The Journal of Supercomputing (2024) 80:13434–13474
https://doi.org/10.1007/s11227-024-05990-5

A novel multi-level hybrid load balancing and tasks scheduling algorithm for cloud computing environment

Nadim Elsakaan[1] · Kamal Amroun[1]

Accepted: 9 February 2024 / Published online: 6 March 2024

Abstract

Ensuring optimal load balancing is imperative for maintaining reliability and upholding quality of service as specified in service-level agreements (SLAs) for cloud computing providers. This research addresses the most common shortcomings of existing state-of-the-art methods, which often lack responsiveness and struggle to adapt to exponentially increasing demand, especially in the era of the internet of things (IoT). The proposed hybrid approach surpasses current literature approaches in performance metrics such as makespan, response time, number of cloudlet migrations, and SLA violations. It operates on two levels, initially employing a k-means clustering algorithm to group servers within each datacenter based on similar utilization rates. Subsequently, a round-robin method allocates task groups sequentially to non-overloaded clusters, and within each cluster, a genetic algorithm optimally assigns tasks to servers. This multilayered approach facilitates hot-deployment and scalability in operational cloud environments while promoting strong interoperability and decoupling of core mechanisms missions. Simulation experiments conducted on CloudSim Plus validate the superiority of our method, positioning it as a robust solution for enhancing load balancing and tasks scheduling in cloud environments, especially in the face of rapidly increasing IoT-related demands.

Keywords Load balancing · Quality of service (QoS) · Service-level agreements (SLAs) · Cloud computing · Hybrid approach · K-means clustering · Round-robin · Genetic algorithm · Makespan · Response time · Cloudlet migrations · Hot-deployment · Scalability · Interoperability · CloudSim Plus

✉ Nadim Elsakaan
nadim.elsakaan@univ-bejaia.dz

Kamal Amroun
kamal.amroun@univ-bejaia.dz

[1] LIMED Laboratory, Faculty of Exact Sciences, University of Bejaia, 06000 Bejaia, Algeria

1 Introduction

Cloud computing was a game-changer technology since its emergence nearly a decade ago, the key feature behind this revolutionary paradigm is its ability to provide resources like hardware or software as services over internet to individuals and companies. It offers a certain number of advantages like elasticity, pay as you go, multi-tenancy and so on [1]. It is worth pointing that it must guarantee sustainably users trust by maintaining availability, reliability and scalability. If we take a closer look at how this technology works, we realize that behind the simplicity of use and service delivery models, there is a set of extremely complex mechanisms interoperating to ensure an optimal functioning [2].

Our work focuses on load balancing (LB) module which is one among these mechanisms and is in charge of maintaining a fair workload distribution between servers and virtual machines (VMs). The role of this component is crucial to the operation of cloud services and to meet the service-level agreement (SLA). Guaranteeing an optimal use of hardware resource and a fair distribution of the workload helps to increase the global performance of the system by reducing the makespan of useful jobs [3]. A lot of propositions have been made to face expectations of cloud service providers in term of load balancing. When studying the literature, we understand that these solutions can be categorized according to two criteria impacting the manner in which workload distribution is made: first (i) information on environment, tasks and resources. Then, (ii) step during which the balancing occurs.

Indeed, while a first category to which we refer as being static acts only at reception of new tasks and decides on how assigning them according to a set of non-evolving information like tasks length and servers physical capabilities, a second category regroups dynamic approaches which are continuously operational over time incorporating information on current workload on each server, individual makespan of virtual machines and primitives like tasks or VMs migration to maintain an optimal utilization rate of all servers. A last category stands for regrouping all approaches that combine static and/or dynamic approaches together or with other complementary mechanisms to enhance performance of load balancing and to mitigate common shortcomings [4, 5]. This is the way we build in our proposal, by integrating complementary algorithms for hybridizing load balancing and tasks scheduling.

There are several ways to approach load balancing problem and tasks scheduling. We commonly find solutions formulating it as a bin packaging problem, clustering problem or even more like a path finding problem. Independently of the formal modeling, we can describe the elements that constitute the problem we are addressing as follows: given a set of tasks and a set of resources grouped into virtual machines and physical hosts, how to decide for each task to which virtual machine should it be assigned and which physical server should host which virtual machine. It is important to keep in mind that each server is resource limited in term of CPU, RAM, Storage and Bandwidth [6]. Once this has been achieved, we need to monitor the evolution of the use of these same resources and the makespan of servers and VMs to ensure fairness in the distribution of workloads and enable an increase in datacenter performance by reducing metrics such as waiting, execution, control and migration times.

The aim of this paper is to present a novel hybrid algorithm to ensure tasks scheduling and load balancing in cloud environments. First of all, the choice of this combination comes from the fact that a good workload balance starts with the optimization of the tasks assignment procedure. Our algorithm goes through three main stages: first (i) servers clustering: a k-means-based procedure is first triggered to partition servers with similar occupation characteristics in a set of clusters with bounded size. Then, (ii) tasks assignment is realized in two phases: first using a round-robin algorithm to choose the cluster to which a set of tasks will be assigned then using a genetic-based algorithm to select inside this cluster the servers on which they will be scheduled. And finally (iii) load balancing: also requiring two steps, the algorithm decides which cluster to unload and which servers exactly. Once done the cloudlets are retrieved and sent (migration) to the global tasks scheduler module to plan them again.

To realize this our architectural model embed following components, first we introduce a new module called cluster manager which will ensure primitives relative to creating and updating clusters. Then, we use a classical tasks scheduler at datacenter level qualified as global to which we add a local monitor in each cluster. And finally, we propose the same organization for load balancer with a central module and local probe in each cluster.

The method that we propose is based on realistic assumptions, using particular configurations of already existing architectural components. The main contributions of our work are:

- A hot-deployment enabled algorithm which can be deployed in already operational cloud environments.
- A highly scalable algorithm, thanks to the strategy grouping the servers in clusters and operating at two levels, it keeps the same performance even if the number of servers and cloudlets in the datacenter is considerably increased.
- A very strong interoperability and complementary that avoids interference in mission of each and redundancy of actions between clusters management, load balancing and tasks scheduling mechanisms. This decreases the non-useful delays related to cloud management and allows to considerably reduce the number of SLA violations.

To prove the validity of our approach, we implemented it using the standard cloud simulator CloudSim plus. The results are very promising. Indeed, our approach outperforms most recent and relevant works. For example, we achieved an SLA violation rate close to 8%, compared to an average of 18% for [7]. We also achieved a reduction in the makespan per server and reduced the proportion of migrations required to 12%, compared with a ratio varying from 19 to 35% according to the proposal.

The rest of this paper is organized as follows: core concepts on cloud computing and literature review of existing solutions to perform load balancing are given in

Sect. 2. Section 3 is dedicated to our proposition, we start it by formulating the problem statement, and then, introducing our architectural model with related assumptions, we finally depict our method and give corresponding algorithms. Results of realized simulation are given and discussed in Sect. 4. Section 5 concludes our work by giving a last overview and by opening perspectives for future works.

2 Related works

In this section, we will first review core concepts and preliminary elements on cloud computing, its core concepts and service providing models. Then, we will summarize following respective categories that we consider to be the most pertinent and recent works on load balancing.

Cloud computing was a game-changer technology since its emergence nearly a decade ago, the key feature behind this revolutionary paradigm is its ability to provide resources as services directly over internet to individuals and companies. The resource can be hardware or software or even take another form; it offers a certain number of advantages like elasticity, pay as you go, multi-tenancy and so on [1, 8].

The cloud providers deliver services in several standardized manners among which we retain [9]:

- IaaS (Infrastructure-as-a-Service): Hardware is delivered to client which is in charge of installing all the stack components over material: operating systems, middlewares, runtime environments and applications. The cloud provided is only responsible on management of hardware part.
- PaaS (Platform-as-a-Service): The responsibility of the supplier is shifted a little higher in the stack; he will be in charge of the installation and the management of operating systems, middlewares and all required execution environments.
- SaaS (Software-as-a-Service): In this pattern client interacts with cloud services via a GUI (graphical user interface), indeed all required services are delivered as a ready to use application and provider is responsible on the entire building stack.

After reviewing the different possible architectures and organizations for the operation of the cloud, we came to the conclusion that, regardless of the model, the architectural elements can be placed in one of the three levels as shown in Fig. 1:

- Requests handler: Covering a set of components in charge of collecting requests from client and retrieving related information such as tasks length, priority, deadline, required data and so on.
- Datacenter controller: It plays an orchestration role; on the one hand, it receives information from the requests handler on tasks to schedule; on the other hand, the resource manager provisions it with available resources. The controller then applies a scheduling strategy to decide which tasks to assign to which VM, and which host will receive which VM.

