighbour is already part of another partition, the algorithm proceeds to the next available neighbour. Otherwise, the neighbour joins the partition and becomes the new leader.

3. The leader gathers resource utilisation data from all partition members and attempts to devise a reconfiguration plan. If no solution is found within the current partition, the algorithm iteratively expands to include additional nodes.

This iterative, partition-based strategy ensures that reconfiguration tasks are distributed and parallelised, minimising overall resolution time. By forming small, independent partitions, DVMS achieves high scalability and responsiveness, even in large and dynamic environments. The efficiency of simultaneous event handling in DVMS is visually depicted in Figure 2.

The DVMS implementation in VMPS was developed using SCALA, leveraging Simgrid's Java primitives for inter-agent communication. The overlay network was modelled as an unstructured topology, enabling flexible neighbour relationships and rapid adaptation to dynamic system conditions.

This decentralised methodology is particularly effective in large-scale, heterogeneous settings where centralised solutions may struggle with the scale and diversity of resource demands. Figure 3 illustrates the architecture of the complementary hierarchical system Snooze, while Figure 4 provides a comparison between simulated and real-world execution behaviour. Figure 5 presents a detailed scalability and responsiveness analysis across centralised, hierarchical, and distributed systems.



(a) (b) (c) (d) Legend

Figure 2: Processing two events simultaneously

Moreover, the performance implications of group size variations in hierarchical placements are explored in Figure 6. Quantitative assessments of failure rates and computation times across varying infrastructure sizes and configurations are reported in Tables 1 and 2, respectively, providing empirical support for the advantages of distributed techniques.

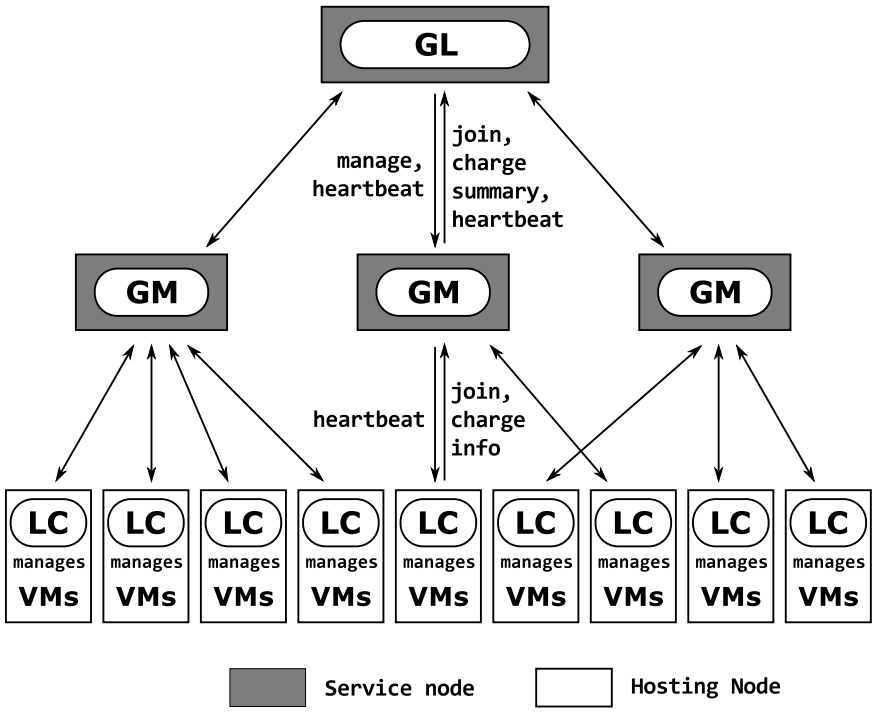


Figure 3: Snooze Architecture – hierarchical organisation with Group Leaders, Group Managers, and Local Controllers.

