Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Agent-based cloud simulation model for resource management

Dapeng Dong[1*]

Abstract

Driven by the successful service model and growing demand, cloud computing has evolved from a moderate-sized data center consisting of homogeneous resources to a heterogeneous hyper-scale computing ecosystem. This evolution has made the modern cloud environment increasingly complex. Large-scale empirical studies of essential concepts such as resource allocation, virtual machine migration, and operational cost reduction have typically been conducted using simulations. This paper presents an agent-based cloud simulation model for resource management. The focus is on how service placement strategies, service migration, and server consolidation affect the overall performance of homogeneous and heterogeneous clouds, in terms of energy consumption, resource utilization, and violation of service-level agreements. The main cloud elements are modeled as autonomous agents whose properties are encapsulated. The complex relationships between components are realized through asynchronous agent-to-agent interactions. Operating states and statistics are displayed in real time. In the evaluation, the efficiency of the simulator is studied empirically. The performance of various resource management algorithms is assessed using statistical methods, and the accuracy of server energy consumption models is examined. The results show that agent-based models can accurately reflect cloud status at a fine-grained level.

Keywords Cloud, Simulator, Agent-based modeling, Resource management

Introduction

The widespread adoption of cloud services and the advancement of cloud-enabling technologies have driven the development of cloud computing into a heterogeneous hyper-scale computing ecosystem, and with it, reducing operational costs and improving user experience have become two of the most concerning aspects for cloud service providers. Software-based solutions typically focus on developing and applying optimization algorithms to address specific system objectives, such as minimizing energy consumption, maximizing resource utilization, and preventing violations of service-level agreements. These objectives are often formulated as multi-objective optimization problems. Experiments and evaluations in production environments can be challenging for current software-based solutions. Instead, cloud resource management studies have primarily been conducted using simulators or analytical methods.

Current simulation methods used in cloud resource management span a range of abstraction levels, including system dynamic modeling, agent-based modeling, and discrete-event modeling [1]. System dynamic modeling is considered a strategic method suitable for simulating macro-level phenomena in social networks, economies, and ecosystems. In contrast, discrete-event simulation deals with detailed models where the state of the system changes when an event occurs (i.e., next-event temporal advance approach) or at a fixed interval (fixed-increment temporal advance approach) [2]. Several modern cloud simulators such as CloudSim [3] and CloudSim Plus [4], are implementations of discrete-event models. Despite this complexity, creating viable cloud models remains

*Correspondence:
Dapeng Dong
Dapeng.Dong@xjtlu.edu.cn; Dapeng.Dong@liverpool.ac.uk
[1] Department of Communications and Networking, Xi'an Jiaotong-Liverpool University, Suzhou, China

Springer Open

© The Author(s) 2023. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

crucial for the success of any simulation. As abstract representations of actual systems, these models can only provide approximations and generate statistical insights. Cloud modeling has become increasingly challenging due to the growing number of elements, functions, and complexity of interactions among functional components. Additionally, for certain problems, there may be no effective analytical solution, such as systems with Poisson arrival rates, general distributions of service times, and K servers ($K > 1$), i.e., the $M/G/K$ model [1].

In comparison, agent-based models are able to account for the emergence of complex systems through simple rules for interaction between agents and between agent and environment [5, 6]. If elements of clouds such as services, servers, and other functional components are considered as agents, an agent-based model of clouds can be created.

This paper presents an alternative to existing cloud models by employing agent-based modeling techniques. Agent-based models are widely used in social sciences to study population dynamics resulting from collective behavior among individuals with diverse traits, enabling the model to effectively manage a large number of heterogeneous elements, which is essential for simulating complex and diverse clouds. For instance, this approach can account for varying resource utilization patterns, deployment methods, and configurations for each service or server within the cloud infrastructure. Unlike other models that rely on prior knowledge of system trends (as required by equation-based methods) or predicting system states for the next step (as demanded by discrete-event simulation techniques), agent-based modeling focuses on describing individuals without imposing such constraints. This enables a more flexible and adaptive representation of complex systems, allowing researchers to examine both individual and collective behaviors in real time. Furthermore, as agent interactions with their environment are continuously recorded during the execution of these models, it becomes possible to perform comprehensive analyses using statistical methods, rather than relying on a single set of final results.

The main contributions of this study are summarized as follows.

- Cross-platform simulation model: A highly configurable agent-based simulation model for clouds was developed. The source code can be viewed at [7]. This model allows researchers and practitioners to experiment with different cloud management strategies across various configurations of environments, providing valuable insights into the performance and efficiency of different approaches in a simulated environment.
- Balanced-fit algorithm: A new balanced-fit algorithm was developed and evaluated. The algorithm is designed to optimize resource allocation in dynamic, heterogeneous cloud environments by minimizing both underprovisioning (balancing) and overprovisioning (fitting).
- Quantification and allocation strategy for SLA violations: A strategy for penalizing violations of service-level agreements was proposed and evaluated. This approach introduces a novel method for measuring and redistributing penalties among services operating on servers experiencing excessive load.
- Effects of server migration and consolidation: The study also investigated the effects of service migration and server consolidation in both homogeneous and heterogeneous cloud environments. By analyzing these scenarios, researchers can gain valuable insights into how different configurations of servers and workloads can impact overall system performance, efficiency, and cost-effectiveness.

The organization of the remainder of the paper is as follows. Related work section discusses related work and several representative developments in the field. Architecture section presents the architecture and design of the simulator. Resource optimization algorithms section discusses important resource optimization algorithms and management strategies. Evaluation section presents evaluations of the algorithms and the simulator. Discussion section discusses the potentials and limitations of the work. Finally, Conclusion section concludes this study.

Related work

As consumer needs continue to drive innovation in modern cloud computing, the landscape has evolved significantly from its early days when sharing compute resources using the Xen hypervisor was a primary method [8, 9]. Today's clouds encompass an array of heterogeneous hardware and software components, platforms, services, and management frameworks that have collectively contributed to their increasing complexity. This complexity has propagated interest among researchers who are tackling almost every aspect of cloud computing, such as service reliability [10], predicting resource utilization based on time-series data [11, 12], and developing cost-effective scheduling algorithms [13]. As widely acknowledged in the community, one of the most effective methods for studying clouds is through simulation. In this section, several representative cloud simulators that have contributed significantly to our understanding of these complex systems are discussed.

CloudSim [3] is an influential cloud simulator. Since its first publication, its functionality has been enhanced to include support for simple energy-aware virtual machine placement as well as more advanced features such as federated data centers with customized networking topologies, message-passing applications, and automatic scaling. It has been used in many studies investigating resource allocation [14], energy efficiency [15] and operational costs algorithms [16, 17]. Many simulators have been derived from CloudSim with extended features, such as the CloudAnalyst, which offers a graphical user interface, high degrees of flexibility for simulation definition, and replay mechanisms [18]. CloudSim Plus is a re-engineered and refactored version of CloudSim, providing better code clarity and improved accuracy [4]. It is important to note that when using CloudSim Plus, the sampled state values may not always align well with the current status of the simulated system. Achieving statistical soundness in results typically requires long-run simulations. In contrast, the proposed cloud model in this work can output more accurate system states in real time. This is particularly important for machine learning-based resource optimization algorithms. For example, deep reinforcement learning has been used to study cloud resource management [19–21]. One of the fundamental requirements is that the system states should accurately reflect the effects after applying policies (such as a virtual machine placement schema or server consolidation interval adjustment) to the system.

In addition to discrete-event simulators that emphasize simulation scalability and speed, fine-grained cloud models such as GreenCloud [22] and iCanCloud [23] provide detailed analysis of energy consumption in data center IT equipment (e.g., servers and switches) with higher accuracy. GreenCloud integrates the NS-2 network simulator [24], allowing for an analysis of energy consumption associated with communication patterns at packet-level, as well as the effectiveness of low-level power management mechanisms such as voltage scaling and frequency scaling. iCanCloud was built on top of the OMNeT++ platform [25]. The simulator facilitates the evaluation of various cloud architectures, storage systems, and virtual machine configurations using trace logs of real applications. Certainly, with many details included, simulation speed has to be sacrificed.

On the other hand, several studies have been conducted in real-world environments. For instance, authors in [26] proposed a workload-aware performance model for serverless computing and evaluated it on Amazon Lambda platform. A machine learning-based prediction algorithm for workflow execution time was introduced in [27]. The algorithm was assessed within an internal OpenStack cloud consisting of eight servers [28]. Although using real environments may produce more trustworthy results, integrating new algorithms into existing systems can be technically challenging, especially when working with public clouds. There are also studies conducted purely analytically. A general issue with an analytical approach is that studies often concentrate on specific aspects, which may not fully capture the complex interactions within a heterogeneous computing ecosystem like modern clouds. As summarized in [29], cloud infrastructures have evolved from providing simple shared hardware resources (e.g., CPU time, storage space, and network bandwidth) to more sophisticated environments that include various featured services, platforms, and hardware components. To comprehensively study the overall effects resulting from the interactions of interleaving components within such ecosystems, an extensible and flexible cloud model is necessary. An agent-based cloud model offers maximum flexibility for adding, modifying, or removing functional/conceptual components. As Edge and Fog computing begin to gain popularity in the community [30–32], using an agent-based cloud-edge/fog model can better capture the distributed nature and geographic location of Edge/Fog devices. The following section focuses on a high-level design of the proposed cloud model.

Architecture

The cloud model is written in NetLogo-specific language and runs on the NetLogo platform [33]. The model consists of three main elements including *Service*, *Server*, and *Scheduler*, and they are modeled as agents. Operations of cloud elements are realized as agent interactions. For example, the deployment of services is modeled as moving services to servers. When a service is in the vicinity of its designated server, it will be captured by the server, and the server changes the service's status accordingly. Each type of agent has a set of attributes that reflect the characteristics and operations of its real-world counterpart. For example, a service agent has a set of resource requirements, a lifetime, its hosting server information, memory access ratio, and migration status. The conceptual architecture of the system is shown in Fig. 1. The detailed implementation of agents and functional components can be found at [7].

A simulation is started from establishing an environment that contains a set of servers and schedulers. Servers are conceptually grouped in racks. Each rack contains a dedicated scheduler and a set of servers. Servers are characterized by hardware models, which differ from resource capacity and energy consumption models. Each rack may contain different servers of various models that simulate a heterogeneous environment. All properties of servers will be initialized when created.

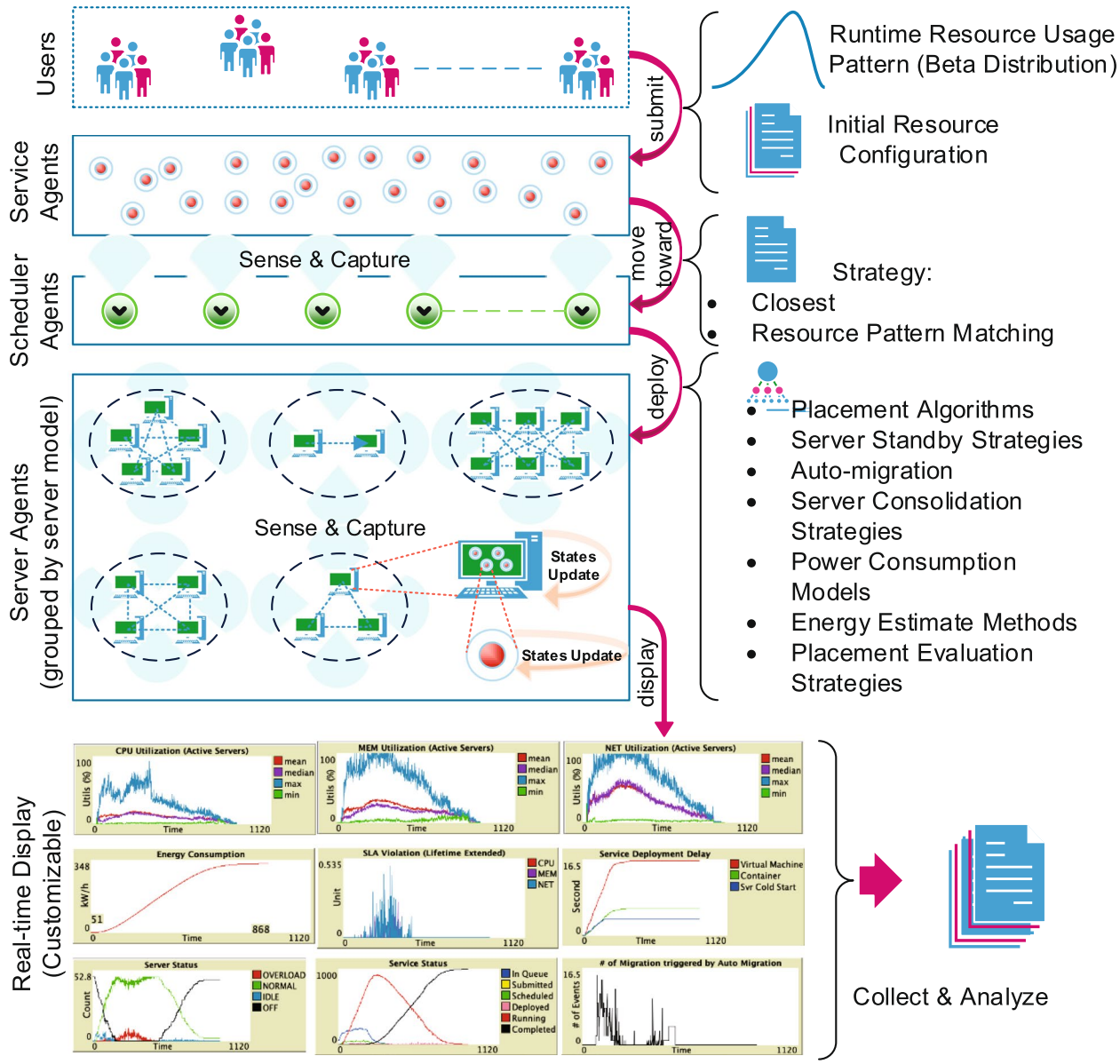

Fig. 1 The conceptual architecture of the agent-based cloud model

The corresponding scheduler of servers in the same rack is responsible for switching *on*/*idle*/*off* of servers, based on a selected server standby strategy. During a simulation, servers will update their status solely based on their current resource utilization level, as depicted in Fig. 2 (Server).