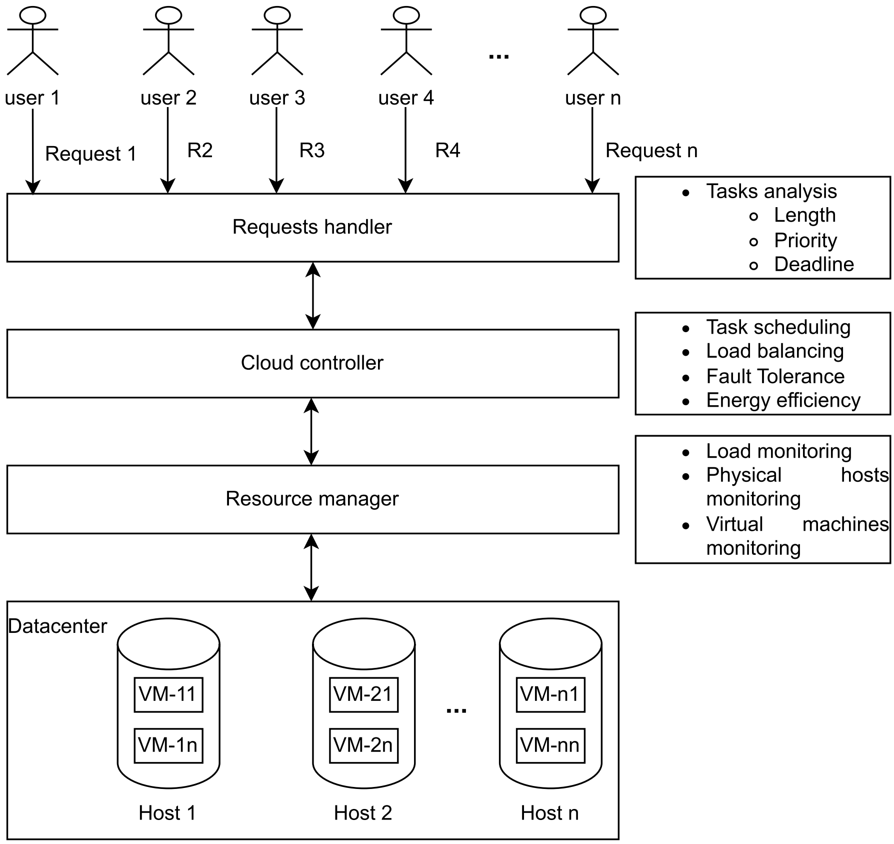

Fig. 1 Cloud datacenter organizational architecture

- Resources manager: In charge of monitoring states and utilization rates of hosts and virtual machines, it provides crucial information on which the controller relies to schedule tasks and to perform load balancing.

Load balancing (LB) is a critical module to ensure good functioning in cloud environment; it aims to describe techniques in charge of distributing workload over datacenter servers. In other words, it is the method used to maintain resource utilization of servers in equilibrium and avoid to overload or underload some of them. The load balancing can be achieved at one of the two levels: (i) virtual machines (VM) level or (ii) hosts level. In first case, the load balancing algorithm deals with workload on virtual machines, so it manages tasks distribution and migration over virtual machines to maintain good workload partition conditions. In the second case, it manages virtual machines distribution over physical servers.

A lot of approaches were proposed in the literature to perform load balancing in cloud environment, we will give insights on the most important ones in Sects. 2.1–2.3. Whatever it is critical to first understand the used categorization.

Indeed, a classification of algorithms that is often encountered involves three main classes which are depicted as follows [4]:

- Static approaches: This category of approaches relies on prior information on jobs and servers/virtual machines capabilities to decide on tasks assigning policy.
- Dynamic approaches: In contrast to static methods, dynamic algorithms integrate real-time information such as workload on resources and utilization rate to decide on tasks affectation strategy.
- Hybrid approaches: Hybrid approaches are obtained by mixing static and dynamic approaches in order to overcome lacks of each one. Even more lot of researchers go further and make hybridization by combining load balancing techniques with fault tolerance or tasks scheduling mechanisms.

There are other modules involved in the smooth running of jobs on the cloud that are intrinsically linked to load balancing and cannot operate without close cooperation between them.

The first mechanism is tasks scheduling which, considering a set of constraints, decides for a set of tasks on which resource they should ideally be executed. This can be achieved in a static or dynamic way regarding if the scheduler relies on prior information on tasks and resources for taking decisions or if it continuously monitor nodes behavior to decide which one is more suitable for a specific kind of job. It can be preemptive so that tasks can be interrupted during running time or not, it can operate in online or offline manner according to if tasks are directly planned on resources or grouped in batches before [10].

The second mechanism is fault tolerance which measures the capacity of a system to recover after a failure. A failure can be explained as a succession of undesirable actions which lead the system to an unsuitable or a not specifications conform state. Two main families of fault tolerance approaches can be introduced: (i) proactive: In this case, the technical effort is focused on ways to anticipate the failure and reduce its impact on the overall system. (ii) Reactive: In this case, means are concentrated on the methods that make it possible to recover quickly after the fault occurrence and try to bring back the system to the last known coherent state. A lot of known approaches like hardware-redundancy, jobs replication, checkpoint and restart and so on are utilized to ensure reliability, availability and integrity in cloud environment [11].

2.1 Static load balancing methods

Static load balancing is a class of algorithms that allocate tasks to different resources without considering their current states. Indeed, depending on the applied policy, the algorithm will distribute new jobs equally or randomly over all treatment units regardless to their actual workload [12].

Among the static load balancing methods, we can cite min–min algorithms whose principle is to evaluate beforehand execution time of each task and to find the one with the shortest duration. Once done, the algorithm locates the resource with minimum completion time to perform this task and assigns it. Min–min approaches repeat these steps until all jobs are done. The major shortcoming these methods are suffering from is when the number of short tasks exceed the number of long ones the allocation of resources is not optimal. Another category is the max–min which overcomes this shortage by scheduling first larger tasks but it penalizes short ones and increases their waiting time [4].

Lot of improvements were proposed to make these approaches that perform better load balancing and reduce makespan. For example, Kokilavani et al. [13] proposed to start their algorithm with a min–min phase, which provides a fast start by sending the shortest tasks to be executed by the most efficient resources. In a second step, the algorithm checks each resource makespan and retrieve tasks from ones with heavy load and reassigns them to resources with short makespan. The principle of the LBMM approach is simple, but they showed that it allows a reduction of the global execution time and a better jobs distribution. Another group of researchers goes further in improving min–min approach by proposing a new algorithm considering three key constraints in cloud environment which are quality of service, tasks priority and cost of service. Their solution also passes by a first min–min step where short tasks have initial higher priority; then, they rearrange load balance by weighting these priorities by the three constraints (expressed as numerical values) to produce dynamic priorities allowing to order all the jobs over the resources [14]. We can find in the literature approaches oriented toward particular use cases, or ones based on meta-heuristics inspired from nature. An example that embodies both is the proposition made by Zhan et al. [15] where they use a discrete particle swarm optimization (PSO) for building a static load balancing algorithm which ensures tasks distribution in cloud environment. They proposed an adaptation of functions to update personal and global bests on one hand and to update velocity on the other hand; this renders PSO performing better for this particular discrete problem avoiding to be trapped in local optimums. As powerful as, they may be that these techniques have the disadvantage of not being able to adapt to increasingly dynamic cloud environments. Static approaches schedule tasks at the reception and rely on a logic independent of real-time workload distribution; this does not allow them to review the load distribution in a fluid way when the exploited resources utilization evolves. In other words, static load balancing performs well in cloud environments with reduced workload variability which is not always adequate given that there are peak periods.

2.2 Dynamic load balancing approaches

Dynamic load balancing techniques are the set of algorithms that consider real-time information about the utilization rate and remaining makespan on each server before deciding how to assign new jobs and in which manner to migrate already scheduled ones. These approaches can be separated into two main categories according to their calculation mode which are online or offline mode. In the first family of algorithms, tasks are assigned as they arrive in the system unlike offline mode which works by batch since tasks are grouped then taken at predefined times [16].

A category of methods stands out as an ideal candidate to face the dynamic load balancing challenge: nature-inspired meta-heuristics. By going through several stages of adaptation, many researchers have succeeded in making it a dominant class of approaches for dynamic LB. It is worth pointing that before proceeding to any improvement of this kind of algorithm, it is necessary to define an adequate mapping between the parameters of the algorithm and the cloud environment on the one hand and on the other hand to find a way to define novel search functions [17].

For example, the authors of [18] proposed an enhancement of bee colony optimization algorithm to realize a dynamic load balancing. This approach incorporates as constraints avoiding at once virtual machines overloading and underloading, reducing makespan and the number of migration operations. The key idea behind the proposed improvement is to use standard deviation of processing time on each VM as input to the model of load balancing where a threshold is given separating VMs into two groups: overloaded VMs modeled as honeybees and underloaded VMs modeled as food. Considering that the approach is dynamic, the deviation values are updated each time; a new task is received. On their side, Seyedeh et al. [7] have combined two meta-heuristic approaches to better fit cloud SLA requirements. They first use a firefly based algorithm to generate an initial population of possible task/resource assignment then optimize it using an imperialist competitive algorithm (ICA). In a first time, two instances of firefly algorithm are executed separately to find two assignments: one achieving best makespan and the other for best load balancing. The outputs of these heuristics are aggregated as multi-objective function for the ICA algorithm which will incorporate the two constraints and attempt to produce a workload balance ensuring a makespan as small as possible. Another approach using a meta-heuristic bio-inspired is proposed in [19] where authors approach the load balancing as a clustering problem: regrouping sets of virtual machines on physical servers with specified CPU and memory capacities. Clusters are first build by randomly placing VMs according to a feasible distribution; then, they are updated based on their respective workload to maximize remaining resources on each server and eliminating workload on weakly utilized ones. Authors applied bat algorithm to optimize global and local search for finding new cluster centers and speed up convergence. This means reviewing the set of VMs and physical servers that make up each of the clusters in order to dynamically maintain equity.

Many other works have been proposed and tried to create variants that improve a subset of the performance criteria. This objective can be achieved by eliminating or adding constraints depending on the service-level agreement and the type of service provided, by modifying hyper-parameters of known models or by proposing new fitness functions for meta-heuristics. Thus, Dalia et al. [20] add constraints not commonly considered by other researchers to their algorithm system. They add complexity by dealing with simultaneous arrival of requests, the tasks that they consider have priorities and a deadline is assumed for each job according to the relative service-level agreement. The distinction point of this approach is that if the workload on a server does not meet requirement of correct execution for a given task; then, it is migrated to another one. Therefore, in order to realize an efficient workload distribution, the authors integrated load balancing and task scheduling within the same algorithm. Another example is given in [21] where the authors provide a variant of genetic algorithm which deals with performance degradation of VMs during migration time and impact on tasks execution.

Authors of [22] dived deeper in key constraints integration by adding elasticity to their model. Their architecture supports hardware proactive horizontal scale-up. Indeed after assigning tasks a component of the resource broker monitors activity on servers and estimate if there are scheduled tasks that will overhead their deadline and decide to create new virtual machines to balance the workload.

It is worth pointing even if these meta-heuristics build the backbone around which most of the load balancing algorithm in the cloud are built; other methods exist and approach this problem by a different modeling or statement and take advantage of the power of several different mathematical techniques.