Table 1: Failed Reconfigurations

Size	2	4	8	32
128	19	0	0	0
256	29	0	0	0
512	83	1	0	0
1024	173	7	0	0

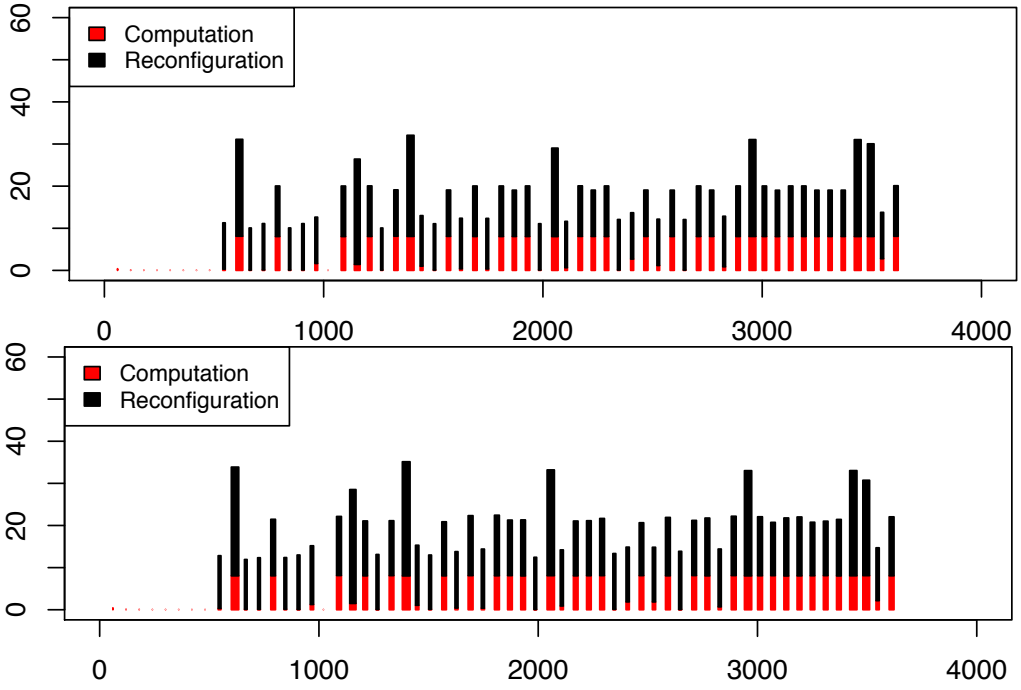
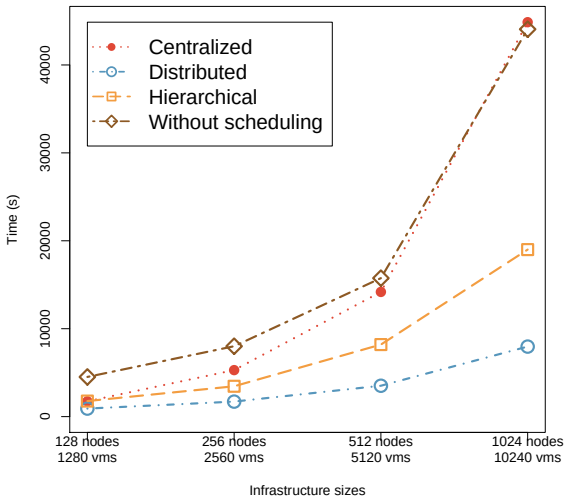


Figure 4: Comparison between simulated (top) and *in-vivo* (bottom) executions.



Size	C	H	D
128	21.26	21.07	9.55
256	40.09	21.45	9.58
512	55.63	24.54	9.57
1024	81.57	29.01	9.61

Size	C	H	D
128	3.76	2.52	0.29
256	7.97	2.65	0.25
512	15.71	2.83	0.21
1024	26.41	2.69	0.14

Size	C	H	D
128	10.34	10.02	10.01
256	10.26	10.11	10.01
512	11.11	10.28	10.08
1024	18.90	10.30	10.04

Figure 5: Scalability/Reactivity analysis of Entropy, Snooze and DVMS

Table 2: Computation Duration ($\mu \pm \sigma$)

Size	2 LCs	4 LCs	8 LCs	32 LCs
128	0.16 ± 1.23	0.34 ± 1.81	0.58 ± 2.40	2.53 ± 4.62
256	0.18 ± 1.31	0.42 ± 1.99	0.66 ± 2.50	2.65 ± 4.69
512	0.15 ± 1.20	0.33 ± 1.78	0.67 ± 2.54	2.83 ± 4.98
1024	0.19 ± 1.37	0.42 ± 2.02	0.89 ± 2.90	2.69 ± 4.91

'±' indicates maximum positive deviation from mean.