Once an environment is created, services carrying workloads will be generated in accordance with simulation plans. To reflect the dynamics of real-world cloud environments, services are randomly placed in a submission zone with a default random moving speed when created. This allows services to arrive at their designated schedulers at different times. The scheduler is responsible for managing and coordinating the placement of services and consolidation of servers. When services arrive at schedulers, they are scheduled to run on servers that meet resource requirements based on a selected algorithm for service placement (described in Resource optimization algorithms section). Thereafter, services move toward their designated servers with updated moving speeds. The new moving speeds are calculated partially based on services' deployment methods, which account for the delays associated with deployment initialization processes. Three deployment methods are currently

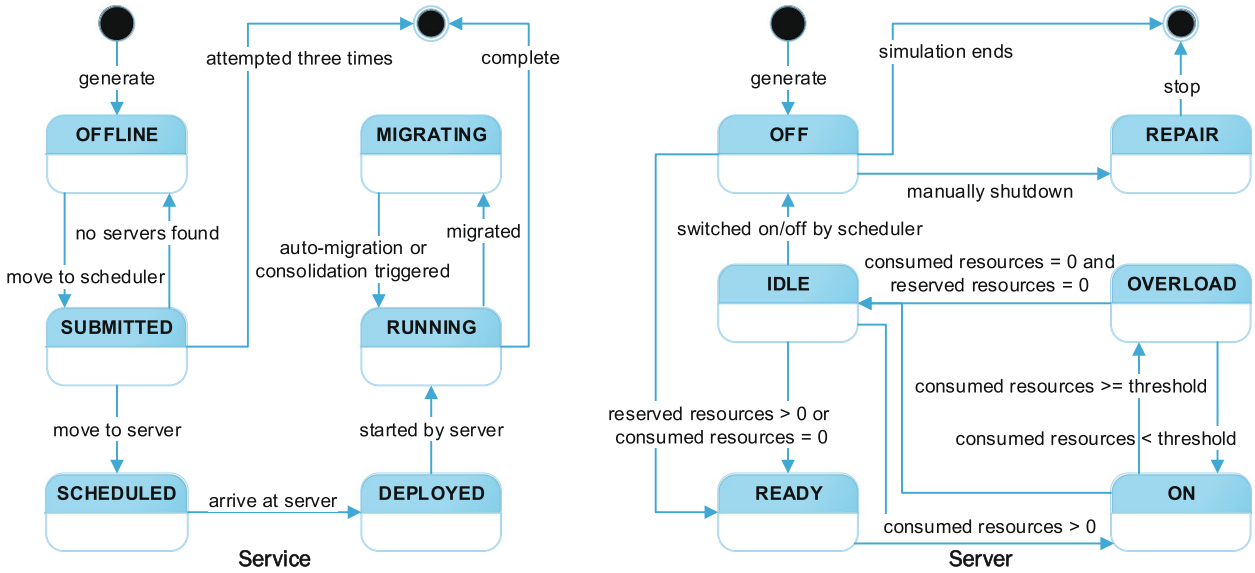

Fig. 2 State transition models of *Service* and *Server* Agent

supported: virtual machine, container, and bare-metal. Each has a fixed delay associated with it during deployment initialization processes.

Once a service arrives at its hosting server, it will have an "*RUNNING*" status that was set by the server itself. The state transitions are illustrated through the use of finite state machines for services, as depicted in Fig. 2 (Service). A service agent does not carry out any actual computation, but rather draws a portion of resources from its underlying server based on the resource utilization models assigned to it. The resource usage model for each service is characterized by tuning the α and β parameters in *beta* distributions, $R \sim Beta(\alpha, \beta)$. For example:

- Setting $\alpha = \beta = 1$, a uniform distribution of resource usage can be obtained;
- Setting $\alpha = 2$ and $\beta = 1$, a linear distribution can be achieved;
- Setting $\alpha = \beta = 2$, a normal distribution can be generated;
- Setting $\alpha = 2$ and $\beta = 3$, a right-tailed normal distribution can be obtained.

These distributions can simulate various types of service workloads as summarized in [29]. Furthermore, to maximize the flexibility, resource usage models can be adjusted for different kinds of resources on a per-service basis. For example, a communication-intensive service may have a normal distribution for CPU usage, a uniform distribution for memory usage, and a left-tailed distribution for network usage. Note that three kinds of resources are considered including computation power (measured in Server-side Java Operations per Second, *ssj_ops*), memory (measured in MB), and network bandwidth (measured in Mbps). In future work, resource usage models with seasonal effect will also be incorporated into the simulator to further enhance its versatility in modeling various service workloads.

When a service completes its tasks, i.e., reaches the end of its lifetime, it will be terminated and removed from its hosting server. In another case, if a service experiences performance degradation due to resource scarcity occurred on its underlying hosting server, the service's lifetime may be extended. The extension of a service's lifetime reflects the amount of violation of SLA. Calculation of the lifetime extension is detailed in Penalty for performance degradation section. An overview of the workflow and agent interactions is depicted in Fig. 3. Additionally, the simulator also provides several real time plots including accumulated resource usage, average resource usage, server status, energy consumption, number of migrating services triggered by auto-migration and server consolidation, accumulated lifetime extension (SLA violation), and service rejection rate when the system is overloaded. Figure 4 shows a screenshot of the user interface for the simulator. Other types of real time plots and parameter configuration widgets can be easily added, facilitated by the NetLogo platform.

Resource optimization algorithms

Service migration algorithms

There are two primary scenarios for service migration in cloud computing environments:

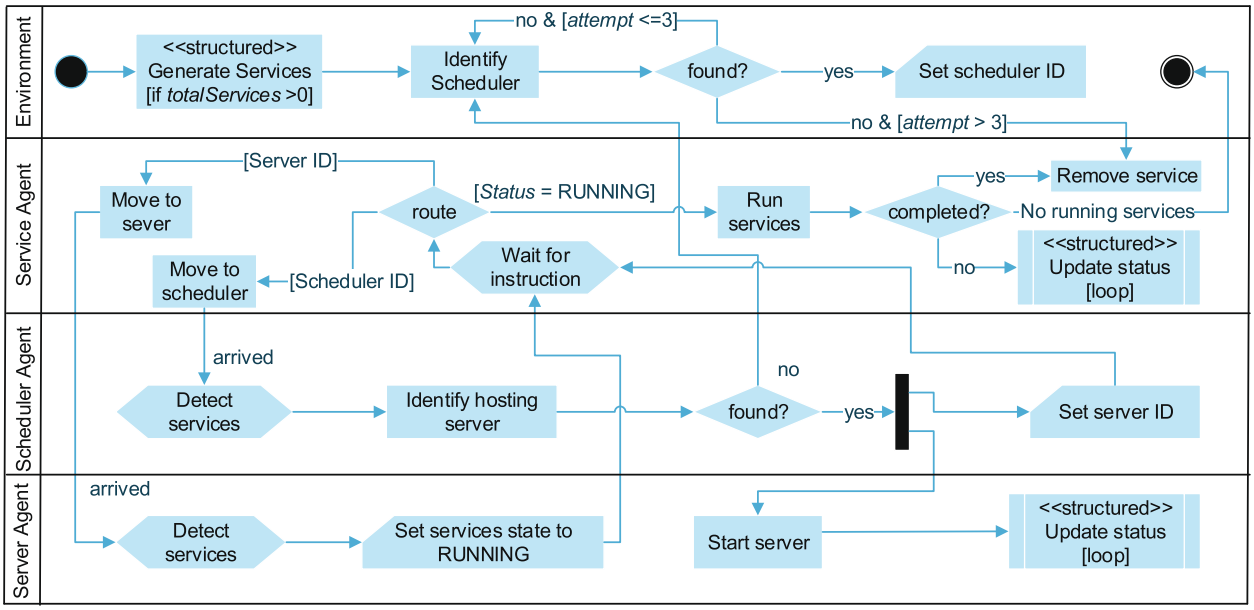

Fig. 3 Agents and system operating model

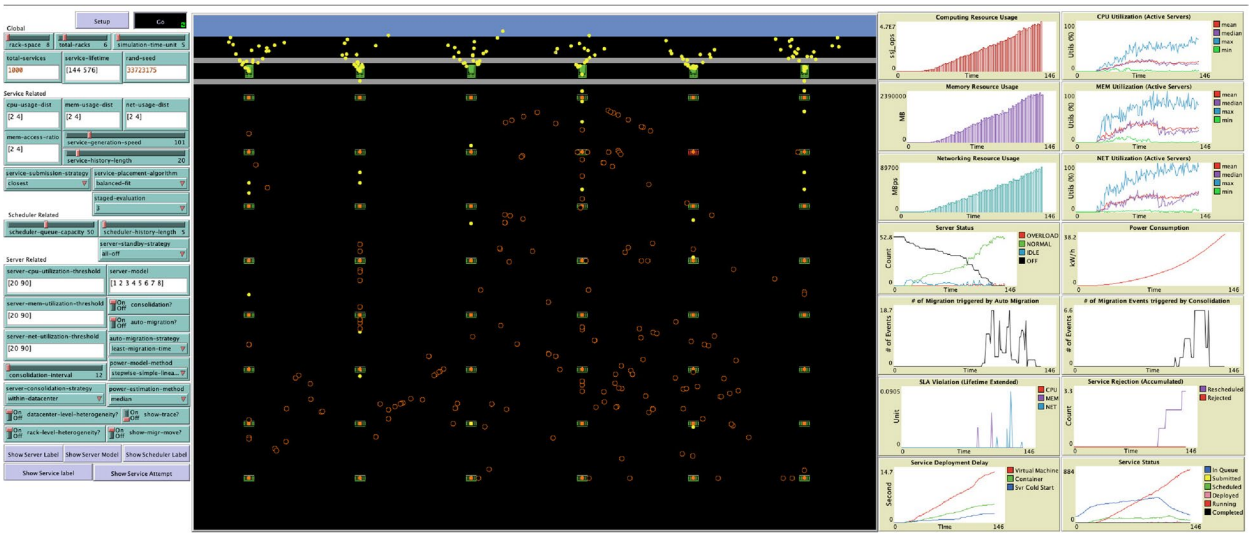

Fig. 4 User interface of the cloud simulator

- Under-utilization. When a hosting server experiences low resource usage due to falling below a specified under-utilization threshold. The practice is to move all running services to other more active servers. This process of relocating services from the less utilized host is referred to as server consolidation. Server consolidation is a recurring activity that optimizes resource allocation and improve overall system efficiency.
- Over-utilization. When a hosting server exhibits high resource usage due to surpassing a specified over-utilization threshold, some or all of the running services may need to be relocated to other less busy servers. In this situation, service migration is referred to as auto-migration. Auto-migration is a response mechanism that operates on an event-triggered basis. Auto-migration aims to minimize SLA violation.

The following sections discuss the algorithms and strategies implemented in the model in addressing both scenarios: server consolidation and auto-migration.

Server consolidation

The migration of services will lead to an overall increase in resource utilization while minimizing potential fragmentation within the cloud infrastructure. As a result, server consolidation efforts may be necessary. This process is typically performed periodically and involves either relocating all services from a particular hosting server or leaving them where they are. Current research in server consolidation or service placement has focused on designing algorithms for identifying suitable servers to host services based on various system-level objectives, such as energy minimization, resource utilization maximization, and scheduling efficiency. Examples of these approaches include the implementation of greedy algorithms, statistical & prediction-based methods, and meta-heuristics within a simulator environment. The implemented algorithms in the simulator are diverse: *random*, *first-fit*, *balanced-fit*, *max-utilization*, and energy minimization (*min-energy*). The first two algorithms are self-explanatory, while a *best-fit* algorithm has been implemented in two distinct flavors: *balanced-fit* and *max-utilization*. To facilitate understanding, the following notation conventions are used within this context.

Consider a list of services, denoted as $\mathbf{a_j} = \{a_{j0}, a_{j1}, ..., a_{jn}\}$, running on a server $s_j \in \mathbf{s}$, where $\mathbf{s}$ is a set containing servers s_0, s_1, ..., and s_m. Each service requesting specific types of resources at runtime denoted by an n-tuple $R_r(a_{ji})$. Similarly, each service has provisioned resources at deployment time represented as the configured resource tuple $R_c(a_{ji})$; $R_o(a_{ji})$ and $R_p(a_{ji})$ denote the resources that a_{ji} is currently occupied and previously occupied, respectively. For instance, consider a DNS service is to be deployed in the cloud. When deploying a new service to the cloud, users must estimate or specify the amounts of resources based on past experience or other factors, known as the provisioned resources. For example, the DNS service might be provisioned with an estimated configuration: R_c = {CPU: 5000ssj_ops, MEM: 4GB, NetBW: 100Mbps}. A placement algorithm will search for available hosting servers that can accommodate the service with the specified resources. Once deployed and running on a server s_j, the service will consume actual resource usage over time following a *beta* distribution per type of resource, at each simulation tick. For instance, let's consider two consecutive timestamps:

1. At time t_0: The current state is represented by the observed (or occupied) resource tuple $R_o(t_0)$ for the DNS service running on server s_j. This might be updated from its provisioned resources R_c to $R_o(t_0)$ = {CPU: 2354ssj_ops, MEM: 1.8GB, NetBW: 67Mbps} at this point in time.
2. At time t_1: The observed resource tuple for the service $R_o(t_1)$ might be updated to {CPU: 3054ssj_ops, MEM: 2.8GB, NetBW: 97Mbps}, and the previously occupied resources $R_o(t_0)$ becomes $R_p(t_1)$ at this point in time.

It is also important to emphasize the distinctions between $R_r(a_{ji})$ and $R_o(a_{ji})$ under the following circumstances.

1. When the total requested resources is not greater than the physical capacity of the underlying server, i.e., $\sum_{i=0}^{n} R_r(a_{ji}) \leq R_c(s_j)$, then $R_o(a_{ji}) = R_r(a_{ji})$.
2. In case where the total requested resources exceed the physical capacity of the underlying server, resulting in triggering of SLA violation events, then $R_o(a_{ji}) = R_r(a_{ji}) - \zeta(\sum_{i=0}^{n} R_r(a_{ji}) - R_c(s_j))$, where ζ is a distribution factor explained in Penalty for performance degradation section.