We often meet in the literature fuzzy-based approaches to achieve tasks scheduling and load balancing. We can observe that authors in [23] proposed a fuzzy-based algorithm for multidimensional resource planning focusing on file sharing service in cloud environment. The approach operates in three stages: first, (i) collecting requests from users. Then, (ii) a trapezoidal fuzzification and fuzzy square inference are utilized to achieve multidimensional resource scheduling. Finally, (iii) queuing network is designed for the assigned tasks and resources. We can also find methods relying on machine and deep learning such as the one proposed by Zhao et al. [24] in which they combine a Q-learning approach with a neural network. They first represent the scheduling plan by a directed acyclic graph in which nodes are described by a quaternion composed of a specific task, the cost of execution, cost of communication and edges representing relation to successor tasks. The dynamic scheduler before planning a workflow by distributing its jobs on virtual machines calls the algorithm to evaluate the execution scenario which is modeled in the form of a graph and applies a reward function which helps the decision-making by emitting an action to be applied by the scheduler. Another hybridization with a meta-heuristic method is proposed by Jena et al. [25] where a particle swarm algorithm is combined with a Q-learning approach used to adjust velocity of particle and global bests to achieve quicker convergence toward optimal load balancing solution.

Dynamic approaches are responsive to constraints evolution in real-time; they keep maintaining an equitable distribution of workload over time and balancing the load flow as it occurs by taking into consideration the resource usage on each host. The only disadvantage is that it suffers from compared to static methods that it generates some latency at the start of the LB and is bounded on reaction times even with light and few tasks.

2.3 Hybrid load balancing algorithms

Hybrid load balancing is a combination of static and dynamic approaches. It leverages the strengths of each category to cover the gaps of the other; static methods allow an initial quick distribution of tasks while dynamic algorithms maintain optimal workload balancing over time. Hybridization is not limited to this combination and can be achieved by integration with other mechanisms such as fault tolerance or dynamic task scheduling.

Bio-inspired meta-heuristics also form a cornerstone in the edifice of hybrid approaches. Indeed, the literature of hybrid LB is rich in examples such as the proposition made by Marwa et al. [26] where the authors combined swarm intelligence of bee and ant colonies to build an osmotic hybrid optimization load balancing algorithm. After an initial random distribution of jobs, an artificial bee colony is used for quickly find overloaded and underloaded servers; then, an ant colony is used to find optimal migration scheme for VM among osmotic servers.

In another proposal [27], a group of researchers has opted for the combination of ant colonies with fuzzy models. While they also use ACO to find the suitable migration pattern, they integrate a fuzzy module to evaluate the quality of the returned solution. To be more concise, the fuzzy part is used to update the pheromone traces and thus accelerate the convergence process toward an optimal solution. In [28], authors proposed to combine genetic algorithm and gravitational search method to enhance searching procedure and reduce computing cost. The improvement comes from hybrid method to calculate particles position at each step which is made by using crossover technique combined with gravitational constant function. In [29], authors combined a queuing model for managing virtual machines and a crow search-based approach to improve tasks placement and reduce at once time wastage and energy consumption.

Other researchers go further by hybridizing load balancing with proactive fault tolerance mechanism. In [30], authors proposed to combine several reactive and proactive fault tolerance techniques with an accelerated decision-making procedure to ensure quick recovery. The heart of the proposed method is a dynamic scheduling approach which inserts replication as fundamental constraint. Haoran et al. [31] proposed a similar combined approach by embedding fault tolerance and task scheduling into their model which improves load balancing. The authors focus on hybrid real-time tasks which they divided to data intensive, process intensive tasks or balanced tasks; by doing the same with virtual machines, it increases the chances of improving system performance and facilitates the work of the scheduler. They did not stop here; they went further by mixing checkpoint and primary backup techniques to generate the recovery policy and the corresponding task description. Finally, they added resulting tasks list (including redundancy) as constraints to the scheduler.

Huaiying et al. [32] have also proposed an approach to ensure quality of service in edge-cloud by combining fault tolerance with task planning in order to maintain equitable load balancing. The authors enhance classical primary/ backup fault tolerance approach by incorporating quality of service (QoS) constraints such as time-based ones, the primary and copy tasks are then scheduled with a dedicated method integrating an adjustment procedure that guarantees the placement of copies in a such manner to reduce both recovery time in case of failure and overlapping in case of well-functioning. A last example of this kind of hybridization is given in [33]; apart the planning aspect, the authors proposed a solution for monitoring activities on the virtual machines which form logical clusters over physical hosts. They used metrics built based on previous performance of each server that allows the system to early anticipate deviations and behaviors that do not conform to specifications which allows it to quickly resume from the last consistent checkpoint.

We have not provided an exhaustive list of the hybrid proposals that exist, and it must be taken into account that many other works have been done. A last example is given in [34] where authors rely on machine learning technique to enhance resource utilization; they deal with horizontal and vertical load balancing. An agent is trained using a custom reinforcement learning approach and is rewarding according to the desirability of selected action which can be task assigning on a specific virtual machine or migration on another host and so on. Other works deviate from the mainframe application and focus on specific context like [35] which ensures a dynamic resource provisioning for a specific usage in meteorological intensive dataflows treatment.

As we have shown, hybrid approaches cover the shortcomings of static- or dynamic-isolated techniques, they have the advantage of being more reactive; managing more constraints and being faster in the distribution of tasks, they nevertheless suffer from the drawback of being complex to implement.

2.4 Comparative analysis

Table 1 provides a comparative analysis of the latest and most pertinent approaches presented in this section, drawing upon structural criteria such as:

- The nature indicating whether the approach is static, dynamic or hybrid.
- Combination to describe the eventual integration with any other mechanism, including fault tolerance or task scheduling.
- The core method or meta-heuristic forming the foundation of the approach.
- The benefits offered by the approach.
- The constraints or challenges encounter in its presentation.
- The performance metrics utilized for validation and assessment of the approach.

2.5 Synthesis

Load balancing plays a crucial role in keeping cloud environments running smoothly. Its performance has a direct impact on quality of service (QoS) and the degree of compliance with service-level agreements (SLAs) with customers. Numerous approaches have been proposed since the advent of the cloud to meet this need. Firstly, static approaches were proposed, offering a rapid search for a balancing solution based on invariable information. Unfortunately, as they were unable to adapt to changes in the environment and load variability, they were gradually replaced by dynamic approaches. The latter rely on heuristics and attempt to find optimal solutions by integrating variable constraints that were previously ignored. Although dynamic approaches are effective, they are slow to execute. This has led to the emergence of hybrid approaches, combining static methods for speed in finding a feasible

Table 1 Comparison of load balancing algorithms

Approach	Nature	Combination	Core method	Merits	Limitations	Metrics
Kokilavani and Amalarethinam [13]	Static	No	Min–Min	Fast	Not adaptative	Makespan
Miao et al. [15]	Static	No	Particle Swarm Optimization	Fast	Not adaptative	Makespan
Babu and Samuel [18]	Dynamic	No	Bee colony	Fast	VM level migrations	Makespan
				Semi-adaptative		Migrations
Shafiq et al. [20]	Hybrid	Task scheduling	Opportunistic load balancing	IaaS levele	Ignore common constraints	Makespan
			Min–min	Optimize resource		Resource utilization
Adhikari et al. [19]	Hybrid	Task scheduling	Bat fly algorithm	Considers pricing	Ignore common constraints	Makespan
						Flow time
						Resource utilization
Vanitha and Marikkannu [21]	Dynamic	No	Genetic algorithm	Fast	Lack of performance evaluation	Performance degradation
Kumar and Sharma [22]	Hybrid	Task scheduling	Custom method	Fast	Not adaptative	Makespan
				Task-level migration		
				High scalability		Migrations
Kashikolaei et al. [7]	Hybrid	Task scheduling	Imperialist competitive algorithm	Fast	Complexity	Resource utilization
		Sort of fault tolerance	Firefly algorithm	Considers linked tasks	Lack of performance evaluation	Makespan
Priya et al. [23]	Hybrid	Task scheduling	Fuzzy-based logic	Very-responsive	Complexity	Resource utilization
				Fast	Lack of performance evaluation	
				Resource optimization		

Table 1 (continued)

Approach	Nature	Combination	Core method	Merits	Limitations	Metrics
Tong et al. [24]	Hybrid	Task scheduling	Deep Q-learning	Highly-adaptative	Complexity	Makespan
				Scalability	Lack of performance evaluation	
Ragmani et al. [27]	Hybrid	Task scheduling	Ant colony algorithm	Highly responsive	Complexity	Response time
			Fuzzy-based logic		Lack of performance evaluation	
Gamal et al. [26]	Dynamic	No	Ant colony algorithm	Enhance QoS of service	Complexity	Energy consumption
			Bee colony algorithm	Optimize energy consumption	Lack of performance evaluation	Migrations
Abohamama et al. [30]	Hyrbid	Fault tolerance	Custom method	Optimize cloud resource utilization for real-time application	Complexity	Makespan
					Lack of performance evaluation	
Chinnathambi et al. [33]	Hybrid	Fault tolerance	Byzantine fault tolerance	Synchronous checkpointing which keeps a global job consitency	Ignore common constraints	Makespan
						Energy consumption
					Slow	SLA violations

solution and dynamic methods for optimizing these solutions. Hybrid approaches are widely used today but suffer from the complexity in implementation phase.

3 Our proposal

We discussed in the previous section of the existing solutions to achieve load balancing in cloud environments. We will now in this section introduce our novel method, we first formalize the problem statement in Sect. 3.2, then expose the architectural model on which our proposal relies and its related assumptions respectively in Sects. 3.3 and 3.1. Finally, we give an overview on the solution in Sect. 3.4 before we dive deeper and depict it in algorithms within Sect. 3.5.