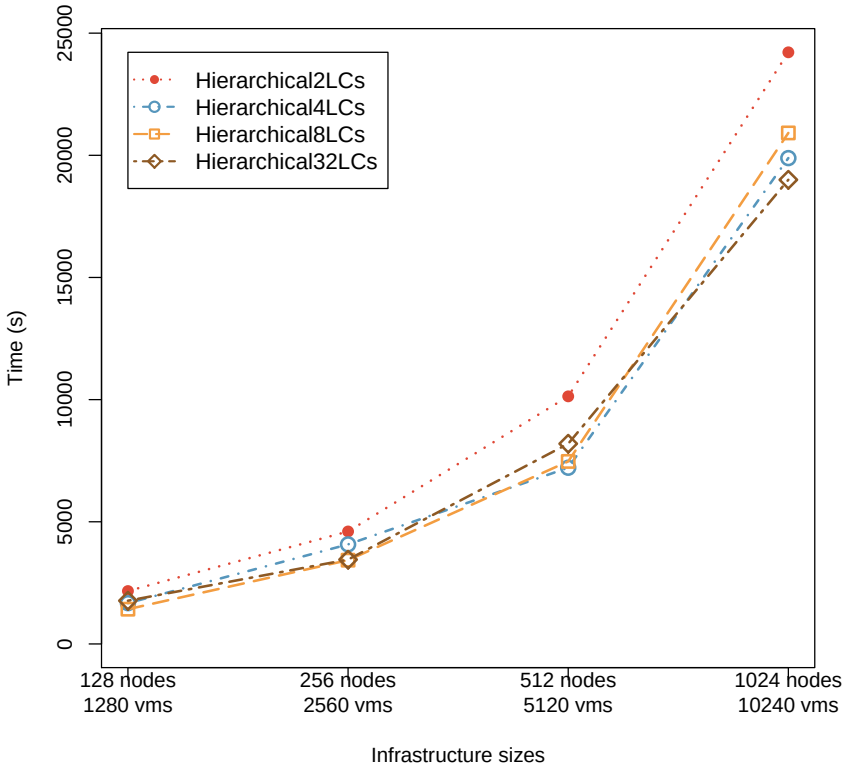


Figure 6: Hierarchical placement: influence of varying group sizes

6. Experiments

We conducted two classes of experiments to validate the effectiveness of VMPS: one focusing on accuracy and the other demonstrating its application in analysing the Entropy, Snooze, and DVMS strategies.

6.1 Accuracy Evaluation

To evaluate the precision of VMPS, experiments were performed using real-world scenarios on the Graphene cluster. The experiments employed the Entropy strategy, executed at 10-second intervals over a one-hour period, comparing simulated and *in-vivo* outcomes.

The Graphene cluster consisted of nodes with Intel Xeon X3440 CPUs, 16 GB memory, and GbE NICs, each hosting 6 VMs. VM configurations varied across 8 predefined classes, with memory update rates ranging from 0% to 80% of migration bandwidth. Load profiles followed uniform and Gaussian distributions, refreshed every 10 seconds. Simulations mirrored these conditions, modelling network parameters to reflect Graphene's performance.

As illustrated in Figure 4, the duration of the Entropy algorithm phases under both environments shows close alignment. Simulations closely matched real-world results, with reconfiguration times differing by 6%–18% (median 12%). Variations primarily stemmed from bandwidth fluctuations during concurrent migrations, highlighting potential refinements in the simulation model. Nonetheless, the accuracy of the simulations was deemed sufficient for performance trend analysis. Although simulation outcomes were closely aligned with in-vivo executions, it is possible that rare, high-impact events not present in the experimental scenarios may produce divergent behaviours in real-world deployments.