Using the same notation style, $R_o(s_j)$ and $R_c(s_j)$ represent the currently used resources and the physical capacity of s_j. $|R(\cdot)|$ denotes the number of types of resources considered. The main idea behind the *balanced-fit* algorithm is to maintain a similar resource utilization level across different kinds of resources. For example, $\frac{R_c(a_{ji})+R_o(s_j)}{R_c(s_j)}$ calculates resource utilization ratios of computing, memory, and network bandwidth of s_j given that if service a_i is placed on s_j. Calculating ratios of computing, memory, and network bandwidth of s_j is necessary because different kinds of resources might be measured in different units. The *balanced-fit* algorithm favors placing service a_i on server s_j with the minimum difference across all kinds of resources to avoid situations where some kinds of resources are heavily used but others are lightly used, leading to waste of resources or resource fragmentation. For instance, if a server's compute, memory, and networking resource utilization averages are {80%, 50%, 10%} respectively, the algorithm will try to place a computation-light, memory-neutral, and communication-intensive service on the server. The max-utilization algorithm tries to minimize the overall resource fragment of servers with an assumption that the resource requirements of services and resource configuration of servers are relatively balanced across different kinds. Pseudocode of the *balanced-fit* and *max-utilization* algorithms are shown in Algorithm 1.

function BEST-FIT($\mathbf{a}, \mathbf{s}$)
  min-dist $\leftarrow |R(\cdot)|$
  s $\leftarrow$ null
  for $s_k \in \mathbf{s}$ **do**
    if $R_c(a_{ki}) + R_o(s_k) < R_c(s_k)$ **then**
      d $\leftarrow \max\{\frac{R_c(a_{ki})+R_o(s_k)}{R_c(s_k)}\}$ - $\min\{\frac{R_c(a_{ki})+R_o(s_k)}{R_c(s_k)}\}$ ▷ if using the balanced-fit
      d $\leftarrow$ sum $(1 - \frac{R_c(a_{ki})+R_o(s_k)}{R_c(s_k)})$ ▷ if using the max-utilization
      if min-dist $>$ d **then**
        min-dist $\leftarrow$ d
        s $\leftarrow s_k$
      end if
    end if
  end for
  return s
end function

Algorithm 1 The *balanced-fit* algorithm focuses on maintaining balanced resource occupancy levels across various types of resources for a given server; meanwhile, the *max-utilization* approach is dedicated to minimizing the overall residual resources available on a server

An energy minimization algorithm is also implemented in a similar manner to the algorithms shown in Algorithm 1. A notable aspect is how the energy consumption of servers is modeled. In the simulator, eight types of recently manufactured servers are considered. The specifications of the servers were collected from spec.org[1] and details are listed in Table 6. Relationships between energy consumption, system workload, and CPU utilization are shown in Fig. 5. Observing that server performance varies greatly, and the energy consumption of servers does not align well with the system performance measured in *ssj_ops*, i.e., server-side Java workload in operations per second. For example, the system performance of RS700A-E9-RS4V2 increases much faster with a relatively slow increase in energy consumption. In comparison, the energy consumption and system performance of the Inspur NF8480M6 and ProLiant DL110 Gen10 Plus increase at a similar pace. Different models of servers were usually built with varying computational capacities, as shown in Fig. 7. Although some servers have a relatively higher baseline energy consumption, for example, the UniServer R4900 G5, but their Active Average Power increases sub-linearly with the increase of performance, which makes them suitable candidates for minimizing overall energy consumption. Generally, packing more services on a server will increase the level of the server's resource utilization, but it may not yield an optimal overall energy efficiency of the cloud. This apparently presents an opportunity for optimization when clouds consist of heterogeneous hardware.

Energy consumption plays a significant role in calculating cloud operational costs, with various models proposed to estimate energy usage. For instance, researchers have developed methods such as additive models, linear/non-linear regression, and polynomial models (e.g., [34]). In order to provide an unbiased evaluation, several modeling techniques were used to understand server energy consumption patterns. These include the Simple Linear Regression, Quadratic Polynomial, Cubic Polynomial, and Step-wise Linear Regression methods. The Step-wise method serves as a baseline for comparison since it closely aligns with the raw data. Among these models, the cubic polynomial model demonstrates the best fit for the energy consumption data but comes at an increased computational complexity. A detailed breakdown of model parameters and accuracy measured by R-squared values can be found in Table 6.

Auto-migration

In the second scenario, service migration becomes necessary when occupied resources approach the physical capacity limit of a server, which is defined by an over-utilization threshold. In such cases, some services must be moved to other servers so that the original server can maintain sufficient resources to prevent potential performance degradation caused by resource scarcity. This

[1] http://www.spec.org/power_ssj2008/results/

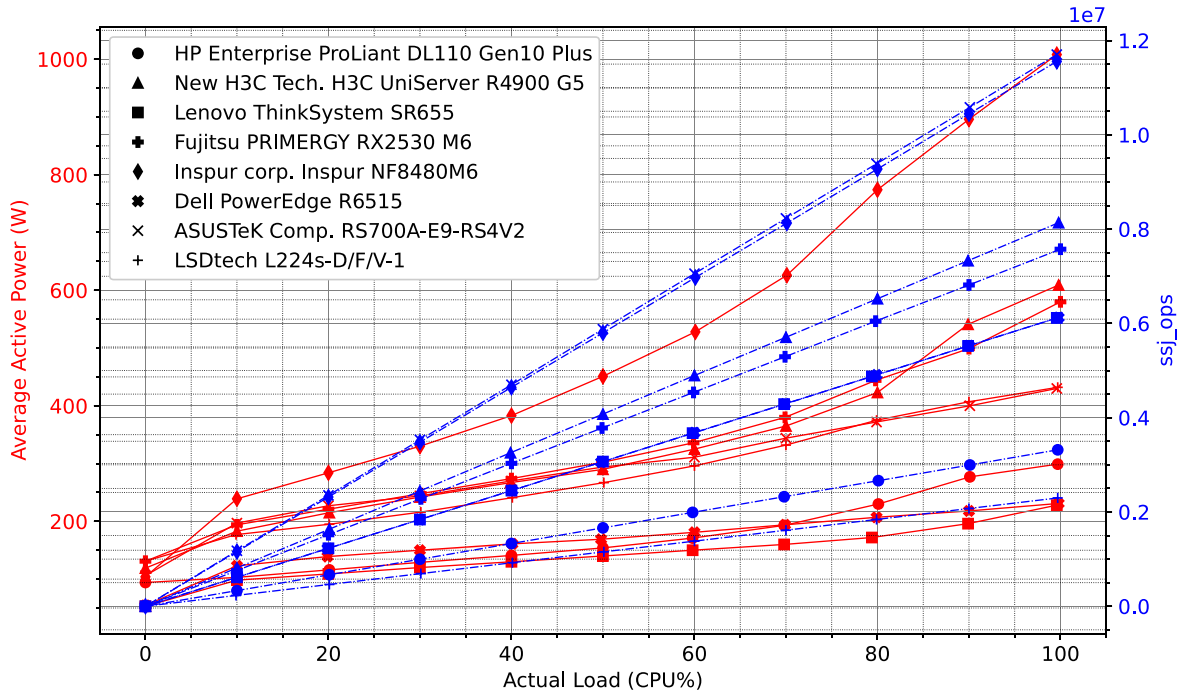

Fig. 5 Runtime energy consumption and system performance (measured in *ssj_ops*) under different CPU loads. The results show that energy consumption varies among servers, whereas system performance generally increases in a linear manner with increasing CPU loads

process is typically event-driven and involves migrating only selected services in order to keep the affected server running smoothly. The selection of these services should be strategic, based on system-level objectives. Two strategies have been implemented in order to optimize resource utilization and minimize disruption caused by service migration: *minimum number of migrations* (MNM) and *least migration time* (LMT). The primary goal of the MNM strategy is to reduce network congestion resulting from migrating *aggressive* services that consume a large portion of limited resources. An aggressive service, as defined in this context, refers to the one that consumes the most significant share of scarce resources. However, it's important to note that the impact on the network also depends on the memory footprint of potential migration candidates. Nevertheless, the simulator remains open to more advanced strategies. The LMT strategy focuses on minimizing service disruption caused by the migration process. Technically, estimating the delay involves considering factors such as available point-to-point network bandwidth between source and destination servers, memory footprint of candidate services, and memory dirtying rates [35]. After identifying potential candidate services, target server selection follows the same algorithms employed during the server consolidation process.

Penalty for performance degradation

When a hosting server cannot provide sufficient resources for its services, their performance may degrade. In such cases, all services hosted by that server will experience penalties in terms of extended lifetimes. If $\sum_{i=0}^{n} R_r(a_{ji}) > R_c(s_j)$, the resource allocation vector $\mathbf{a_j}$ receives penalties P_j, which are calculated based on three factors:

1. The total unsatisfied resources required by the services, $\sum_{i=0}^{n} R_r(a_{ji}) - R_c(s_j)$;
2. A distribution factor ζ that is applied across all services running on the same server;
3. A local factor specific to each individual service.

Denoting each simulation *tick*, representing the amounts of real-time elapsed, τ, in minutes, the total amount of penalty is quantified as $\tau \cdot \frac{\sum_{i=0}^{n} R_r(a_{ji}) - R_c(s_j)}{R_c(s_j)}$. Since services are very likely requesting different amounts of resources, either less or more, than their previously occupied, the distribution of penalty is therefore calculated based on the relative difference between currently requested resources and previously occupied resources. Let $\delta(a_{ji}) = R_r(a_{ji}) - R_p(a_{ji})$, and $\delta(\mathbf{a_j}) = \{\delta(a_{j0}), \delta(a_{j1}), ..., \delta(a_{jn})\}$, furthermore, let $\delta'(\mathbf{a_j})$ denotes a *scaled* $\delta(\mathbf{a_j})$, i.e., $\delta'(\mathbf{a_j}) = \{\delta'(a_{j0}), \delta'(a_{j1}), ..., \delta'(a_{jn})\}$, where $\delta'(a_{ji}) = \alpha + abs(min\{\delta(\mathbf{a_j})\}) + \delta(a_{ji})$, and α is a positive number. The

Table 1 Global configurations for the experiments

num of rack = 12; *server per rack* = 8; *simulation unit time* = 5 (min)
service generation speed = 300; *service lifetime* = [300, 300]
service cpu, memory, and network bandwidth runtime usage distribution: beta(2, 4)
memory dirtying rate: beta(2, 4); *server models* = [1 - 8]
server cpu, memoryt, and network bandwidth under-/over-utilization threshold = {20, 90}
consolidation interval = 12; *heterogeneity*: {data center, rack}

Note that the total number of servers can be determined jointly by the parameters "number of rack" and "servers per rack". Services are submitted at various points throughout the simulation, which can be partially adjusted using the parameter "service generation speed". The server code corresponds to the models of servers listed in Table 2

distribution factor for penalty allocation is expressed as $\zeta(a_{ji}) = \frac{\delta'(a_{ji})}{\sum_{i=0}^{n} \delta'(a_{ji})}$. The scaling process is necessary because the ratio cannot be zero or a negative value. This ensures that the total amount of penalty can fairly be distributed across all services running on s_j. It's important to note that some services might need fewer resources than their previously occupied, but regardless, those services will still receive penalties as resource scarcity occurs at the physical server level. This impact is captured by the distribution factor above, emphasizing that all services will be affected, albeit with varying degrees of influence due to the constraints imposed on the shared resources.

Moreover, a local factor is essential. When a service encounters resource scarcity, its impact can vary depending on the proportion of resources requested compared to the provisioned resources of the service. For instance, if a service requests substantial amounts of resources relative to both its previously occupied and provisioned resources, the effect will be relatively stronger; conversely, if the service requests fewer resources than its previously occupied, the impact shall be weaker. The local factor captures this variability. It is expressed as $(1 + \frac{\delta(a_{ji})}{R_c(a_{ji})})$. Note that $\delta(a_{ji})$ can also be a negative value. Finally, the extended lifetime for each service is presented in Eq. 1.

$$\mathbf{p_{ji}} = \frac{\tau \cdot \delta'(a_{ji}) \cdot (\sum_{i=0}^{n} R_r(a_{ji}) - R_c(a_{ji})) \cdot (R_c(a_{ji}) + \delta(a_{ji}))}{R_c(s_j) \cdot R_c(a_{ji}) \cdot \sum_{i=0}^{n} \delta'(a_{ji})} \quad (1)$$

Recall that $R(\cdot)$ represents an n-tuple, and the calculation is performed separately for each type of resources. In the simulation, the impact of resource scarcity accumulates across different types of resources. The formula can also be applied to other scenarios where quantifying SLA violations might be necessary.

Evaluation

The evaluation and analysis center around assessing the performance of algorithms and strategies employed within the simulator, with specific emphasis on energy efficiency, the effectiveness of penalty allocation, service migration, and server consolidation, as well as the accuracy of the servers' energy consumption model.

The global parameters for the configurations of the experiments are outlined in Table 1. These parameters govern various aspects of service deployment and management within a cloud environment. One of the key parameters is the "service generation speed", which determines how many services will be submitted to the cloud at any given time, ensuring that a specified number of services remain active in the service submission zone. Each new service has a uniformly drawn lifetime from the range specified by the "service lifetime" parameter. For consistency across experiments, all services were configured with the same length of lifetime, i.e., "service lifetime" [300, 300]. The resource usages of each service, such as compute, memory, and network bandwidth, are drawn from *beta* distributions. The "server utilization threshold" parameter specifies the under- and over-utilization thresholds for server consolidation and auto-migration, respectively. These thresholds help maintain optimal server performance while managing resource allocation in a cloud environment with dynamic service demands. Throughout Energy efficiency – Resource utilization sections, the Step-wise Linear Regression model was employed for calculating energy consumption. In order to study the behaviors of heterogeneous clouds, all server models were utilized in the relevant experiments. Specifications of the servers are listed in Table 2. The servers were mixed either within a rack or across the entire data center, as specified by the parameter "heterogeneity".

To ensure reproducibility and facilitate further investigation, random seeds were used. Each experiment was repeated 100 times with different random seeds. All experiments were conducted using NetLogo v6.3 on a Windows 10 Enterprise LTSC (64-bit) Dell OptiPlex 5090 workstation featuring an Intel Hexa-Core i5-1150 @2.70GHz processor.