3.1 Assumptions

In order to avoid confusion, it is important to make some assumptions upon which our approach is built:

- The load balancing is realized at cloudlets level; therefore, once a cloudlet is selected for migration it is forwarded to the global tasks scheduler to be planned again.
- A workload is already present in the datacenter before the deployment of our algorithm and is randomly distributed on the servers. A major advantage of our approach is that it can ensure hot-deployment in already operational datacenters. However, our framework can also be deployed in new datacenters without workload.
- We are only interested in the main resources of a server which are: the CPU, RAM, bandwidth, storage.
- A task is executed to completion without interruption and randomly utilizes virtual machine resources according to a particular model (selected for simulation scenario).
- A virtual machine is destroyed if it finishes executing tasks in its queue before the scheduler has assigned it other tasks (there is no fully unoccupied virtual machines).

3.2 Problem statement

Keeping in mind that the main purpose of tasks scheduling and load balancing hybrid algorithm is first to decide how to assign a task to a specific virtual machine which is running on a particular server, then to focus on the manner that allows to maintain a workload distribution such that it eliminates overloaded and underloaded servers and try to bring them all within a moderate and fair usage level. In this section, we will formalize the problem; we face and introduce used formulas.

In the following sections, the notations used, along with their corresponding explanations, are summarized in Table 2.

Table 2 Notations meaning

Notation	Meaning	Notation	Meaning
R	Resource	CL	Computation load
MIPS	Million instructions per second	RL	RAM load
BW	Bandwidth	SL	Storage load
STR	Storage	BDU	Bandwidth utilization
CP	Computation power	RH	Request handler
ET	Execution time	CM	Cluster manager
CT	Completion time	GTS	Global tasks scheduler
WT	Waiting time	LTS	Local tasks scheduler
VM	Virtual machine	GLB	Global load balancer
CUR	Cluster utilization rate	LLB	Local load balancer
CMS	Cluster makespan	MMS	Mean of makespan

The resources available in a datacenter constitute the set of resources accessible at the servers level, and this can be expressed through Eqs. 1 and 2 [19]. Let assume that each datacenter is composed of a set denoted S of N servers $S = \{S_1, S_2, \dots, S_n\}$ to which corresponds a set of Resources as described in Eq. 1:

$$\text{Resources} = \Big\{ \{R_1^{\text{cpu}}(t), R_1^{\text{ram}}(t), R_1^{\text{bw}}(t), R_1^{\text{str}}(t)\}, \{R_2^{\text{cpu}}(t), R_2^{\text{ram}}(t), R_2^{\text{bw}}(t), R_2^{\text{str}}(t)\}, \dots, \{R_n^{\text{cpu}}(t), R_n^{\text{ram}}(t), R_n^{\text{bw}}(t), R_n^{\text{str}}(t)\} \Big\} \tag{1}$$

where $R_i^{\text{resource}}(t)$ gives the remaining level of resource of server i at instant t. We assume also that on each server evolve a set VM of M virtual machines such that: $\text{VM} = \{\text{vm}_1, \text{vm}_2, \dots, \text{vm}_n\}$. Each virtual machine lives on a physical server and has dedicated resources, for example, the first vm assigned to the first host noted vm_{11} has as remaining resources at instant t as given by Eq. 2:

$$R_{11}(t) = \{R_{11}^{\text{cpu}}(t), R_{11}^{\text{ram}}(t), R_{11}^{\text{bw}}(t), R_{11}^{\text{str}}(t)\} \tag{2}$$

We assume that all virtual machines are initially of a same fixed amount of resources: two CPU cores with each 5000 MIPS and four Go of RAM.

Each task has two main parameters which are length and resources utilization model; we commonly also call this the cloudlet model. According to its type, a cloudlet has a determined size (quantity in MIPS) and a variable percentage (from 0.2 to 0.8) of available resources utilization such as RAM. Table 3 gives an overview on available server types with corresponding resources (number of processing cores, RAM and each core calculation capacity) and possible cloudlet sizes.

To formally articulate the problem, we will introduce crucial parameters and associated equations in our approach. These parameters, commonly employed in the literature, can be categorized into two main groups: (i) temporal-based parameters [25], illustrated by Eqs. 3–9, and (ii) load-based parameters [19], represented by Eqs. 10–15.

Eq. 3 allows the calculation of computation power of a host or a virtual machine:

Table 3 Hosts and cloudlets configuration

Type	CPU cores	RAM (Go)	Core size (MIPS)	Cloud-let size (MIPS)
Small	2	8	5000	3000
Medium	4	16	10,000	5000
Large	8	32	15,000	7000
Extra large	16	64	30,000	10,000

$$\mathrm{CP}_i = |\mathrm{P.U}| * \mathrm{sizeof(p.u)} \tag{3}$$

where P.U is the set of allocated processing units, $|P.U|$ is its cardinality and sizeof(p.u) is the individual capacity of a processing unit in millions of instructions per second (MIPS).

The execution time of a task on a specific virtual machine is given by Eq. 4:

$$\mathrm{ET}_{ji} = \frac{\mathrm{sizeof}(\mathrm{task}_j)}{\mathrm{CP}_{\mathrm{VM}_i}} \tag{4}$$

Equation 5 gives the full completion time of a task on a virtual machine:

$$\mathrm{CT}_{ji} = \mathrm{WT}_{ji} + \mathrm{ET}_{ji} \tag{5}$$

where WT_{ji} is the waiting time of the task_j on the $\mathrm{virtual\_machine}_i$ and is given by Eq. 6:

$$\mathrm{WT}_{ji} = \sum_{k=1}^{j-1} \mathrm{CT}_{ki} \tag{6}$$

i.e: waiting time of a task on a specific virtual machine is the sum of completion times of all preceding tasks.

We can now define the makespan as the main parameter which measures the entire completion time of all tasks. It is respectively given on virtual machine, hosts and clusters levels by 7, 8 and 9:

$$\mathrm{makespan}(\mathrm{VM}_i) = \sum_{j=1}^{n} \mathrm{CT}_{ji} \tag{7}$$

$$\mathrm{makespan}(\mathrm{host}_i) = \max\{\mathrm{makespan}(\mathrm{VM}_{ji})\} \tag{8}$$

$$\mathrm{makespan}(\mathrm{cluster}_i) = \max\{\mathrm{makespan}(\mathrm{server}_{ji})\} \tag{9}$$

Now, we are done with temporal or time based parameters, we will move to define the most important used load related parameters. The first of all estimates the load on processing unit and is given by Eq. 10:

$$CL_i^t = CL_i^{t-1} + CL(Tasks_{ji}^t) - CL(finished\_tasks_{ji}^t) \tag{10}$$

In the same manner, we can obtain the load on the RAM and storage at a particular timestamp by Eqs. 11 and 12, respectively:

$$RL_i^t = RL_i^{t-1} + RL(Tasks_{ji}^t) - RL(finished\_tasks_{ji}^t) \tag{11}$$

$$SL_i^t = SL_i^{t-1} + SL(Tasks_{ji}^t) - SL(finished\_tasks_{ji}^t) \tag{12}$$

This estimation is obvious, considering that on a particular a virtual machine, at a specific timestamp, the load corresponds to the already present workload to which we add the load induced by new assigned tasks and deduce load of finished ones. The utilization of bandwidth on a virtual machine is obtained by summing the amounts of data streams generated by active tasks and can be calculated with Eq. 13:

$$BDU_i^t = \sum_{i=1}^{n} data\_stream(task_i) \tag{13}$$

The global load score on a virtual machine is then given by Eq. 14:

$$Load(VM_i) = \alpha * CL_i + \beta * RL_i + \gamma * SL_i + \sigma * BDU_i \tag{14}$$

where α, β, γ and σ are pondering and normalizing factors. Then, we can calculate the load of a particular server by Eq. 15:

$$Load(server_i) = \frac{\sum_{i=1}^{m} Load(VM_{ji})}{M} \tag{15}$$

These formulas are common ones, and we will make use of them in our method in order to build clusters, define a task scheduling strategy based on genetic algorithm and improve load balancing within cloud environment.

3.3 Architectural model

Our method is built upon the architectural organization shown in Fig. 2 which is nearly similar to standard ones (see Fig. 1). Our method relies on hybridizing between tasks scheduling and load balancing, it incorporates a new module called clusters manager which is new and crucial to the purpose of this paper. We deliberately omit other components such as energy efficiency and so on. We will refer to mechanisms operating at datacenter level as level-2 and ones operating at clusters level as level-1. The major components/modules involved in our solution can be described as follows:

- Requests handler (RH): It builds the interaction interface with end users. It retrieves important information relative to each request like length and deadline.

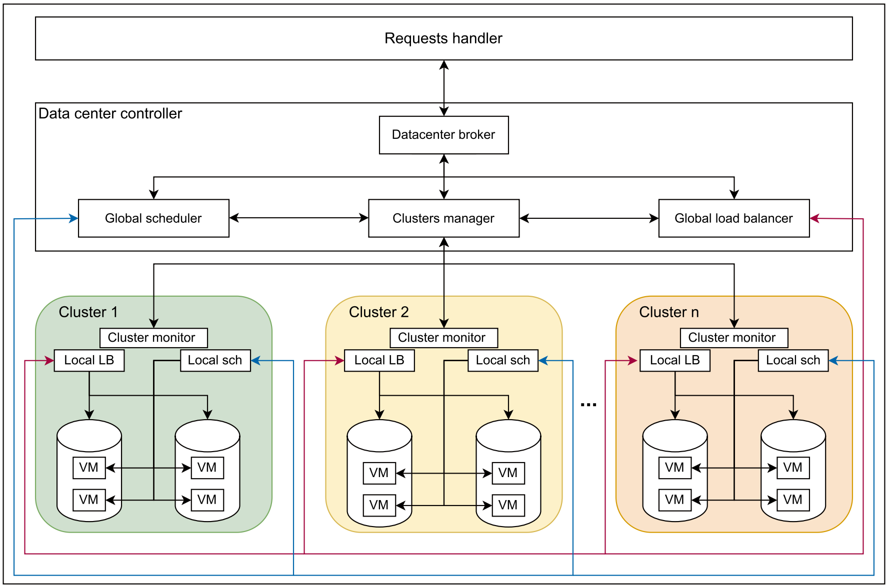

Fig. 2 Our architectural model

Each request is analyzed, modeled and transmitted to datacenter broker in the form of a set of tasks.