6.2 Analysis of Entropy, Snooze, and DVMS

We compared the three approaches through simulations across clusters ranging from 128 to 1024 PMs, each hosting 10 to 80 VMs per node. Simulations employed homogeneous PMs with 8-core CPUs, 32 GB RAM, and 1 Gbps NICs. Entropy and Snooze used service nodes, executing periodic invocations every 10 seconds. The Snooze hierarchical strategy deployed one Group Manager (GM) per 32 Local Controllers (LCs), as represented in Figure 3.

6.2.1 General Comparison

Figure 5 presents aggregated violation durations, as well as computation and reconfiguration times. Entropy, due to its centralised nature, struggled with scalability, displaying higher overhead in larger setups. Snooze performed better owing to its hierarchical design. However, DVMS outperformed both, benefiting from its dynamic and localised partitioning mechanism. This efficiency is further exemplified by its ability to process multiple events in parallel, as depicted in Figure 2.

6.2.2 Algorithm Variations and Insights

We further investigated Snooze's performance with differing LC group sizes (2, 4, 8, 32 LCs per GM). Smaller group sizes resulted in higher violation times due to resource scarcity, while larger groups improved reconfiguration success but increased computation times. DVMS dynamically balanced these trade-offs by adjusting partition sizes based on the load.

VMPS enabled comprehensive evaluation of algorithm variations, metrics such as migration overhead, and scalability, supporting simulations of up to 8K PMs and 80K VMs. Future work includes refining the Snooze protocol and further improving simulation efficiency in collaboration with SG developers.

7. Discussion and Future Work

7.1 Modelling Network Congestion and Latency

VMPlaceS currently employs simplified network models; however, upcoming releases will incorporate congestion-aware routing algorithms and latency mapping based on empirical traces from cloud data centres.

7.2 Integration of Real Workload Traces

The framework is designed to support real workload traces, such as from the Google Cluster dataset. Integrating these traces can improve simulation realism but may impact scalability due to complexity in workload variability.

7.3 Scalability to Hyperscale Data Centres

Scaling beyond 100,000 PMs introduces challenges like memory limits and scheduling overheads. VMPlaceS plans to integrate lightweight event handling and dynamic resource throttling for efficient scaling.

7.4 SLA Violation Measurement

SLA violations are measured as a function of delayed service delivery and unmet QoS requirements. These are modelled to match industry-standard SLA definitions to reflect practical implications.

7.5 Extension to Other Cloud Management Tasks

We plan to expand the framework to support energy-efficient scheduling, dynamic fault recovery, and resource migration using predictive modelling techniques.

7.6 Handling Computational Complexity

To manage complexity, VMPlaceS uses event-based simulation and parallel processing. Future versions will support distributed simulations using MPI for large-scale runs.

7.7 Impact of Statistical Workload Models

Synthetic workloads using exponential and Gaussian distributions provide a controlled and reproducible environment for simulation. However, such models may not fully capture certain complex and abrupt real-world behaviours, such as sudden traffic spikes, anomalous user activities, or hardware-induced failures. These phenomena can significantly affect VM placement performance but are difficult to emulate using purely statistical processes. To address this, future extensions of VMPlaceS will incorporate hybrid workload generation, blending synthetic distributions with empirical traces from production cloud environments, thereby enabling evaluation under a broader spectrum of operating conditions.

8. Conclusion

In this work, we have introduced *VMPS*, a comprehensive and scalable framework designed to support the development, simulation, and evaluation of Virtual Machine (VM) placement algorithms. The framework offers extensive capabilities for defining generic placement strategies, executing large-scale simulations across diverse infrastructures, and performing trace-based analyses to assess algorithm effectiveness. The flexibility of *VMPS* enables it to serve a wide range of use cases, from academic research to real-world deployments in complex virtualised environments.