Table 2 Server specifications

Code	Manufacturer and Model	Processor (GHz)	Memory	Release Date
1	HP ProLiant DL110 Gen10 Plus	Intel Xeon Gold 6314U @2.30GHz	64 GB	Aug-2021
2	Lenovo ThinkSystem SR655	AMD EPYC 7763 @2.45GHz	128 GB	Jun-2021
3	Fujitsu PRIMERGY RX2530M6	Intel Xeon Platinum 8380 @2.30GHz	256 GB	May-2021
4	New H3C Tech. UniServer R4900 G5	Intel Xeon Platinum 8380 @2.30GHz	256 GB	May-2021
5	Inspur Corp. NF8480M6	Intel Xeon Platinum 8380HL @2.90GHz	384 GB	Nov-2020
6	DellEMC PowerEdge R6515	AMD EPYC 7702P @2.00GHz	64 GB	Jul-2020
7	LSDtech L224S-D/F/V-1	Intel Xeon Gold 6136 @3.00GHz	196 GB	Jul-2020
8	ASUSTeK Inc. RS700A-E9-RS4V2	AMD EPYC 7742 @2.25GHz	256 GB	Feb-2020

Note that specifications of the servers were collected from *spec.org*, published in 2020 and 2021

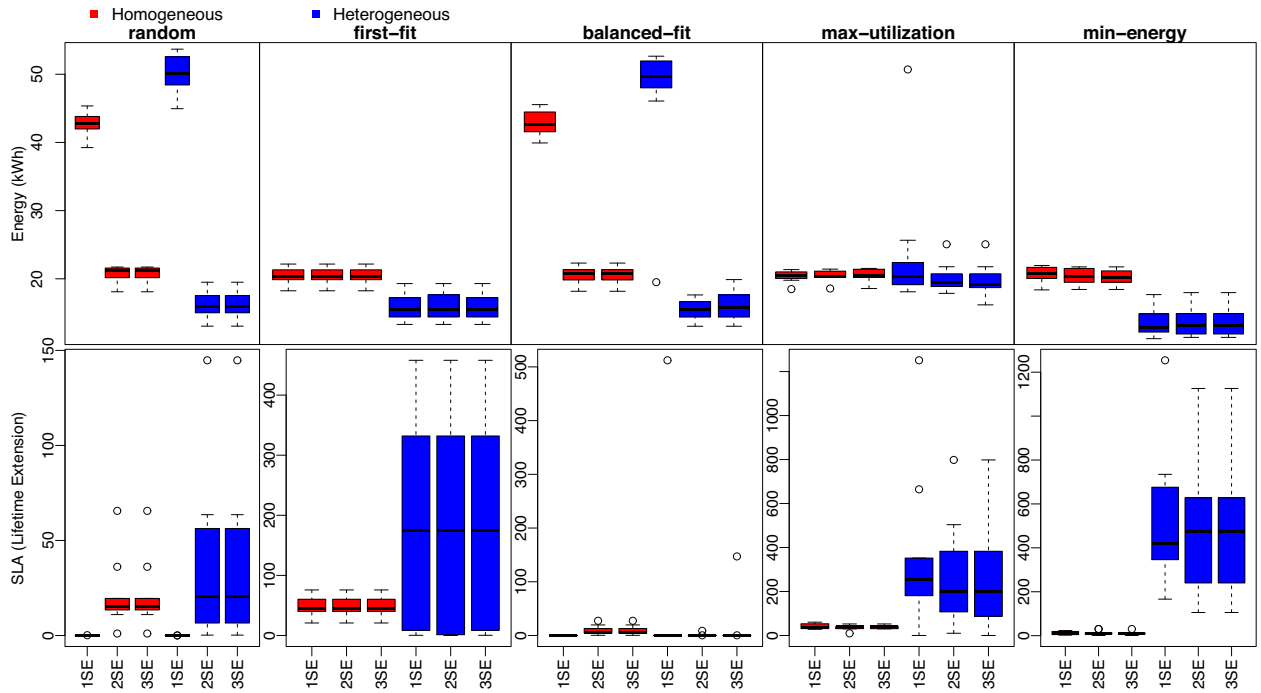

Fig. 6 A comparison of energy consumption and SLA violation between homogeneous and heterogeneous environments with auto-migration and server consolidation features disabled is presented. In heterogeneous clouds, the level of heterogeneity was set to rack level. SE indicates the staged evaluation strategy, as explained in Energy efficiency section. Energy usage is measured in units of kWh, while SLA violations are calculated according to the equation discussed in Penalty for performance degradation section

Energy efficiency

Energy consumption is a significant concern in cloud management. While higher resource utilization may not always lead to lower overall energy usage, Fig. 6 illustrates how different placement strategies impacted the performance of the cloud with respect to energy consumption and SLA violations. It becomes evident that when clouds contain heterogeneous hardware, there are more opportunities for optimization. However, it is crucial not to discuss algorithm effectiveness in isolation as their associated SLA violations can vary significantly from case to case.

Server status (*on*, *idle*, or *off*) and specifications (e.g., energy consumption patterns and resource capacity) are key factors in optimization efforts. The algorithms implemented within the simulator can be further categorized into three-staged evaluations, denoted as {1, 2, 3}SE. In the 1-staged evaluation (1SE), no consideration is given to server status when placing services; in the 2-staged evaluation (2SE), priority is given to placing services on servers that are either *on* or *idle*; finally, in the 3-staged evaluation (3SE), services are placed according to the order of $\{on\} \rightarrow \{idle\} \rightarrow \{off\}$.

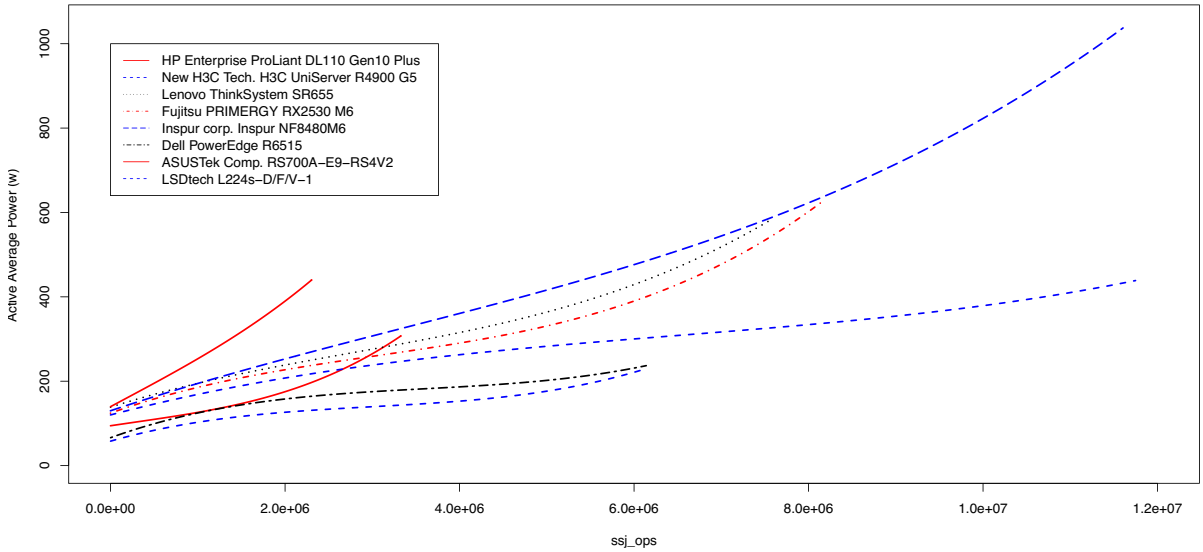

Fig. 7 Runtime energy consumption and system performance (*ssj_ops*) at various levels of CPU load

Table 3 Statistical mean differences observed between the staged evaluations of each algorithm, conducted independently within both homogeneous and heterogeneous cloud environments

Test Methods	Levene	Shapiro	ANOVA	Kruskal	Environment
random (rnd)	p>0.999	(W=0.992, ***p=0.326***)	***p>0.999***	p>0.999	Homogeneous, {1, 2, 3}SE
	p=0.961	(W=0.986, p=0.041)	p=0.997	p=0.976	Heterogeneous, {1, 2, 3}SE
first-fit (ff)	p=0.996	(W=0.988, p=0.013)	p=0.999	p>0.999	Homogeneous, {1, 2, 3}SE
	p=0.976	(W=0.955, p=4.9e-8)	p=0.989	p=0.996	Heterogeneous, {1, 2, 3}SE
balanced-fit (bf)	p=0.937	(W=0.980, p=6.6e-3)	p=0.864	p=0.851	Homogeneous, {1, 2, 3}SE
	p=0.804	(W=0.976, p=1.8e-3)	p=0.829	p=0.872	Heterogeneous, {1, 2, 3}SE
max-utilization (mu)	p=0.985	(W=0.987, p=6.9e-3)	p=0.889	p=0.924	Homogeneous, {1, 2, 3}SE
	p=0.767	(W=0.993, ***p=0.204***)	***p=0.611***	p=0.794	Heterogeneous, {1, 2, 3}SE
min-energy (me)	p=0.893	(W=0.978, p=1.4e-4)	p>0.863	p=0.794	Homogeneous, {1, 2, 3}SE
	p=0.986	(W=0.977, p=1.1e-4)	p=0.995	p=0.996	Heterogeneous, {1, 2, 3}SE

The performance difference between the algorithms were tested using either ANOVA (parametric) or Kruskal-Wallis (non-parametric) methods, depending on whether the Levene's test for homogeneity of variance and Shapiro-Wilk method for normality residuals passed ($p > 0.05$ indicates no violation). If both tests pass, ANOVA results are more trustworthy; otherwise, Kruskal-Wallis results prevail. In all cases, p-values were above the significance threshold (0.05) from both ANOVA and Kruskal-Wallis tests, accepting the *null* hypothesis and suggesting no significant differences between evaluation stages for each algorithm in both homogeneous and heterogeneous cloud environments

When using both the *random* and *balanced-fit* algorithms with the 1SE approach, there is a higher probability of placing services on servers that are currently in an {*off*} status. Due to the baseline electricity consumption associated with turning on such servers (as shown in Fig. 7), the two algorithms resulted in significantly increased energy consumption over time, as demonstrated in Fig. 6. Despite these outliers, mean differences for each individual algorithm configured with the set of {1, 2, 3}SE were found to be statistically insignificant when evaluated independently in both homogeneous and heterogeneous cloud environments, as shown in Table 3. However, there was a slight difference observed in energy consumption across algorithms assessed within homogeneous clouds, but statistically insignificant, as suggested by Kruskal tests {*2SE: p=0.129, *3SE: p=0.221}, where the * indicates all the algorithms configured with 2SE or 3SE, respectively. It is important to note that considering the energy consumption and SLA violations shown in Fig. 6, it becomes evident that the *balanced-fit*, *max-utilization*, and *min-energy* algorithms outperform other strategies such as *random* and *first-fit* approaches in homogeneous cloud environments. In contrast, when operating within heterogeneous cloud environments, the

Table 4 A pairwise comparison was conducted to evaluate the impact of implementing various features, including automatic migration (AM) and server consolidation (Co), on the statistical mean difference between algorithms for energy consumption and number of service migrations. The analysis considered both homogeneous and heterogeneous cloud environments

Algo.	AM , Co (Energy)		AM^+, Co (#Migration)			AM^+, Co^+ (#Migration)			
	He-2SE	He-3SE	Ho-{2,3}SE	He-2SE	He-3SE	Ho-2SE	Ho-3SE	He-2SE	He-3SE
ff - bf	0.247	0.303	-1.69	1.83	1.81	-1.57	-1.58	0.77	0.72
mu - bf	4.017	4.146	-1.53	0.51	0.50	-1.39	-1.37	-1.66	-1.45
me - bf	-2.621	-2.565	-0.85	0.50	0.49	-0.38	-0.39	-0.92	-0.80
rnd -bf	0.989	1.061	-0.67	2.02	2.01	-0.39	-0.37	0.25	0.23
mu - ff	3.769	3.843	0.16	-1.32	-1.31	0.18	0.21	-2.43	-2.17
me - ff	-2.868	-2.868	0.84	-1.33	-1.32	1.19	1.19	-1.69	-1.52
rnd - ff	0.741	0.757	1.02	0.19	0.20	1.18	1.21	-0.52	-0.49
me - mu	-6.638	-6.711	0.68	-0.016	-0.01	1.01	0.98	0.74	0.65
rnd - mu	-3.028	-3.086	0.86	1.51	1.51	1.00	1.00	1.91	1.68
rnd - me	3.609	3.626	0.18	1.52	1.52	-0.01	0.02	1.17	1.03

The differences between algorithms were calculated using Tukey Honest Significance Differences method, considering both homogeneous (Ho) and heterogeneous (He) cloud environments. The mean differences between 2SE and 3SE with/without auto-migration ($AM^{+/-}$) and server consolidation ($Co^{+/-}$) features were analyzed separately for each environment type. Since the mean differences in homogeneous clouds are identical when comparing 2SE with AM^+ to 3SE with AM^+, these values have been merged into a single column under "Ho-{2, 3}SE"

balanced-fit and possibly the *min-energy* algorithms are more suitable options for service placement.

The results in Fig. 6 show that less electricity was consumed when running algorithms in heterogeneous clouds compared with homogeneous ones, which is visually observable. Additionally, it becomes more evident, {*2SE: Kruskal($p \ll 0.001$), *3SE: Kruskal($p \ll 0.001$)}, how the algorithms behave differently from each other in these environments. In particular, multiple pairwise-comparison tests were conducted and the results are shown in Column AM^-, Co^- (Energy) of Table 4. The *min-energy* algorithm saved an average of 2.621 electricity units with 2SE and 2.565 units with 3SE compared to the *balanced-fit* algorithm, which resulted in a significant number of SLA violations on average: 475.20 for 2SE and 479.36 for 3SE. It seems that when only auto-migration was enabled, the *min-energy* migrated services more frequently compared to the *balanced-fit* algorithm, which might indicate an unbalanced use of resources. Overall, the *balanced-fit* algorithm appeared to be the most stable one among all algorithms considered in this study.