- Datacenter broker (Broker): It is the central module responsible of coordination of all other functional components; it receives information, controls consistency, synchronizes and transmits commands to other modules.
- Clusters manager (CM): It is in charge of organizing clusters. In other words, it builds and maintains clusters of servers inside a datacenter while relying on following criteria and primitives:
  - Cluster size: Which is a dynamic parameter but for the first iteration, we randomly fix it to the ratio defined as: $N/50$ where N is the total number of servers in the datacenter. It can take any integer value and experiments are performed to determine its optimal value.
  - Server utilization information: To be able to setup clusters, the module is fed by information on utilization rates of the resources of each server, mainly: CPU, RAM, Bandwidth, storage.
  - Fusion and fission primitives: Dynamic thresholds are chosen to determine when a subset of a cluster should leave it (fission) and create another autonomous cluster. Same mechanism is implemented and determined when and which clusters should fusion to build a larger cluster.
- Cluster monitor: In the assigned cluster, it is responsible of monitoring the evolution of resource utilization of servers. It is recommended to deploy it as a virtual machine in each cluster (same recommendation for other local components) since clusters are dynamic and change over time. It continuously collects data

from servers, aggregates them to statistical metrics and transmits deciding information on which cluster manager relies to trigger fusion and fission primitives.

- Tasks scheduler: It corresponds to the classic module in charge of assigning tasks to the VMs hosted by the servers; however, in our model, it acts on two levels.
  - Level-2 global tasks scheduler (GTS): It decides which of the least loaded clusters will receive the incoming tasks.
  - Level-1 local tasks scheduler (LTS): It is responsible within a cluster for determining which server and virtual machines will run the tasks assigned by the GTS.
- Load balancer: It is responsible of keeping a fair workload distribution among servers in the cloud environment, it is also updated such that it operates within two levels for our purpose:
  - Level-2 global load balancer (GLB): It decides at datacenter level which cluster (among overloaded ones) should be relieved of workload and to which cluster (among underloaded ones) the tasks should be migrated.
  - Level-1 local load balancer (LLB): It selects source servers from the origin cluster for the cloudlets migration process. Sink ones within the destination cluster are selected by local scheduler.

3.4 Overview

Our approach is designed in a such manner to reduce complexity and delays in operations of tasks scheduling and load balancing. To meet this objective, we propose to divide the datacenter into a set of clusters partitioned on four categories according to the makespan and utilization rate of the servers composing them.

There are two ways to give an overall view of our solution. The first focuses on the functional model which is depicted by Fig. 3 and can be broken down as follows:

1. Clustering: In a first phase, we use a k-means-based clustering procedure to divide the servers into four major categories according to their respective utilization rates and makespans. Once this is done, we divide the categories into clusters of bounded sizes. Primitives that we have called fission and fusion allow clusters to evolve in a quasi-cellular fashion, so that certain sub-groups of servers can leave a cluster if they approach the centroid of another category. Two clusters can also be merged if they are in the same category and meet particular size constraints.
2. Tasks scheduling: The second stage is dedicated to job scheduling, with the module acting on two levels: (i) At datacenter level, a round-robin procedure is used to decide which cluster a group of jobs will be assigned to. Then, (ii) at cluster level, a genetic algorithm is used to assign tasks to the servers.
3. Load balancing: Load balancing stage begins as soon as the tasks are scheduled. We have designed our solution in such a way that this mechanism focus on two tasks: (i) identifying the clusters to be lightened and (ii) locating the servers to be freed. Once done, the reallocation of released tasks is left to the scheduling module.

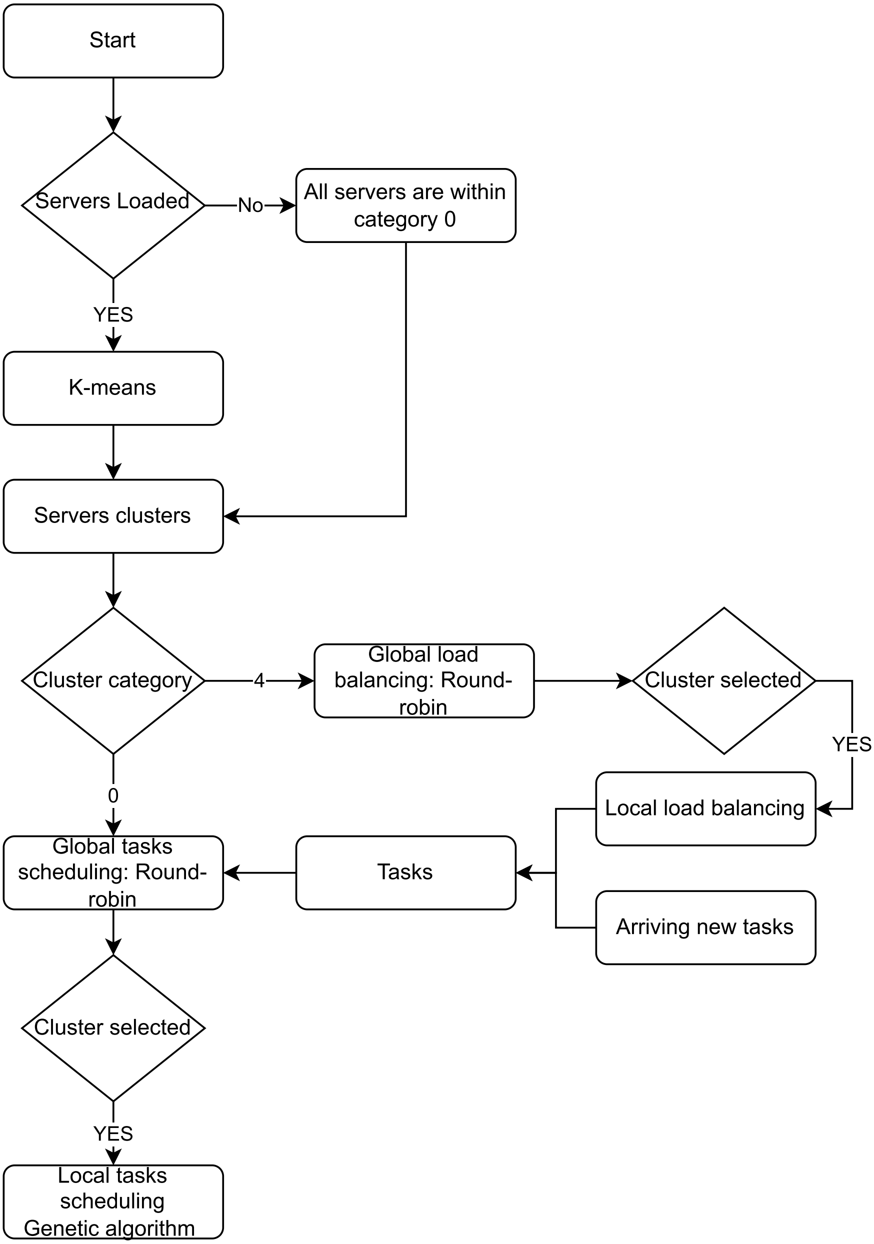

Fig. 3 Functional model flowchart

The second perspective focus on architectural levels on which the mechanisms act and can be depicted as follows:

1. At datacenter level: The mechanisms at this level deal with clusters and are responsible of realizing their respective missions in an independent way while relying on their local modules.

    (a) Cluster manager: It is responsible for cluster creation and development. During hot-deployment, it initiates the k-means-based clustering procedure. It then supervises cluster evolution by gathering information from local monitors and decides on fission and fusion operations.
    (b) Global tasks scheduler: It is in charge of executing a round-robin procedure between clusters to decide which will receive the next jobs. It groups tasks into groups of size equal to the standard number of servers in a cluster, then decides to which cluster to send them to.
    (c) Global load balancer: It determines the overloaded clusters to be released by applying a round-robin algorithm between clusters in the fourth category, if any, and those in third category, if none exist.

2. At cluster level: The modules here act inside clusters and are in charge of realizing their missions on servers while following instructions from global managers and feeding them with local information.

    (a) Cluster monitor: It is on the lookout for evolving information on its own cluster like servers load and cluster size. It is continuously observing the movement of the cluster center and its proximity to main categories centroids. It is responsible for sending alerts regarding the Eulerian distance to the cluster manager when a fission or fusion procedure needs to be triggered.
    (b) Local tasks scheduler: It receives groups of jobs to schedule from the global module and applies a genetic algorithm to decide of tasks assignment over the servers within the same cluster.
    (c) Local load balancer: When called upon from the global load balancer, it runs a particular function to calculate a score based on makespan and utilization rate per server. It then rely on these scores to decide which cloudlets should be retrieved from servers and migrated to another cluster.

This is just an overview of how the load balancing and tasks scheduling mechanisms work, depending on the architectural organization of the environment. The following sub-section is dedicated to the details of each step.

3.5 Our method

Our method goes through succession of steps and involves multiple algorithms; we will detail each of them in this section.

3.5.1 Servers clustering

The power of our method lies in the fact that it reduces the amount of information and constraints the task scheduling and load balancing modules should deal with. To do this, it starts by decomposing the entire datacenter into clusters making the above mechanisms act on size-reduced sets of hosts. We will first focus on how clusters are built in the datacenter level then zoom to clusters level and explain fusion and fission primitives.

Intuitively, we can conceive a classification of servers according to the criteria of makespan length and scores relative to resources utilization rate; those criteria were previously calculated by the formulas 8 and 15 and allow to get the following categories as shown in Fig. 4:

- Category 1: grouping servers with short makespan and low resource utilization rate.
- Category 2: including servers with short makespan and high resource utilization rate.
- Category 3: including servers with long makespan and low resource utilization rate.
- Category 4: grouping servers with long makespan and high resource utilization rate.