A key contribution of this work is the validation of the framework's accuracy and reliability. To this end, we conducted a detailed comparison between simulated VM placement scenarios and real-world (*in-vivo*) executions using the Entropy placement strategy. As shown in Figure 4, the simulation outcomes closely matched actual execution patterns, with reconfiguration durations differing by a median of approximately 12%. This high fidelity affirms that *VMPS* provides a trustworthy simulation environment for experimentation and optimisation of placement strategies. Furthermore, we demonstrated the versatility and relevance of *VMPS* by evaluating a diverse set of placement algorithms representing three major virtualisation paradigms: centralised, hierarchical, and fully distributed. These approaches are widely used across industry and academia. Our evaluation included environments with up to 1,000 nodes and 10,000 VMs, offering valuable insights into algorithmic behaviour at scale. The comparative results presented in Figure 5 highlight the performance trade-offs, with the distributed DVMS approach consistently outperforming others in terms of violation time, computation, and reconfiguration durations.

Additional insights are provided in Table 1 and Table 2, which detail the number of failed reconfigurations and the corresponding computation durations across varying hierarchical configurations and

infrastructure sizes. These metrics confirm the robustness of distributed approaches, particularly in minimising failures and ensuring low-latency decision making.

The influence of hierarchical group sizes on system performance is further illustrated in Figure 6, showing how optimal group sizing can significantly reduce violation time in hierarchical models like Snooze.

Looking ahead, we are actively collaborating with the core developers of the *SG* platform to enhance the scalability of *VMPS*, targeting simulation capabilities for environments containing up to 100,000 PMs and 1,000,000 VMs. This scale-up will facilitate modelling of hyperscale data centres and cloud infrastructures.

Beyond scaling, we aim to enrich the simulation capabilities of *VMPS* by incorporating factors such as dynamic network behaviour and hard disk I/O performance. This will enable more realistic modelling of operational conditions, including network congestion and storage bottlenecks key factors in real-world VM placement. Another promising direction involves developing an API for dynamic VM provisioning and removal during simulation runtime. This feature will allow users to emulate more realistic and adaptive workloads by dynamically modifying the simulation context, supporting VM migration and load rebalancing under operational conditions.

In conclusion, *VMPS* represents a significant advancement in VM placement simulation and analysis. Its ability to accurately model and evaluate placement strategies at scale provides essential insights into the behaviour and efficiency of virtualisation systems. With continued development and enhancements, *VMPS* holds great promise as a critical tool for optimising VM placement in complex, resource-intensive environments.

While the proposed framework achieves high fidelity for typical workload patterns, it is important to acknowledge that the synthetic workloads employed may not fully represent extreme or anomalous events frequently observed in operational cloud systems. Examples include diurnal usage surges, correlated application failures, and sudden network congestion. In future work, we aim to model these scenarios explicitly through a combination of real trace integration, anomaly injection, and fault simulation.

Data Availability

This study does not use any real-world or third-party datasets. All results presented are based on synthetically generated data produced using the simulation framework described in detail in the manuscript. The synthetic workloads were created using statistical distributions (e.g., exponential inter-arrival and Gaussian CPU load models), and therefore, no external dataset exists. The simulation configuration files and scripts are available from the corresponding author upon reasonable request.

References

[1] R. Moreno-Vozmediano et al. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer Journal*, 45(12), March 2012.

[2] R. Birke et al. Multi-resource characterization and their (in)dependencies in production datacenters. In *IEEE NOMS'14*, May 2014.

[3] Eugen Feller et al. Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds. In *IEEE CCGRID'12*, May 2012.

[4] Fabien Hermenier et al. Entropy: A Consolidation Manager for Clusters. In *VEE'09: Virtual Execution Environments*, New York, NY, USA, 2009. ACM.

[5] Flavien Quesnel, Adrien Lebre, and Mario Südholt. Cooperative and Reactive Scheduling in Large-Scale Virtualized Platforms with DVMS. *Concurrency and Computation: Practice and Experience*, 25(12):1643–1655, August 2013.

[6] Hien Nguyen Van, F.D. Tran, et al. SLA-aware virtual resource management for cloud infrastructures. In *IEEE CIT'09*, October 2009.

[7] Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *IEEE INFOCOM'11*, April 2011.