In this section, two sets of comparisons were conducted to provide insights into the performance of individual algorithms in staged evaluations and their overall effectiveness across different environments. The first set of comparisons focused on comparing the stability of each algorithm configured with various staged evaluations. The results showed that both 2SE and 3SE are generally stable across all algorithms, indicating a consistent performance throughout the evaluation process. Based on these findings, the second set of comparisons aimed to identify *stable* algorithms in terms of energy consumption and SLA violation for homogeneous and heterogeneous environments. This analysis helps to pinpoint favorable algorithm configurations that can effectively balance between energy efficiency and service level agreement compliance. Moreover, it is possible to further optimize the performance of these stable algorithms by implementing dynamic switching mechanisms that adjust their behavior based on specific SLA thresholds. Such adaptive strategies can help maintain a delicate balance between energy consumption and SLA violation in various cloud environments.

Although real time auto-scaling can improve overall resource utilization, it frequently leads to service level agreement violation, which may negatively impact service performance. To address this issue proactively, some services should be migrated from overcrowded servers to less busy ones. Figure 8 shows the probability density functions of energy consumption & SLA violations, and the cumulative distribution functions of service migrations collected from simulations of both homogeneous and heterogeneous cloud environments that revealed interesting insights into energy consumption, SLA violations, and the effectiveness of different algorithms for managing resources. In a homogeneous environment, two specific algorithms - *balanced-fit* and *min-energy* - resulted in lower energy consumption levels while maintaining better control over SLA violations. The majority of these violations were concentrated within the range of 3 to 5 units, indicating that these algorithms were able to effectively minimizing energy consumption and control SLA violations. When comparing the number of service migrations between

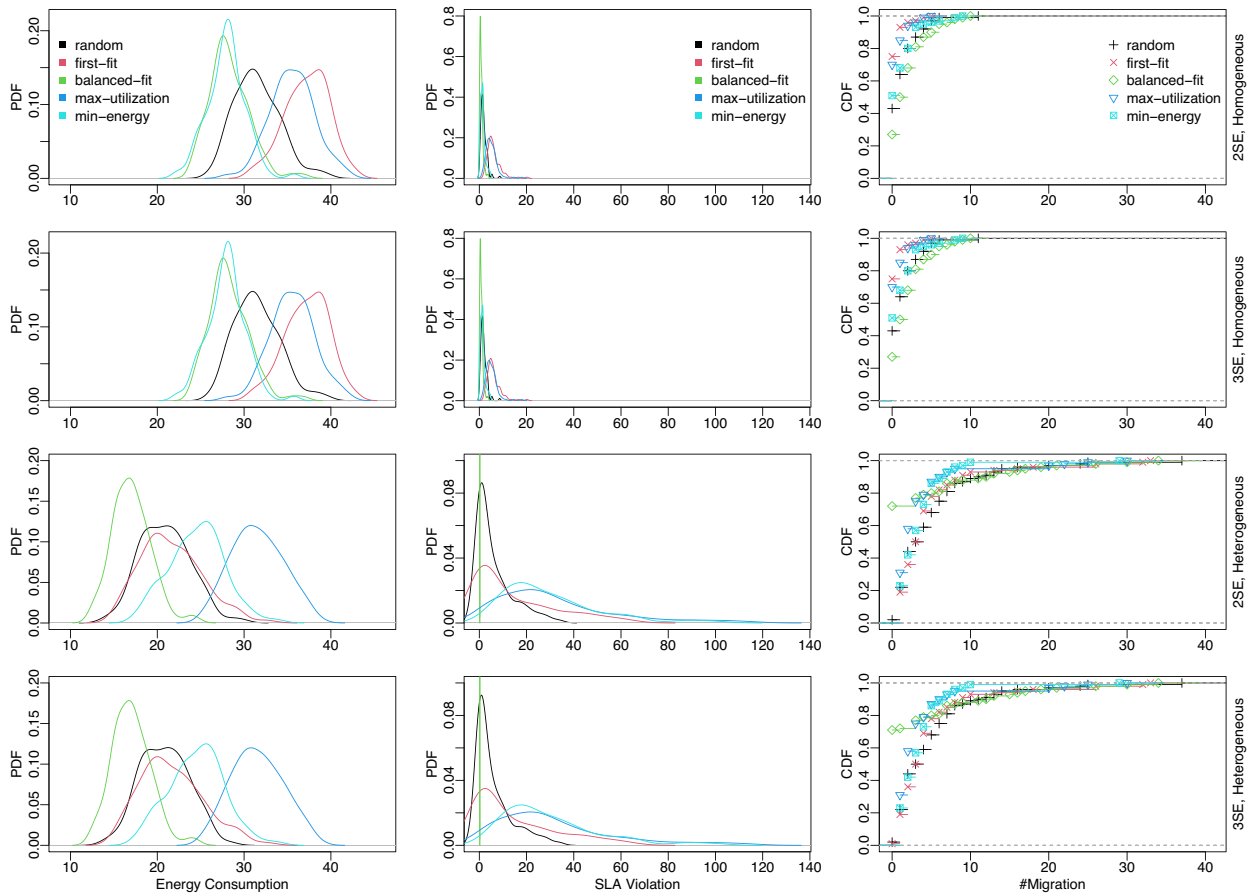

Fig. 8 The energy consumption, SLA violation, and number of service migrations obtained from homogeneous and heterogeneous clouds with auto-migration enabled

Table 5 Statistical mean differences in energy consumption and the number of service migrations obtained through simulations of different cloud configurations

Configuration	balanced-fit (energy diff in unit)		min-energy (energy diff in unit)	
	2SE	3SE	2SE	3SE
(Ho,MA ,Co) - (Ho,MA$^+$,Co)	(-8.418, -7.395)	(-8.448, -7.424)	(-8.189, -7.196)	(-8.193, -7.200)
(He,MA ,Co) - (He,MA$^+$,Co)	(-0.909, 0.286)	(-0.969, 0.211)	(-11.187, -9.727)	(-11.192, -9.735)
(Ho,MA$^+$,Co) - (Ho,MA$^+$,Co$^+$)	(5.367, 6.549)	(5.374, 6.556)	(5.450, 6.587)	(5.450, 6.587)
(He,MA$^+$,Co) - (He,MA$^+$,Co$^+$)	(0.586, 1.776)	(0.590, 1.782)	(5.730, 7.214)	(5.693, 7.168)

Note that 95% confidence interval was used

Ho homogeneous, *He* heterogeneous, $AM^{+/-}$ with/without auto-migration, $Co^{+/-}$ with/without server consolidation

different algorithms in the experiments, it was found that there was only a marginal difference between them (1-2), as shown in Table 4. The primary distinction lies in the level of SLA violations compared to the results shown in Fig. 6. This difference translates into trading off from higher energy consumption levels, as demonstrated in the first row of Table 5. It is also important to note that both the *balanced-fit* and *min-energy* algorithms consumed less energy when utilizing a server consolidation mechanism. For example, the mean differences in energy consumption between the configurations [(Ho,MA$^+$,Co$^-$) - (Ho,MA$^+$,Co$^+$)] and

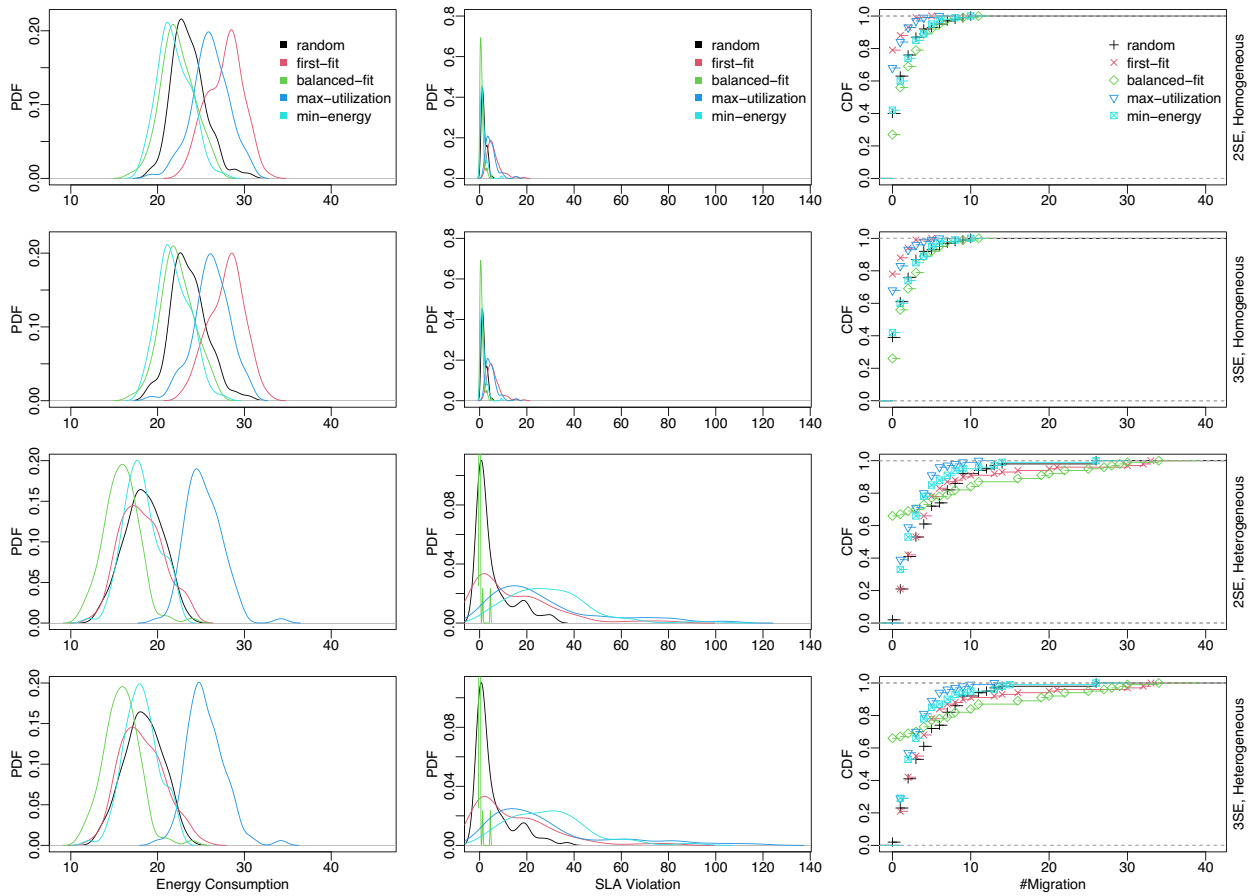

Fig. 9 Energy consumption, SLA violation, and number of service migrations obtained from experiments involving both homogeneous and heterogeneous cloud environments with auto-migration and server consolidation features activated

[(He,MA$^+$,Co$^-$) - (He,MA$^+$,Co$^+$)] fall in the range (95% CI = 5.367, 6.549) and (95% CI = 0.586, 1.776), respectively. These results demonstrate the effectiveness of server consolidation in reducing energy consumption. Overall, while all algorithms in this study showed some ability to manage resource allocation and reduce energy consumption, the *balanced-fit* approach emerged as particularly effective strategies for maintaining good performance in cloud computing environments.

Effectiveness of service migration

In contrast to other algorithms, the *balanced-fit* approach demonstrated superior performance in terms of energy consumption and SLA violation (near zero) within heterogeneous environments. However, this algorithm resulted in a slightly higher number of service migrations compared with others operating under similar conditions, with the largest mean difference of 2.02, as shown in Table 4. When cross-compared to the results obtained from Fig. 6 (heterogeneous environments), it becomes evident that energy efficiency was slightly improved overall (second row of Table 5). With a 90% probability, the number of service migrations under this algorithm falls within a range between 0 and 10. Overall, the *balanced-fit* approach emerged as the most stable and efficient algorithm in both homogeneous and heterogeneous environments with the specific configurations mentioned in Table 1.

To enhance energy efficiency and resource utilization even further, a server consolidation feature was incorporated into the experiments. It's important to note that auto-migration and server consolidation are two separate processes. Auto-migrations occur when services experiencing resource scarcity, while server consolidation runs periodically (12 ticks for the following experiments), simulating a consolidation process triggered on a per hour basis in real-world scenarios. As shown in Fig. 9, server consolidation can significantly reduce energy consumption compared to the results displayed in Fig. 8 (same set of services, but without server consolidation). The differences were further analyzed using an independent *t*-test with a 95% confidence interval, as presented in Table 5.

In these experiments, both the *balanced-fit* and the *min-energy* algorithms demonstrated strong performance. However, it's worth noting that while the *min-energy* algorithm had a lower average energy consumption, its variance of SLA violation was much wider (unstable). On the other hand, the *balanced-fit* algorithm maintained a more stable and efficient performance with a smaller variance concentrated around 2-3 units and number of migrations under 10 at 90% of the time.

Resource utilization

Comparing resource utilization between multiple instances of simulation becomes complex due to various random factors involved in each case, such as differences in service submission times, arrival speeds, and per-service configurations; server heterogeneity enabled/disabled based on configuration settings; varying patterns of resource utilization over time; and the impacts of resource scarcity on service lifetimes. At different timestamps, the number of services, their distribution across servers, and overall resource utilization may vary significantly from one simulation run to another. To draw meaningful conclusions about resource utilization in these experiments, this work focused on calculating such values for only *active* servers, those with running services, and then bin-packing the results based on the number of active servers. Statistics were collected from 100 independent runs of the experiments to provide a comprehensive understanding of resource utilization in both homogeneous and heterogeneous cloud environments.

From Fig. 10, it becomes evident that all algorithms exhibit similar performance in homogeneous cloud environments. In contrast, as illustrated by Fig. 11, the *first-fit* and *balanced-fit* algorithms demonstrate superiority over other approaches when considering server utilization in heterogeneous clouds. It is worth noting that the *random* algorithm appears to use fewer servers than its counterparts due to the influence of the 3-staged evaluation process. This is because, during the initial stages, active servers are given priority and thus tend to be placed on a smaller number of servers.

Effectiveness of the penalty allocation

In order to evaluate the proposed SLA quantification and allocation scheme, a controlled experimental environment was established. For this experiment, ten services were configured with memory-intensive workloads and deployed on a single server (with model code 1 and 64GB of memory). The provisioned memory resources for each service were uniformly drawn from the set {8192MB, 12288MB, 16384MB}. The runtime memory usage of each service was still drawn from the *beta* distribution with parameters $\{\alpha = 2, \beta = 4\}$. The requirements for computing and networking resources were intentionally kept at minimum to ensure that no SLA violations would be triggered by resource constraints of these kinds. All services had a fixed lifetime of 500 units. Other parameters remained consistent with those shown in Table 1.