Category 1 represents underloaded servers while category 3 represents the worst category since it regroups badly exploited servers and category 4 contains overloaded ones. The category considered to be ideal is category 2 which is made up of

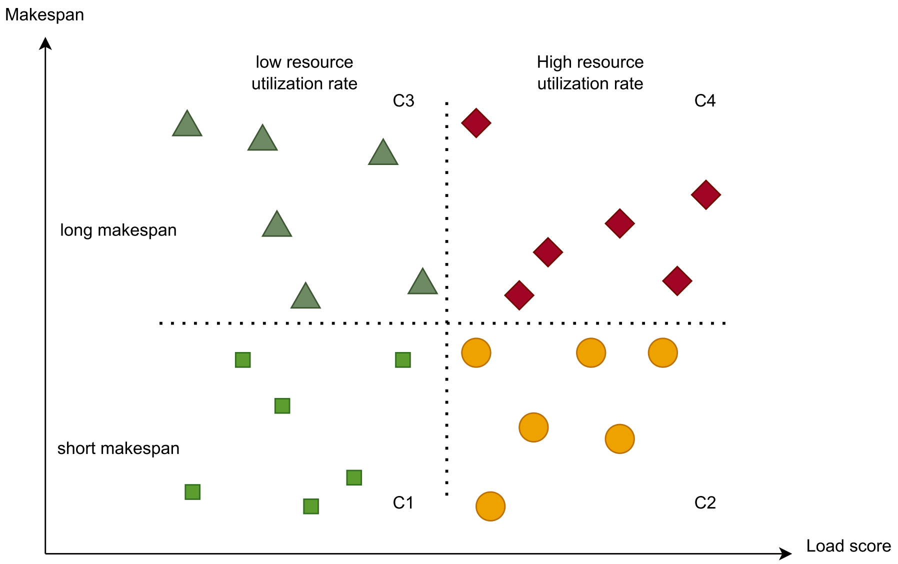

Fig. 4 Server's categories according to makespan and resource utilization score

servers that fairly use their resources and whose makespan is reduced. In the load balancing algorithm proposed in 3.5.3, the migration operations will be made in such manner to bring maximum number of servers within the category 2.

1. Cluster manager: In order to perform this clustering, we propose in Algorithm 1 to use k-means method:

   - First, Algorithm 1 takes as input the list of servers and is expected to return a set of clusters with corresponding servers.
   - In order to realize k-means clustering, informative features on servers must be modeled. We propose the feature representation given in lines 2 and 3, respectively, using Eq. 8 for the first feature which is the makespan of the server. Then, combining Eqs. 14 with 15 for calculating what we call resource utilization rate score to build the second feature.
   - Lines 7 and 8 allow initialization of what we call main centroids coordinates to optimize running time in such a manner that those centroids are initially positioned within the servers features range. Where k is the parameter specifying the number of expected clusters which is four in this case.
   - Line 11 calls a standard k-means clustering procedure. It takes as argument a list of four main centroids generated randomly which should be updated through several iterations by using Euclidean distance and the list of concerned servers. It returns four clusters grouping servers with similar characteristics. This classification around these four main centroids will serve as the basis for our next primitives and algorithms.
   - Finally, line 12 cuts clusters into smaller server pools to render them easier to manage. It takes as input the list of clusters and the desired size which is here equal to 50 and returns a list of clusters of limited size. Local centroids must be recalculated for each cluster, and the CM (cluster manager) keeps track of the initial four main centroids.

2. Local cluster monitor: Once created the CM (cluster manager) setups as a virtual machine on each cluster a cluster monitor which is responsible of gathering information on servers and transmitting them to the CM. The coordinates of the four main centroids obtained from the k-means algorithm are transmitted to each cluster monitor such that it can perform fusion and fission primitives.

   - Fission: It is a function that allows to update clusters so that they stay consistent.

     – When the cluster monitor detects that the third of its server's scores have moved and become closer to one of the other three main centroids than to local centroid, then it informs the cluster manager.
     – The CM initiates a fission operation that produces two clusters.
     – The CM installs a cluster monitor on the new cluster.
     – Obviously, the original cluster updates its local centroid and the new one calculates its own local center.

- Fusion: It is a function that avoids overcrowding of clusters. The cluster manager monitors clusters sizes and under some conditions can trigger clusters fusion primitive:
  - Two or more clusters are near to the same main centroid.
  - Two or more clusters have number of participant servers under a threshold, let it be 25. Tests can be conducted to evaluate its ideal value which allows a better performance of the method.
  - The cardinality of the union of two or more clusters does not exceed a certain threshold, assuming it is set to 75 for our case.

If the datacenter is free of workload at deployment of this framework, then we ignore k-means clustering and split servers into groups of fixed size at the beginning.

Algorithm 1 Servers clustering

```
Require: List of servers
Ensure: Clustered servers
for S_i ∈ Servers do
    makespan(S_i) ← max{makespan(VM_ji)}
    utilization_score(S_i) ← ∑(α · CL_ji + β · RL_ji + γ · SL_ji + σ · BDU_ji)
end for
i ← 0
while i < k do
    centroid[i][X_coordinate] ← random[min(makespan_i), max(makespan_i)]
    centroid[i][Y_coordinate] ← random[min(utilization_rate_score_i),
max(utilization_rate_score_i)]
    i ← i + 1
end while
clusters ← KMeans_clustering(List centroids, List servers)
split_clusters(clusters, 50)
```

Now that we have showed how clusters are built and determined the strategy by which they will be managed; in other words, we have set the laws that govern their evolution. We can move to the task assignment and load balancing algorithms.

3.5.2 Tasks assignment and scheduling

After introducing the methods for creating and managing clusters, we will now present the algorithms for assigning tasks. It is important to remember that we have decomposed the datacenter into a set of clusters in such a way that we will now be able to perform the desired operations at two levels: at the datacenter and clusters one.

1. At datacenter level: At datacenter level, we focus on the method by which tasks will be assigned to clusters. The global scheduler decides on which cluster to plan some arriving tasks regardless to how it will be managed locally by the local tasks scheduler module. To realize this, the global scheduler acts as following:

   - First, it decides of the targeted category of clusters. It will obviously favor clusters of category 1 since they contain underloaded servers, if there is no cluster in this category GTS (global tasks scheduler) will explore the possibilities in class 3. The idea here being to be optimistic, as this category underutilizes its resources, it could be possible to increase this ratio without significantly impacting the makespan. As a last resort, it will choose class 2 which represents the perfect exploitation model and should not be disturbed.
   - Then, it decides of the targeted cluster by using a round-robin algorithm. Once the category is chosen, the global scheduler lists all corresponding clusters first; then, it redirects the tasks it receives on these clusters in turn. Depending on the parameter given to it which is equal to a certain number of tasks, at the end of the scheduling of this number of tasks on the clusters, it will repeat the verification of the first step.

   We propose to start by defining an algorithm for finding clusters corresponding to a specific category relative to the four main centroids. Algorithm 2 explains how we do it.

   Algorithm 3 relies on Algorithm 2 and allows to select the clusters that will be concerned by round-robin tasks assignment procedure.

   A last important procedure must be defined for the round-robin algorithm. Algorithm 2 takes a list of selected clusters and only one task, it assigns the task to one among these clusters and outputs the list of remaining others.

   Now, we have defined all necessary functions, we can introduce the round-robin method as shown in Algorithm 5.
2. At cluster level: The local tasks scheduler operate on cluster level and do not interfere with the global scheduler policy. To perform local tasks, scheduling the concerned module relies on a genetic algorithm. We opted for a genetic algorithm-based approach because, firstly, it is a population-based method and this category offers certain advantages such as the information sharing on the search space among individuals, their ability to avoid local minima and their capability to explore a larger portion of the solution space. Additionally, it is a swarm Intelligence approach, meaning that it preserves the best solution found throughout iterations, possesses fewer operators and has more easily adjustable parameters [36]. While other approaches in the same family, such as PSO (Particle swarm Optimization) and Bee Colony, do exist, genetic algorithms have the advantage of better adaptability to multi-objective problems, a more flexible representation of feasible solutions and a superior balance between exploration and exploitation.

Algorithm 2 Find clusters

Require: List of clusters, category
Ensure: All clusters of a specific category

```
found ← Null
i ← 0
for cluster ∈ clusters do
    if cluster.category = category then
        found[i] ← cluster.id
        i ← i + 1
    end if
end for
return found
```

Algorithm 3 Select clusters

Require: List of clusters
Ensure: Selected clusters for tasks assignment

```
selected_clusters ← find_clusters(clusters, 1)
if selected_clusters = Null then
    selected_clusters ← find_clusters(clusters, 3)
end if
if selected_clusters = Null then
    selected_clusters ← find_clusters(clusters, 2)
end if
if selected_clusters = Null then
    selected_clusters ← find_clusters(clusters, 4)
end if
return selected_clusters
```

Algorithm 4 Assign task

Require: List of clusters, one task
Ensure: Task assigned to one cluster

```
cluster ← random(clusters[0..Length])
schedule(task, cluster)
clusters ← clusters − {cluster}
return clusters
```

Algorithm 5 Round-robin tasks assignment among selected clusters

Require: List of clusters, List of tasks
Ensure: Tasks assigned to clusters
1: $selected\_clusters \leftarrow$ select_clusters(clusters)
2: $task \leftarrow$ random[0..Length]
3: **while** not_empty(clusters) or not_empty(tasks) **do**
4: $selected\_clusters \leftarrow$ assign(selected_clusters, task)
5: $tasks \leftarrow$ tasks - task
6: **end while**

Configuration elements of the algorithm are given in Table 4 where CUR is cluster utilization rate and CMS cluster makespan generated (added) by a particular solution and can be obtained by Eqs. 16 and 17:

$$\mathrm{CUR} = \frac{\sum_1^N \mathrm{UR}_i}{N} \tag{16}$$

$$\mathrm{CMS} = \frac{\sum_1^N \mathrm{makespan}_i}{N * \max\{\mathrm{makespan}_i\}} \tag{17}$$

To perform cluster-level task scheduling, we chose to utilize a variant of genetic algorithms due to their suitability for the nature of the problem, and the positive outcomes observed in related studies. Designing a robust and efficient genetic algorithm entails careful consideration of three key elements: (i) the generation of individuals and the population, (ii) the operators governing population evolution and (iii) the procedure for executing these operations on individuals based on the fitness function.