[8] Adam Barker et al. Academic Cloud Computing Research. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2014)*, June 2014.

[9] Rafsan Uddin Beg Rizan. Automatic parsing and extraction of structured data from fax images. In *2024 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–7, 2024. doi: 10.1109/ICEET65156.2024.10913905.

[10] Rafsan Uddin Beg Rizan. Mus490: Advanced feature selection and unsupervised clustering for music categorization. In *2024 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–6, 2024. doi: 10.1109/ICEET65156.2024.10913486.

[11] Rafsan Uddin Beg Rizan. Evaluating 3-d features for model matching. In *2024 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–7, 2024. doi: 10.1109/ICEET65156.2024.10913962.

[12] Rafsan Uddin Beg Rizan. Enhancing potato blemish detection through interactive image segmentation and classification. In Phayung Meesad, Sunantha Sodsee, Watchareewan Jitsakul, and Sakchai Tangwannawit, editors, *Proceedings of the 21st International Conference on Computing and Information Technology (IC2IT 2025)*, pages 155–168, Cham, 2025. Springer Nature Switzerland. doi: 10.1007/978-3-031-90295-6_16.

[13] Jyoti Singh. Ihpp: An in-depth profiling tool for advanced performance analysis. Technical report, OSF Preprints, 2025.

[14] Jyoti Singh. Transactional memory without hardware support: A comprehensive software-driven approach. Technical report, OSF Preprints, 2025.

[15] Jyoti Singh. Comparative analysis of open-source and closed-source development models using agent-based simulation. Technical report, OSF Preprints, 2025.

[16] Jyoti Singh. Web application development and cybersecurity: Integrating apis, logging, and defense mechanisms. Technical report, OSF Preprints, 2025.

[17] Akshar Patel. Optimizing knowledge graph embeddings for enhanced question answering performance. In *2024 International Conference on Intelligent Computing, Communication, Networking and Services (ICCNS)*, pages 191–199, 2024. doi: 10.1109/ICCNS62192.2024.10776440.

[18] Akshar Patel. Robotic hand-eye coordination fusion. In *2024 Fifth International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*, pages 95–102, 2024. doi: 10.1109/IDSTA62194.2024.10747010.

[19] Akshar Patel. Evaluating attack thresholds in proof of stake blockchain consensus protocols. In *2024 4th Intelligent Cybersecurity Conference (ICSC)*, pages 87–94, 2024. doi: 10.1109/ICSC63108.2024.10895793.

[20] Akshar Patel. Global optimization for quality of service in internet video streaming: Design, implementation, and evaluation. In *2024 International Conference on Intelligent Computing, Communication, Networking and Services (ICCNS)*, pages 1–10, 2024. doi: 10.1109/ICCNS62192.2024.10776453.

[21] Niketa Penumajji. Sound emotion recognition using deep learning. *arXiv Preprints*, 2025.

[22] Niketa Penumajji. Hbsp: A lightweight framework for transparent software protection using hardware virtualization. Technical report, Preprints, 2025.

[23] Niketa Penumajji. Towards equilibrium in human-ai collaboration: A dynamic model of performative feedback and empirical insights. Technical report, Preprints, 2025.

[24] Niketa Penumajji. Benchmarking probabilistic modeling methods for protein fitness prediction and uncertainty quantification. Technical report, Preprints, 2025.

[25] Niketa Penumajji. Optimizing broadcast scheduling in social media: A nonlinear integer programming approach. Technical report, Preprints, 2025.

[26] Henri Casanova et al. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Parallel and Distributed Computing*, 74(10), June 2014.

[27] Takahiro Hirofuchi, Adrien Lebre, and Laurent Pouilloux. Adding a live migration model into simgrid: One more step toward the simulation of infrastructure-as-a-service concerns. In *CloudCom'13: Cloud Computing Technology and Science*. IEEE, 2013.

[28] Fabien Hermenier, Sophie Demassey, and Xavier Lorca. Bin Repacking Scheduling in Virtualized Datacenters. In *CP'11: Constraint Programming*, LLNCS. Springer, 2011.