Figure 12 illustrates the memory usage of both the server and the individual services during runtime. The line in black represents the aggregate memory usage of all ten services, which is equivalent to the memory usage of the server itself. It can be seen that the server's memory capacity was not exceeded by any of the service's memory requests. However, if a service were to request more memory than what the server could provide, an SLA violation would occur and be indicated by the red dashed line in the graph. Although the services had been configured with a fixed lifetime of 500 units, the timeline on the x-axis has reached around 660 due to three possible reasons:

1. The ten services were submitted at different times, which caused the overall duration of the experiment to be extended;
2. Because of the SLA violations, the lifetime of each service was automatically extended accordingly in order to maintain compliance with the specified SLA;
3. Constrained by the limited physical resources, some services may not be able to deploy immediately. They may have to wait for sufficient resources to become available on the server.

In relation to the unfulfilled memory resource requests above the server's physical capacity (the upper part of Fig. 12, where the y-axis is greater than 64GB), Fig. 13 illustrates the penalties calculated for all services running on the server. It becomes evident that the total amount of SLA violations corresponds proportionally to the amounts of unfulfilled memory resources, as shown in Fig. 12.

To further explore the intricate of the distribution of penalties, an arbitrary instance was selected from Fig. 13, highlighted in blue. Additional details can be found in Fig. 14. At the time of execution, ten services were concurrently running on the server, with their initially provisioned memory resources shown in Fig. 14 [left, '*provisioned*']. Certainly, the total amounts of provisioned memory (for services, 112GB) significantly exceed the physical capacity of the server (64GB), causing some services to be delayed upon deployment. However, during runtime, services may require much less resources than their provisioned amount, depending on their workloads and resource utilization patterns. The unused resources can thus be utilized for deploying new services, i.e., *over-commitment*. However, at the previous execution

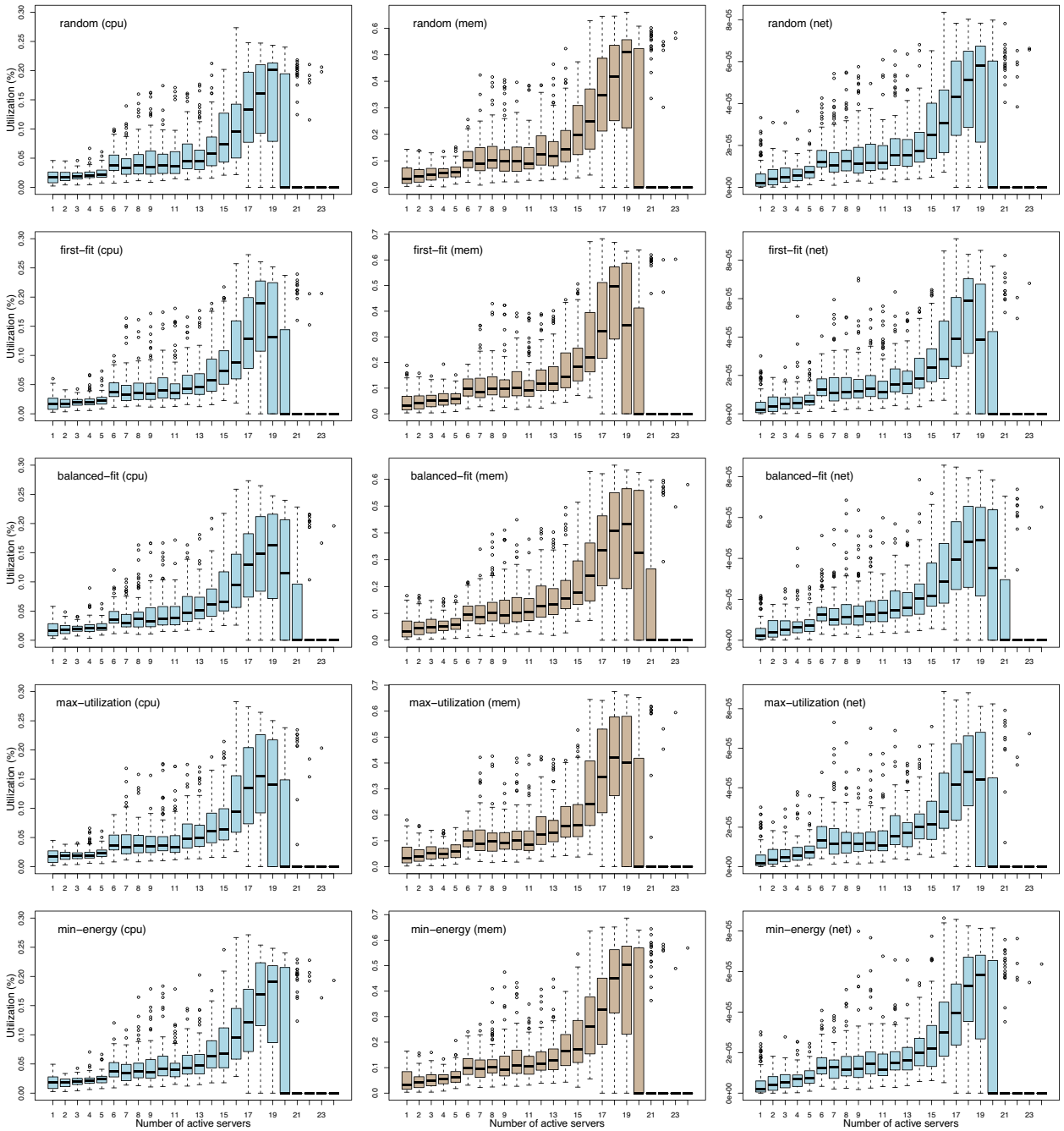

Fig. 10 Resource utilization of active servers was examined in experiments involving homogeneous clouds with a 3-staged evaluation process. For these experiments, auto-migration and server consolidation were disabled to better understand resource allocation patterns without the influence of these features. In the experiments, services represent predominantly memory-intensive tasks

step (as shown in Fig. 14 [left, '*previously occupied*']), the total amount of memory occupied by the services was 55.8GB, which is well below the server's physical capacity of 64GB. Consequently, no SLA violation events occurred during that time. When the execution advanced to the current step, the total requested memory resources increased to 84.76GB, surpassing the server's physical capacity limit. As a result, all services received penalties based on their share of unfulfilled memory resource requests (as shown in Fig. 14 [right]). Although service-0 (a_0) had requested less resources than its previously occupied state, it still faced some penalty due to the

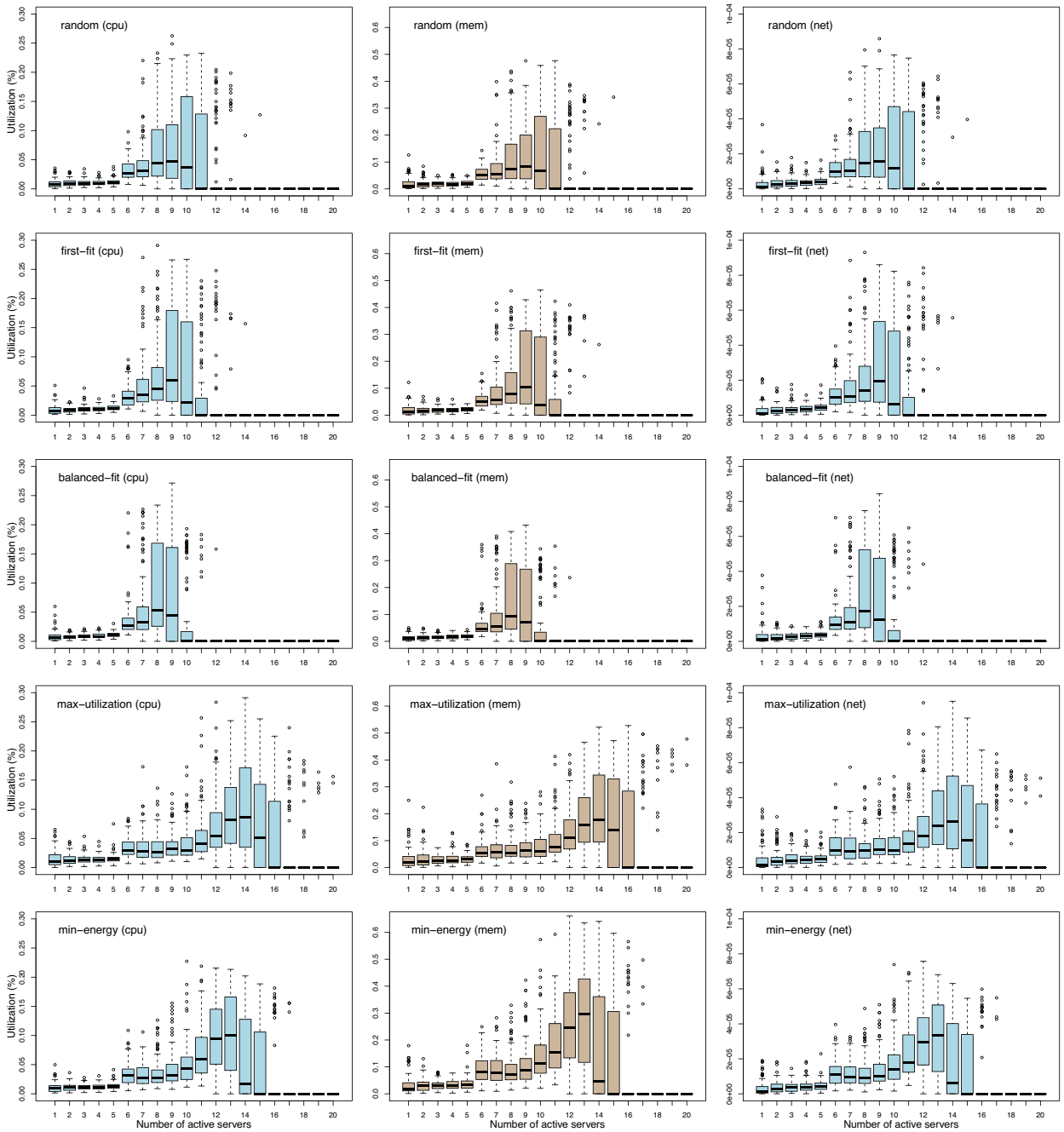

Fig. 11 Resource utilization of active servers was examined in experiments involving heterogeneous clouds with a 3-staged evaluation process. For these experiments, auto-migration and server consolidation were disabled to better understand resource allocation patterns without the influence of these features. In the experiments, services represent predominantly memory-intensive tasks

server's overall performance degradation. On the other hand, services with higher memory requirements, such as service-3, 6, and 8, received more significant penalties since they needed larger amounts of memory to complete their tasks, but could not be fully satisfied due to the server's limited resources. These services experienced a greater impact from the SLA violation events.

To better explain the concept, a concrete example with detailed calculation process is outlined below. As mentioned in Penalty for performance degradation section,

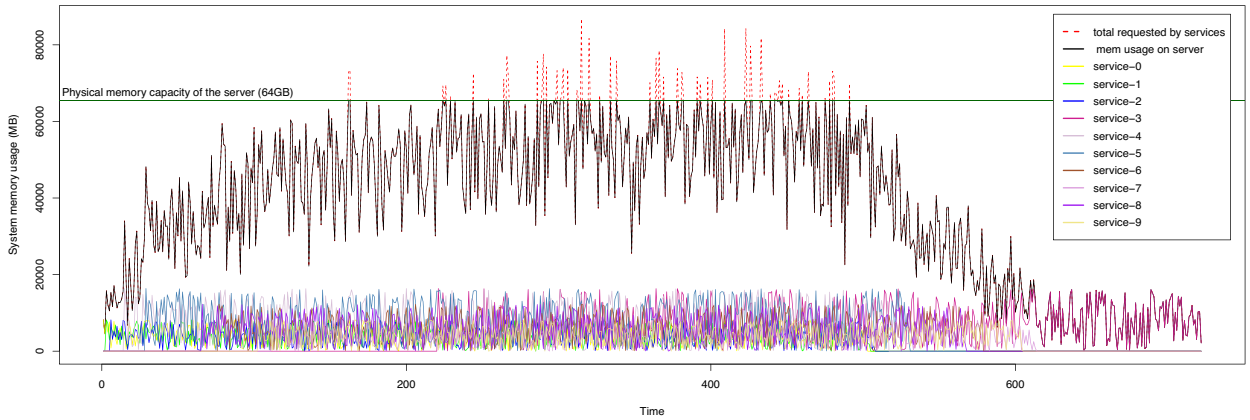

Fig. 12 During runtime, each service made memory resource requests to the system. Ten services were deployed on a single server (with model code 1) that was equipped with 64GB of physical memory. In cases where memory requests exceeded the available physical capacity, SLA violations would occur and be indicated by the red dashed line in the graph. Note that 1GB = 1024MB

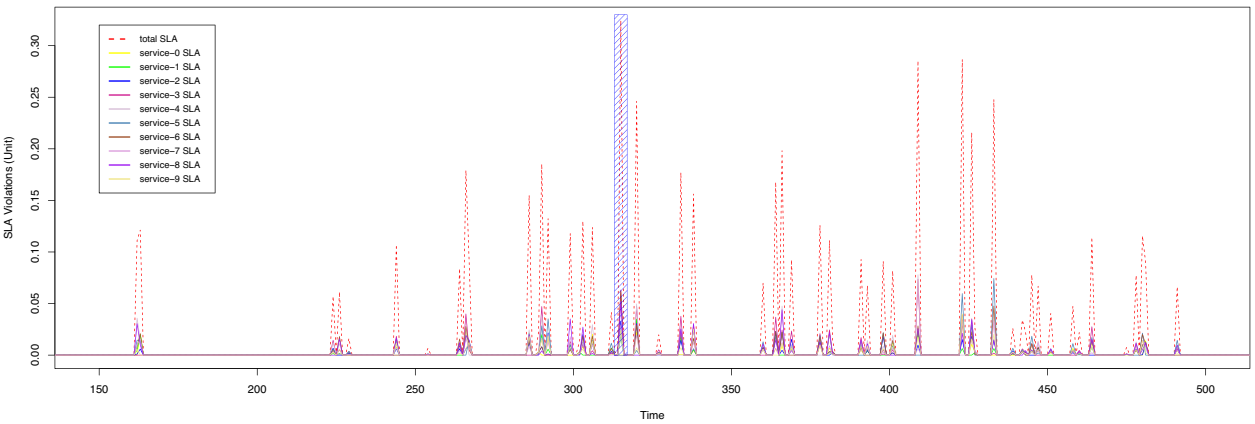

Fig. 13 The extension of services' lifetime was based on the amounts of memory resources that could not be provided by the underlying server