For the first-generation, we have to create a random population of feasible solutions. We first create one feasible solution at once by generating an individual with Algorithm 6 which incorporates as constraints the number of available servers and arriving tasks and generates a random realizable solution.

Table 4 Configuration of the genetic algorithm for local tasks assignment

Element	Description
Coding	Gene: binary value expressing if the ith server is in charge of a task
	Chromosome/Individual: Vector of N genes where N is the number of servers in the cluster. Each expresses a particular disposition which is a feasible solution
	Population: a set of individuals
Fitness function	$\mathrm{Fitness} = 1 - \frac{\mathrm{CUR+CMS}}{2}$
Parents selection	Elitist, tournament
Crossover	Uniform crossover
Mutation	Randomly made on each gene according to a fixed mutation rate threshold

Algorithm 6 Create an individual

```
Require: Number of tasks, number of servers
Ensure: A feasible solution
individual ← [0, 0, ..., 0]
for i ← 0 to servers.length do
    while count < tasks.length do
        gene ← random{0, 1}
        individual[i] ← gene
        count ← count + 1
    end while
end for
return individual
```

Then, we repeat the procedure for individual creation of a certain desired number of times to create the first generation population, this is explained in Algorithm 7.

Algorithm 7 Population generating algorithm

```
Require: Number of individuals
Ensure: Population of feasible solutions
for i ← 0 to desired_number do
    population[i] ← create_individual(M, N)
end for
return population
```

After generating a random set of individuals to build the first-generation of solutions, we move on now to introduce the operators handling theses chromosomes.

The selection operator described by Algorithm 8 is the one that will allow choosing the most adequate individuals. Those which will be the parents to be combined and to reproduce. We have proposed a tournament selection method which offers the advantage of speed, diversity and to allow the least good individuals to have a chance to participate with better chromosomes in the creation of the new generation. The principle of the method is simple; it randomly selects a certain number of individuals from the entire population and pits them against each other to select the best one.

Algorithm 8 Tournament selection operator

```
Require: Population, number of participants
Ensure: Best individual (fittest)
best ← NULL
for i ← 0 to number_of_participants do
    individual ← random(population[0..Length])
    population ← population - individual
    if individual.fitness() > best.fitness() or best = NULL then
        best ← individual
    end if
end for
return best
```

The first evolutionary operator is the crossover operator described in Algorithm 9. It is evolutionary in the sense that it allows to create a new generation from the original population, this offspring is supposed to be composed of better solutions than the one present in the parents' generation. We decided that for this operator, we would use the uniform crossover approach where each gene of the son is inherited from parent 1 or 2 according to a uniformly distributed probability. This means that a task will be randomly scheduled on a particular server according to its initial assignment within first or second solution according to a certain probability.

Algorithm 9 Crossover evolution operator

```
Require: Population
Ensure: New generation
individuals ← NULL
while not_empty(population) do
    parents[0] ← tournament_selection(population)
    parents[1] ← tournament_selection(population)
    for i ← 0 to parents[0].length do
        choice ← random{0, 1}
        if choice = 0 then
            genes[i] ← parents[0][i]
        else
            genes[i] ← parents[1][i]
        end if
        individuals[j] ← genes
        population ← population - {parents}
        j ← j + 1
    end for
end while
return individuals
```

Now that all the elements that go into the operation of a genetic algorithm are assembled, the overall orchestration is performed by Algorithm 10.

Algorithm 10 Genetic algorithm for in-cluster tasks scheduling

```
Require: List of ordered tasks, lists of servers, number of iterations
Ensure: Best solution as execution planning on servers
populations[0] ← generate_population()
for i ← 0 to number_of_iterations do
    for individual ∈ populations[i] do
        fitness[j] ← individual.fitness()
        j ← j + 1
    end for
    populations[i + 1] ← crossover(populations[i])
    for individual ∈ populations[i+1] do
        individual ← mutate[individual]
    end for
end for
fittest ← NULL
for individual ∈ populations[i] do
    if individual.fitness() > fittest.fitness() or fittest = NULL then
        fittest ← individual
    end if
end for
return fittest
```

3.5.3 Load balancing

The power of our approach lies on two elements: first, clustering which enables management mechanisms to be operated on two levels and therefore to handle a reduced number of entities, whether clusters at datacenter level or servers at cluster level. The second element is the high decoupling in the missions of the latter modules. As we previously said, in load balancing step, we are only interested in one question on two levels: which clusters to free up and which servers in particular to lighten. This is because the load balancer is no longer in charge of migrating cloudlets, as the scheduler will reassign them to other servers on category one clusters.

1. At datacenter level: The first step in load balancing is to locate clusters within the fourth category, those with high resource utilization rate and long makespan.

If the category 4 is free, then we try to find clusters of category three that have a bad utilization rate but still have a long makespan. This can be achieved by using Algorithm 11.

Once found, the clusters of category four or ones of category three if the fourth one is empty should be lighten. A round-robin algorithm described in 12 is again used at datacenter level to ensure fairness between clusters.

2. At cluster level: Once designated by the global load balancer, an instruction is sent to the local load balancer which has to determine which servers must be freed and which cloudlets must be migrated to another cluster. To reach this objective, the load balancer start by calculating the mean of servers makespan noted {MMS} in that cluster determined by Eq. 18:

$$\mathrm{MMS}_{C_i} = \mathrm{Mean}(\mathrm{Makespan}(\mathrm{server}_{ij})) \tag{18}$$

The last step of all in our method is realized thanks to Algorithm 13 which is used by local load balancer to decide which cloudlets must be relieved from servers and migrated to another cluster, the latter job is made by the global tasks scheduler and is out of the scope of this algorithm.

Algorithm 11 SelectcClusters to unload

```
Require: List of clusters
Ensure: Selected clusters to unload
selected_clusters ← find_clusters(clusters, 4)
if selected_clusters = NULL then
    selected_clusters ← find_clusters(clusters, 3)
end if
return selected_clusters
```

Algorithm 12 Round-robin to relieve selected clusters

```
Require: List of clusters
Ensure: Cluster to relieve
selected_clusters ← select_clusters(clusters)
i ← 0
while not_empty(clusters) do
    relieved_cluster ← relieve(selected_clusters[i])
    selected_clusters ← selected_clusters - relieved_cluster
    i ← i + 1
end while
```

Algorithm 13 Determine cloudlets to migrate

Require: List of cloudlets within the cluster
Ensure: List of cloudlets to migrate
for cloudlet $\in$ cloudlets **do**
    $Planned\_time \leftarrow$ cloudlet.getPlannedOnTime()
    if $Planned\_time > MMS$ **then**
        $migration\_list \leftarrow$ migration_list + cloudlet
    end if
end for
return $migration\_list$

Now, we have explained our method in details by depicting each step, the role of each architectural component and the strategies used to trigger primitives, we will move in next section to explain our validation method, show details of implementation, discussing obtained results and comparing them to the best ones found in the field literature.

4 Experiments

In order to validate our method, we implemented it with the standard cloud simulator called CloudSim plus. Table 5 shows what is needed to be implemented in term of objects and corresponding parameters. CloudSim is an open-source simulation library widely used for modeling and evaluating cloud computing systems. It enables researchers and developers to simulate cloud datacenters and applications to assess their performance such as energy consumption, makespan and so on. CloudSim offers a simple, flexible interface for creating customized simulation scenarios. It allows researcher to focus on implementation of their own algorithm like virtual machine placement ones, tasks scheduling, load balancing and to assess them according to a certain set of criteria like number of cloudlets migration [37]. CloudSim Plus comes into the game as an extension to the CloudSim simulation library, offering advanced features and performance enhancements to speed up modeling and simulating cloud computing environments [38].

The simulation was conducted on a laptop with following characteristics:

Table 5 Cloudsim simulation model and elements

Simulation	Datacenter	Hosts	Virtual machine	Cloudlets
Datacenter	List of hosts	CPU	CPU	Length (MIPS)
Broker	VM allocation policy	RAM	RAM	Utilization model
List of hosts		Bandwidth	Bandwidth	
List of VMs		Storage	Storage	
List of cloudlets		Resource provisioner	Cloudlet scheduler	

- Processor: Intel ®Core ™i7-10510U CPU @ 1.80 GHz.
- RAM: 16 GO.
- OS: Windows 11 64 bit, x64-based processor.

In order to verify the veracity of the results obtained, we ran our algorithms in several scenarios, varying key parameters such as the number of servers, cloudlets and cluster sizes. We collected key indicators such as duration over the various stages and number of operations, before carrying out a comparative and analytical study against the methods, we considered to be the most relevant in the literature.

The collected metrics are obtained by calculating the mean values among a hundred of repetitions for each scenario. Cloudlets and servers are generated with random parameters as previously shown in Table 3 to make the scenarios more realistic.

4.1 Experiment results

Table 6 introduces the main performance metrics, we used to evaluate the efficiency of our method and to compare it with other approaches. Since the time involved in load balancing processes is very negligible, we are only interested in three metrics which are response time, number of migrations and of SLA violations.

We have estimated clustering time according to the number of servers within a datacenter and for a variable cluster size, the obtained results are shown in Table 7. In comparison, the [39] approach requires 4 s for clustering one thousand servers. In regard of the parameters used with k-means, our approach enables realizing clustering operation within a very negligible time.

The parameters of the genetic algorithm are provided in Table 8. These parameters were selected after conducting simulations with various random configurations using a grid search method. We chose the configuration that optimized performance parameters such as response time and makespan, while minimizing SLA violations and migrations. This configuration also significantly reduced the algorithm's execution times.