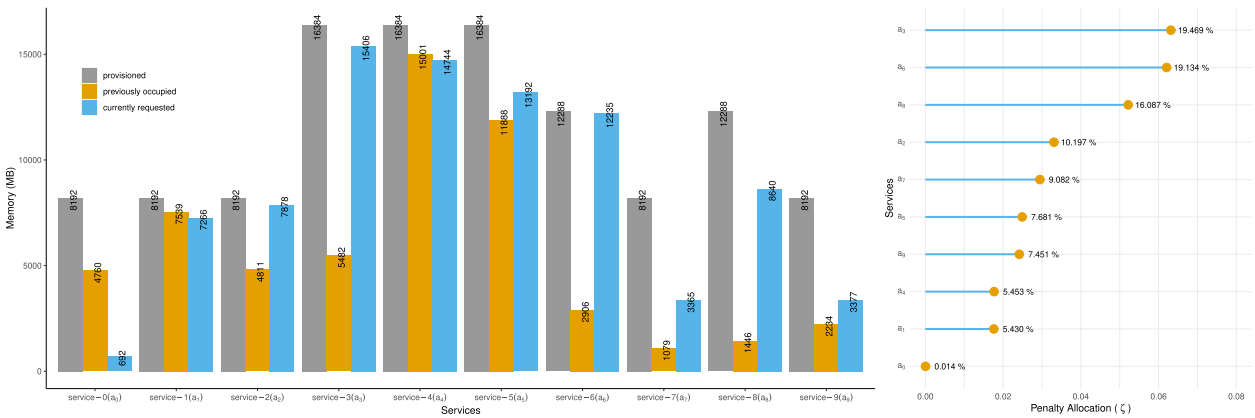

Fig. 14 This figure illustrates one occurrence of penalty distribution (the shaded area in Fig. 13) among all services running on the server. The distribution of penalties was calculated according to the amounts of provisioned resources, previously occupied resources, and currently requested resources for each service, as specified by Eq. (1)

Table 6 Server Side Java workload models and the model accuracy

Code	Method	Model	Accuracy
1	Simple Linear	Power = 6.076e-05 * SSJ + 7.237e+01	$R^2 = 0.933$
	Quadratic	Power = 1.642e-11 * SSJ^2 + 6.182e-06 * SSJ + 9.959e+01	$R^2 = 0.992$
	Cubic	Power = 3.90e-18 * SSJ^3 - 3.01e-12 * SSJ^2 + 3.08e-5 * SSJ + 9.45e+1	$R^2 = 0.995$
	Stepwise Linear	{(2.7e-5, 9.4e+1), (3.9e-5, 9.2e+1), (4.0e-5, 8.0e+1), (3.6e-5, 9.3e+1), (3.9e-5, 9.0e+1),	
	CPU% 0 – 100	(5.2e-5, 6.8e+1), (6.6e-5, 3.9e+1), (1.1e-4, -5.9e+1), (1.4e-4, -1.5e+2), (6.8e-5, 7.2e+1)}	
2	Simple Linear	Power = 2.313e-05 * SSJ + 7.066e+01	$R^2 = 0.950$
	Quadratic	Power = 3.178e-13 * SSJ^2 + 2.118e-05 * SSJ + 7.245e+01	$R^2 = 0.950$
	Cubic	Power = 1.76e-18 * SSJ^3 - 1.58e-11 * SSJ^2 + 5.88e-5 * SSJ + 5.80e+1	$R^2 = 0.994$
	Stepwise Linear	{(7.4e-5, 5.3e+1), (1.8e-5, 8.7e+1), (1.8e-5, 8.7e+1), (1.6e-5, 9.0e+1) (1.6e-5, 9.0e+1),	
	CPU% 0 – 100	(1.7e-5, 8.9e+1), (1.6e-5, 9.0e+1), (2.0e-5, 7.2e+1), (3.7e-5, -9.3e+0), (5.4e-5, -1.0e+2)}	
3	Simple Linear	Power = 5.361e-05 * SSJ + 1.252e+02	$R^2 = 0.967$
	Quadratic	Power = 3.803e-12 * SSJ^2 + 2.481e-05 * SSJ + 1.580e+02	$R^2 = 0.988$
	Cubic	Power = 1.24e-18 * SSJ^3 - 1.03e-11 * SSJ^2 + 6.56e-5 * SSJ + 1.39e+2	$R^2 = 0.998$
	Stepwise Linear	{(8.3e-5, 1.3e+2), (3.6e-5, 1.7e+2), (3.5e-5, 1.7e+2), (3.5e-5, 1.7e+2), (3.7e-5, 1.6e+2),	
	CPU% 0 – 100	(4.5e-5, 1.3e+2), (5.8e-5, 7.2e+1), (8.5e-5, -6.9e+1), (7.2e-5, 1.1e+1), (1.1e-4, -2.3e+2)}	
4	Simple Linear	Power = 5.369e-05 * SSJ + 1.067e+02	$R^2 = 0.935$
	Quadratic	Power = 4.869e-12 * SSJ^2 + 1.405e-05 * SSJ + 1.550e+02	$R^2 = 0.975$
	Cubic	Power = 1.58e-18 * SSJ^3 - 1.45e-11 * SSJ^2 + 7.40e-5 * SSJ + 1.24e+2	$R^2 = 0.995$
	Stepwise Linear	{(7.9e-5, 1.2e+2), (3.9e-5, 1.5e+2), (3.3e-5, 1.6e+2), (3.1e-5, 1.7e+2), (2.8e-5, 1.8e+2),	
	CPU% 0 – 100	(4.3e-5, 1.2e+2), (4.9e-5, 8.4e+1), (7.1e-5, -4.0e+1), (1.5e-4, -5.3e+2), (8.4e-5, -7.7e+1)}	
5	Simple Linear	Power = 7.339e-05 * SSJ + 8.569e+01	$R^2 = 0.967$
	Quadratic	Power = 3.526e-12 * SSJ^2 + 3.258e-05 * SSJ + 1.565e+02	$R^2 = 0.991$
	Cubic	Power = 4.75e-19 * SSJ^3 - 4.71e-12 * SSJ^2 + 6.89e-5 * SSJ + 1.3e+2	$R^2 = 0.995$
	Stepwise Linear	{(1.2e-4, 1.0e+2), (3.9e-5, 1.9e+2), (4.0e-5, 1.9e+2), (4.6e-5, 1.7e+2), (5.9e-5, 1.1e+2),	
	CPU% 0 – 100	(6.6e-5, 7.1+1), (8.5e-5, -6.4e+1), (1.3e-4, -4.2e+2), (1.0e-4, -1.9e+2), (1.0e-4, -1.6e+2)}	
6	Simple Linear	Power = 2.335e-05 * SSJ + 9.456e+01	$R^2 = 0.905$
	Quadratic	Power = -2.637e-12 * SSJ^2 + 3.951e-05 * SSJ + 7.969e+01	$R^2 = 0.938$
	Cubic	Power = 1.64e-18 * SSJ^3 - 1.77e-11 * SSJ^2 + 7.48e05 * SSJ + 6.61e+1	$R^2 = 0.974$
	Stepwise Linear	{(1.1e-4, 5.5e+1), (2.6e-5, 1.1e+2), (1.8e-5, 1.2e+2), 1.8e-5, 1.2e+2), (1.3e-5, 1.3e+2),	
	CPU% 0 – 100	(1.9e-5, 1.1e+2), (2.1e-5, 1.0e+2), (2.1e-5, 1.0e+2), (1.8e-5, 1.2e+2), (2.2e-5, 9.8e+1)}	
7	Simple Linear	Power = 1.285e-04 * SSJ + 1.310e+02	$R^2 = 0.991$
	Quadratic	Power = 1.344e-11 * SSJ^2 + 9.763e-05 * SSJ + 1.416e+02	$R^2 = 0.995$
	Cubic	Power = 4.37e-18 * SSJ^3 - 1.61e-12 * SSJ^2 + 1.11e-4 * SSJ + 1.40e+2	$R^2 = 0.995$
	Stepwise Linear	{(2.0e-4, 1.3e+2), (7.7e-5, 1.6e+2), (9.1e-5, 1.5e+2), (1.1e-4, 1.4e+2), (1.1e-4, 1.4e+2),	
	CPU% 0 – 100	(1.3e-4, 1.2e+2), (1.6e-4, 8.2e+1), (1.9e-4, 2.8e+1), (1.4e-4, 1.2e+2), (1.1e-4, 1.8e+2)}	
8	Simple Linear	Power = 2.409e-05 * SSJ + 1.490e+02	$R^2 = 0.967$
	Quadratic	Power = -5613e-13 * SSJ^2 + 3.067e-05 * SSJ + 1.374e+02	$R^2 = 0.973$
	Cubic	Power =3.0e-19 * SSJ^3 - 5.83e-12 * SSJ^2 + 5.42e-5 * SSJ + 1.20e+2	$R^2 = 0.989$
	Stepwise Linear	{(7.7e-5, 1.1e+2), (2.6e-5, 1.7e+2), (1.4e-5, 1.9e+2), (2.2e-5, 1.7e+2), (2.0e-5, 1.7e+2),	
	CPU% 0 – 100	(1.4e-5, 2.1e+2), (2.8e-5, 1.1e+2), (2.4e-5, 1.5e+2), (2.3e-5, 1.5e+2), (2.7e-5, 1.2e+2)}	

The energy consumption information for servers was collected from http://www.spec.org, published in 2020 and 2021. To verify the accuracy of the models, one can refer to the *Actual Load*, *ssj_ops* (server-side Java operations per second), and *Average Active Power (W)* fields in the data, which are available at http://www.spec.org/power_ssj2008/results/. The code index corresponds to the server specifications listed in Table 2

penalties are expressed as an increase in a service's lifetime. The total amount of penalty is quantified as $\tau \cdot \frac{\sum_{i=0}^{n} R_r(a_{ji}) - R_c(s_j)}{R_c(s_j)}$. Since the server was configured with 65536MB (64GB) memory and it was the sole server involved in the experiment, hence $R_c(s_0) = 65536$. Moreover, denoting services deployed on server s_0 as $\{a_{00}, a_{01}, \cdots, a_{09}\}$, the total requested memory in the current step $\sum_{i=0}^{9} R_r(a_{0i}) = 86795$MB (referring to

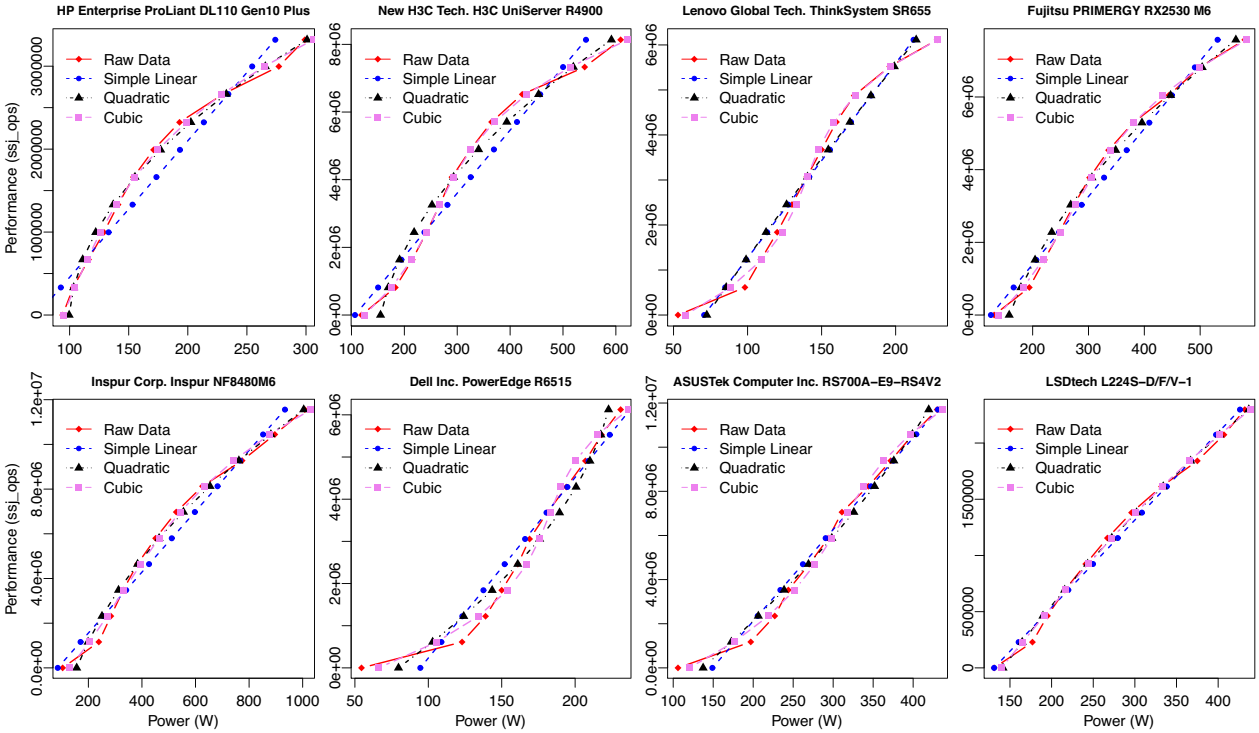

Fig. 15 Energy consumption models of the servers

Fig. 14 [left, '*currently requested*']). By default, one simulation unit represents five minutes in real time, i.e., $\tau = 300$ seconds, therefore, the total penalties for this time frame are approximately equal to 97.31598 seconds.

The second component of Eq. (1) is the distribution factor ζ, expressed as $\frac{\delta'(a_{ji})}{\sum_{i=0}^{n}\delta'(a_{ji})}$, where $\delta'(a_{ji}) = \alpha + abs(min\{\delta(\mathbf{a_j})\}) + \delta(a_{ji})$ and $\delta(a_{ji}) = R_r(a_{ji}) - R_p(a_{ji})$. Taking service a_0 as an example, $\delta(a_{00}) = R_r(a_{00}) - R_p(a_{00}) = 692 - 4760 = -4068$. This is also the smallest value among all other services deployed on s_0, i.e., $min\{\delta(a_{0i})\}_{i=0}^{9} = -4068$. In practice, a small non-zero value α is used to avoid a ratio of zero. In the present implementation of the model, $\alpha = 10$. This gives $\delta'(a_{00}) = 10$, $\delta'(a_{01}) = 3805$, and so on. The distribution factors for the services are shown in Fig. 14 [right]. Applying the distribution factor ($\zeta(a_{03}) = 19.469\%$) to service-3 (a_3), its lifetime was extended by a total of approximately 18.946 seconds ($97.31598 * 0.19469$). This expansion has a noticeable impact on the performance and resource utilization of service-3. In contrast to this significant penalty, service-0 (a_0) experienced an almost negligible increase in its lifetime, $97.31598 * 0.00014 \approx 0.0136$ seconds. This small expansion has an extremely brief impact on service-0's performance. It is worth further clarifying the differences between $R_r(a_{ji})$ and $R_o(a_{ji})$ in this example. Since the total requested resources exceed the physical capacity of the server, $R_o(a_{ji}) = R_r(a_{ji}) - \zeta(\sum_{i=0}^{n} R_r(a_{ji}) - R_c(s_j))$. Taking service-3 as an example, $R_o(a_{03}) = R_r(a_{03}) - \zeta(a_{03}) * (\sum_{i=0}^{9} R_r(a_{0i}) - R_c(s_0)) = 15406 - 0.19469 * (86795 - 65536) \approx 11267$ MB. In this case, 15406 MB system memory was requested by service-3 to complete its tasks. However, due to resource scarcity, only 11267 MB of system memory could be assigned to the service.

Furthermore, a local factor, as explained in Penalty for performance degradation section, was applied to each individual service, expressed as $(1 + \frac{\delta(a_{ji})}{R_c(a_{ji})})$. Taking service-0 ($a_0$) and service-3 ($a_3$) as examples, the local factors for the two services are ~0.5034 and ~1.5837 (where the provisioned memory for a_0 and a_3 are $R_c(a_0) = 8192$MB and $R_c(a_3) = 16384$MB, as shown in Fig. 14 [left, '*provisioned*']), i.e., the total impact on service-0 has further reduced to $0.5034 * 0.0136 \approx 0.0068$ seconds and the total impact on service-3 has further increased to $1.5837 * 18.946 \approx 30.0048$ seconds, respectively.

Accuracy of energy consumption models

In the simulator, four power consumption models were developed to predict energy usage in the clouds: Step-wise Linear Regression, Simple Linear Regression, Quadratic

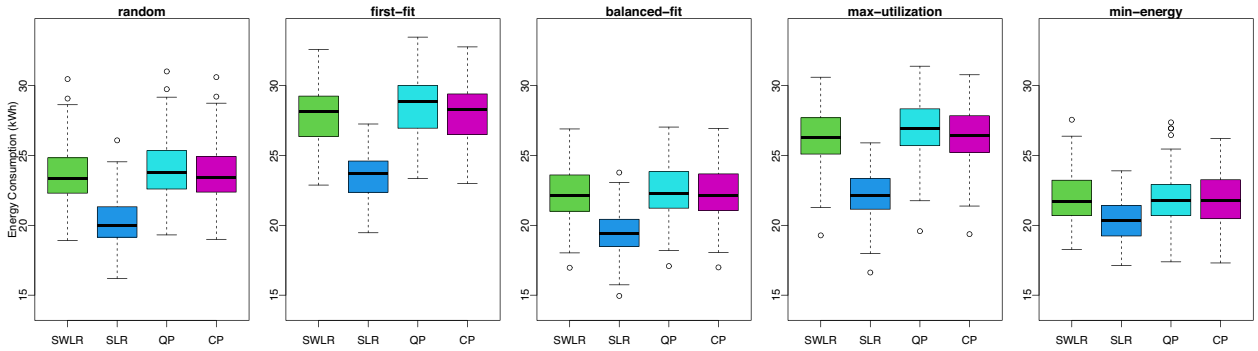

Fig. 16 The energy consumption was estimated using Step-wise Linear Regression (SWLR), Simple Linear Regression (SLR), Quadratic Polynomial (QP), and Cubic Polynomial (CP) models for ProLiant DL110 Gen10 Plus server in homogeneous clouds. The algorithms were configured with 3SE

Polynomial, and Cubic Polynomial models. The accuracy of these models was evaluated using multiple R-squared statistics, as shown in Table 6 and Fig. 15. Figure 16 presents a summary of the statistics for energy consumption collected from homogeneous clouds. The Step-wise Linear Regression model was based on individual data points, serving as a baseline for comparison. It is worth noting that the Simple Linear Regression models consistently underestimate energy usage when compared to the average baseline values, which are {rnd: 23.63, ff: 27.92, bf: 22.32, mu: 26.20, me: 21.92}. In contrast, other models produced very similar results, with the differences being statistically insignificant, {rnd: p=0.327, bf: p=0.433, mu: p=0.115, me: p=0.772}, and marginal for the *first-fit* algorithm, {ff: p=0.0481}, suggested by ANOVA tests. These findings indicate that while Simple Linear Regression consistently underestimates energy usage, other energy consumption models provide similar predictions with insignificant differences. This suggests that more complex models may not always lead to better predictions and highlights the importance of the quality and quantity of the data associated with energy consumption patterns of servers in building accurate baseline models.

The effectiveness of these models is heavily dependent on the quality as well as quantity of energy consumption data. It's important to note that these models are only approximations of actual server energy consumption patterns. The comparisons provided here serve as a reference point for further analysis and evaluation. When using different combinations of servers and energy consumption models, it will be necessary to collect basic statistics in order to establish a baseline for more thorough evaluations.

Discussion

Based on the experimental results, it is evident that the stability of the algorithms is significantly influenced by environmental heterogeneity. In other words, the variance in energy consumption and SLA violation observed in heterogeneous environments was found to be much higher than those obtained from homogeneous environments (as shown in Figs. 8 and 9). However, clouds configured with server consolidation features provide better opportunities for energy consumption reduction. Among the implemented algorithms, the *balanced-fit* algorithm demonstrated statistical robustness and efficiency in service placement for both homogeneous and heterogeneous cloud environments. It should be noted that the efficiency evaluations of the algorithms were based on specific configurations within the experiments, such as service resource utilization models, initial resource requirements of services, and server specifications. It is possible that system performance may vary for different cloud profiles. As a result, environment profiling plays an essential role in optimizing and managing cloud resources. One potential approach to address this issue involves incorporating digital twins of clouds as an abstraction layer on top of the underlying infrastructure. This would enable better environmental profiling, decision-making, and event prediction within cloud systems. The development of such a simulation model also presents opportunities for exploring decentralized and potentially self-organizing cloud architectures through experimentation and analysis.

Conclusion

In this work, an agent-based cloud simulator has been developed, offering significant flexibility in terms of parameter tuning and configurability. The simulator can be easily extended by adding new characteristics to agents or implementing other types of cloud elements through the creation of new *breeds* of agents. It is portable as it is written in a platform-neutral language, which can be executed on most mainstream operating systems facilitated by the NetLogo platform. In the evaluation,

statistical methods were employed to assess the performance of the built-in resource management algorithms. Real time plots were also utilized to visualize the runtime status of clouds. The concept of SLA violation has been redefined as service lifetime extension, which is calculated based on the relative resource requirements of services and their adjacency within the cloud environment. Future versions of this simulator will introduce new features such as a 3D model to study heat dissipation in clouds and a network module for customized network topology design.

Acknowledgements
The author acknowledge the funding provided by RDF XJTLU that made this work possible.

Authors' contributions
I declare that I am the sole author of this manuscript.

Authors' information
Dr. Dapeng Dong is an Assistant Professor with the Department of Communications and Networking at Xi'an Jiaotong-Liverpool University (XJTLU) and affiliated with the University of Liverpool (UoL). Dr. Dong earned his PhD in Computer Science from the National University of Ireland, Cork. Prior to joining XJTLU/UoL, Dr. Dong was a member of the Department of Computer Science at the National University of Ireland, Maynooth. Throughout his academic journey, he also worked as a senior postdoctoral researcher at the Boole Center for Research in Informatics, His research interests include cloud computing, multi-agent systems, and efficient big data analytics.

Funding
This work was supported by the Research Development Fund (RDF) of XJTLU [grant number: RDF-22-01-103].

Availability of data and materials
The simulator is under the GPL-3.0 open-source license, which can be downloaded from https://github.com/sys-connect/sunny. The datasets used in the paper are reproducible from the simulator. The datasets can also be obtained from the corresponding author on reasonable request.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Consent has been granted by all authors and there is no conflict.

Competing interests
The authors declare no competing interests.

Received: 9 July 2023 Accepted: 3 November 2023
Published online: 15 November 2023

References

1. Borshchev A (2013) The big book of simulation modeling: multimethod modeling with AnyLogic 6. AnyLogic Ltd, North America
2. Law AM, Kelton WD, Kelton WD (2007) Simulation modeling and analysis, vol 3. Mcgraw-hill, New York
3. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience 41(1):23–50
4. Silva Filho MC, Oliveira RL, Monteiro CC, Inácio PR, Freire MM (2017) Cloudsim plus: a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In: 2017 IFIP/IEEE symposium on integrated network and service management (IM). IEEE, Lisbon, pp 400–406
5. Railsback SF, Grimm V (2019) Agent-based and individual-based modeling: a practical introduction. Princeton University Press, New Jersey
6. Wilensky U, Rand W (2015) An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo. Mit Press, London
7. sys-connect (2023) Sunny. URL https://github.com/sys-connect/sunny. Accessed 12 June 2023
8. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. ACM SIGOPS Oper Syst Rev 37(5):164–177
9. Barr J (2006) Amazon ec2 beta. Amazon Web Services Blog. URL https://aws.amazon.com/blogs/aws/amazon_ec2_beta/. Accessed 19 Jun 2023
10. Tang X (2021) Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems. IEEE Trans Cloud Comput 10(4):2909–2919
11. Dong D, Herbert J (2013) Energy efficient vm placement supported by data analytic service. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. IEEE, pp 648–655
12. Fu X, Zhou C (2017) Predicted affinity based virtual machine placement in cloud computing environments. IEEE Trans Cloud Comput 8(1):246–255
13. Sahni J, Vidyarthi DP (2015) A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Trans Cloud Comput 6(1):2–18
14. Shi Y, Jiang X, Ye K (2011) An energy-efficient scheme for cloud resource provisioning based on cloudsim. In: 2011 IEEE International Conference on Cluster Computing. IEEE New York, pp 595–599
15. Siavashi A, Momtazpour M (2019) Gpucloudsim: an extension of cloudsim for modeling and simulation of gpus in cloud data centers. J Supercomput 75(5):2535–2561
16. Ouarnoughi H, Boukhobza J, Singhoff F, Rubini S (2017) Integrating i/os in cloudsim for performance and energy estimation. ACM SIGOPS Oper Syst Rev 50(2):27–36
17. Zhao H, Wang J, Liu F, Wang Q, Zhang W, Zheng Q (2018) Power-aware and performance-guaranteed virtual machine placement in the cloud. IEEE Trans Parallel Distrib Syst 29(6):1385–1400
18. Wickremasinghe B, Calheiros RN, Buyya R (2010) Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In: 2010 24th IEEE international conference on advanced information networking and applications. IEEE, pp 446–452
19. Mao H, Alizadeh M, Menache I, Kandula S (2016) Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM workshop on hot topics in networks. ACM, pp 50–56
20. Zhang Y, Yao J, Guan H (2017) Intelligent cloud resource management with deep reinforcement learning. IEEE Cloud Comput 4(6):60–69
21. Van Le D, Tham CK (2018) A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds. In: IEEE INFOCOM 2018-IEEE conference on computer communications workshops (INFOCOM WKSHPS). IEEE, pp 760–765
22. Kliazovich D, Bouvry P, Khan SU (2012) Greencloud: a packet-level simulator of energy-aware cloud computing data centers. J Supercomput 62:1263–1283
23. Núñez A, Vázquez-Poletti JL, Caminero AC, Castañé GG, Carretero J, Llorente IM (2012) icancloud: A flexible and scalable cloud infrastructure simulator. J Grid Comput 10:185–209
24. NS-2 (2023) The network simulator. https://www.isi.edu/nsnam/ns/. Accessed 19 June 2023
25. Varga A, Hornig R (2010) An overview of the omnet++ simulation environment. In: 1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMTOOLS). ICST, Brussels.
26. Mahmoudi N, Khazaei H (2020) Performance modeling of serverless computing platforms. IEEE Trans Cloud Comput 10(4):2834–2847
27. Pham TP, Durillo JJ, Fahringer T (2017) Predicting workflow task execution time in the cloud using a two-stage machine learning approach. IEEE Trans Cloud Comput 8(1):256–268

28. Sefraoui O, Aissaoui M, Eleuldj M et al (2012) Openstack: toward an open-source solution for cloud computing. Int J Comput Appl 55(3):38–42
29. Fehling C, Leymann F, Retter R, Schupeck W, Arbitter P (2014) Cloud computing patterns: fundamentals to design, build, and manage cloud applications. Springer, Wien
30. Tadakamalla U, Menascé DA (2021) Autonomic resource management for fog computing. IEEE Trans Cloud Comput 10(4):2334–2350
31. Chen L, Wu J, Zhang J, Dai HN, Long X, Yao M (2020) Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation. IEEE Trans Cloud Comput 10(4):2451–2468
32. Chen Y, Zhang N, Zhang Y, Chen X, Wu W, Shen X (2019) Energy efficient dynamic offloading in mobile edge computing for internet of things. IEEE Trans Cloud Comput 9(3):1050–1060
33. Wilensky U (1999) Netlogo: A simple environment for modeling complexity. URL http://ccl.northwestern.edu/netlogo/. Accessed 16 June 2023
34. Jin C, Bai X, Yang C, Mao W, Xu X (2020) A review of power consumption models of servers in data centers. Appl Energy 265:114806
35. Clark C, Fraser K, Hand S, Hansen JG, Jul E, Limpach C, Pratt I, Warfield A (2005) Live migration of virtual machines. In: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2. pp 273–286

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

► Convenient online submission
► Rigorous peer review
► Open access: articles freely available online
► High visibility within the field
► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com