Evaluating task scheduling means taking two durations into consideration: (i) the durations required for the global scheduler to perform group tasks in batches and round-robin execution to designate the target cluster, then (ii) the duration

Table 6 Evaluation metrics

Parameter	Description
Duration	Clusters management duration Clustering Primitives: fission and fusion Tasks scheduling times: Round-robin among clusters and Genetic algorithm inside a cluster
Migrations	Number of cloudlets selected by local load balancer for migration
SLA violations	Number of cloudlets violating the service-level agreement
Makespan	Obtained by Eq. 8
Response time	Refers to the time required by the load balancer to detect an unbalanced situation and to determine cloudlets to migrate

Table 7 Clustering duration

Number of servers	2000				5000				10,000				20,000			
Cluster size	50	100	200	500	50	100	200	500	50	100	200	500	50	100	200	500
Clustering duration	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.03	0.03	0.04	0.04	0.05	0.05	0.05	0.05

Table 8 Genetic algorithm parameters

Parameter	Value
Chromosome size	Equal to cluster size, a value among 50, 100, 200, 500
Population size	50 Individuals
Termination criteria	10 Generation
Crossover rate	0.6
Mutation rate	0.01

required for the local scheduler to run the genetic algorithm and assign tasks to servers. The results obtained are detailed in Table 9. When we focus on overall times, we see that smaller clusters deliver better performance, due to the fact that round-robin time is negligible even when the number of clusters is large, and that the genetic algorithm increases considerably in runtime as the number of servers increases.

The results of Table 9 are represented in the graphs of Fig. 5. The graphs show a certain irregularity: The regression is not totally linear between cluster sizes and planning times. This easily observable phenomenon is due to the fact that task characteristics and server capacities are generated randomly and results are obtained by aggregating the outputs of the various scenarios repetitions in average values.

Table 9 Tasks scheduling duration

Number of servers	5000				10,000			
Cluster size	50	100	200	500	50	100	200	500
Round-robin duration	0.002	0.002	0.002	0.002	0.005	0.005	0.005	0.005
Genetic algorithm duration	0.2	0.31	0.68	3.19	0.47	0.46	2.39	3.5
Total scheduling time	0.202	0.312	0.682	3.192	0.475	0.465	2.395	3.505

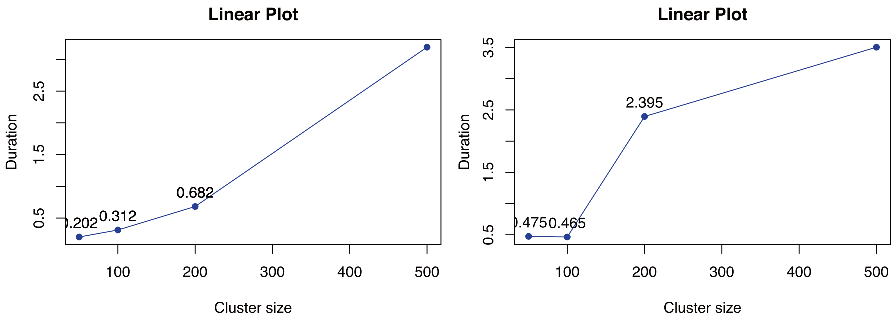

Fig. 5 Tasks scheduling duration according to cluster size

Table 10 Global performance evaluation

Cluster size	50	100	200	500
Migrations	5	13	20	76
Makespan	1.35	1.2	1.5	1.7
SLA violations	5	8	18	35
Response time	0	0.004	0.005	0.009

Table 10 summarizes the most relevant performance criteria of our approach. We have measured those metrics for several possible cluster sizes. The major observations can be summarized as follows:

- The number of cloudlets to migrate increases logically while increasing the cluster size; this is due to the size of the tasks batches transferred by the global scheduler to the cluster which is equal to the number of servers present in the latter. The more the cluster receives tasks that are of variable sizes, the more the load balancer decides to migrate a greater number of them which risk to fall in SLA violations otherwise. It is important to note that the proportion of cloudlets to migrate remains near the ratio of 12% compared to the size of the cluster (batch of tasks).
- The given makespan concerns one batch of tasks of random sizes. We can see that it remains within a very interesting interval of values. Larger sized clusters may have a longer makespan due to the greater variety in the characteristics of the cloudlets that they receive.
- The number of SLA violations increases if the cluster is bigger. The greater the number of tasks received, the greater the probability that some will violate the SLA. It is also important to note that these violations remain below an acceptable threshold of around 8%.
- The last row of the table shows the results of the load balancing module response time evaluation. This time includes the designation of the cluster to be lightened and the precision of the servers to be released. In other words, it encompasses the delays from detecting an imbalance to determining the list of cloudlets to migrate. It is easy to notice that the delays are negligible, in comparison the solution [23] requires 0.008 s to determine the tasks to migrate among a total of just 30.

Table 11 Comparative analysis

	Makespan	Migrations	SLA violations
Our method	1.18	12%	8%
Jena et al. [25]	3.5	19%	Unknown
Babu and Samuel [18]	8	35%	Unknown
Kumar and Sharma [22]	3.8	Unknown	18%

We will now move to the comparative study. We start by comparing the main metrics obtained for our method with some of the most relevant ones in the literature of hybrid approaches. Table 11 gives a comparison between our method and selected other ones according the the standard parameters. The results were estimated for a datacenter of 2000 servers and with 2000 cloudlets. Our method presents a best average makespan, a lower migration ratio and a considerably reduced amount of SLA violations.

Fig. 6 represents the results given in Table 11. The left plot allows a visual comparison of our method and ones given in [25] and [18] according to the number of performed cloudlets migrations. While the right graph compares our method with [22] according to SLA violations. The red graphs are plotted to serve as marks.

The performance evaluation of our methods has produced very promising results, the approach allows an excellent scalability and a reduction in delays, migrations and SLA violations.

4.2 Discussion

The simulations and experiments that we conducted demonstrated that our solution aligns with expectations regarding scalability. Outperforming recent literature, it adapts effectively to highly dynamic environments, which is realistic scenario in light of the rise of the internet of things. This scalability is achieved while maintaining remarkable stability in response times, makespan, task migration count and, most importantly, a consistently low number of SLA violations. The use of the CloudSim Plus simulator, a standard for benchmarking this type of solution, has enabled us to reliably position ourselves in comparison with other proposals. This has even allowed us to identify limitations in existing solutions, whether in their scalability, their inability to ensure hot-deployment, or the lack of validation against both standard and crucial criteria that we have mentioned.

Our solution derives its strength from operating at two levels. By grouping servers based on their loads, we can focus on task scheduling by distributing them over a fixed-size subset of machines. This enables swift cluster selection through round-robin, and, with genetic algorithms operating on a static number of servers, accommodates a multitude of constraints to quickly find an optimal solution. Isolating overloaded server sets through the same k-means-based clustering allows for a rapid assessment of which clusters should be prioritized for offloading and which tasks need to be rescheduled elsewhere. Thanks to this methodology, our approach outperforms the most recent literature, as demonstrated in Table 11 and Fig. 6.

5 Conclusion and perspectives

In this paper, we proposed a new hybrid approach to job scheduling and load balancing in cloud environments. This approach offers several advantages such as hot-deployability, high scalability, decoupling and strong interoperability between the mechanisms that manage the cloud ecosystem. The power of this approach lies in its

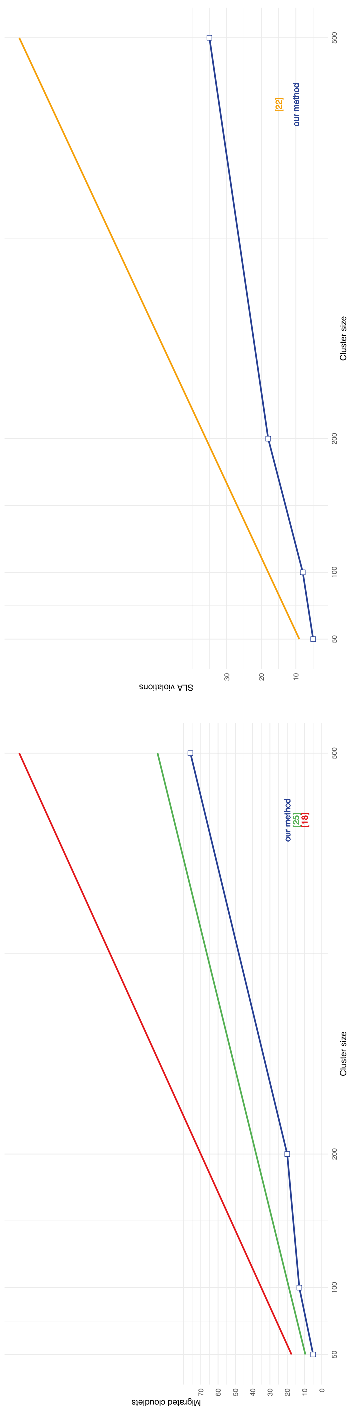

Fig. 6 Migrations and SLA violations

modus operandi: In the first stage, servers are clustered using an algorithm based on k-means and criteria such as utilization rate and makespan, enabling us to approach the problems we face on two scales: at the datacenter level and inside a cluster. The tasks scheduler integrates two modules: one is global and relies on round-robin to transfer task batches to a particular cluster, the other which is local calls a genetic algorithm to assign tasks to servers. The load balancer also operates on these two levels: a global module uses round-robin to designate the cluster to be released, then a particular algorithm which uses an individual score to designate the servers to be unloaded. The clustering mechanism also incorporates probes within each cluster and forecasts their evolution to perform fission and fusion actions designed to maintain cluster coherence.

We were able to validate the performance of our approach by implementing it with CloudSim plus, which produced very conclusive results in terms of makespan, response times, SLA (service-level agreement) violations and cloudlet migrations. The comparative study showed a clear improvement over the most recent and relevant works in the literature. We will now be looking at future improvements, taking advantage of the power of machine and deep learning to optimize cloudlet migration processes and define more optimally the different threshold used to control actions of our algorithms.

Author Contributions Each author participated actively in conducting analyses, drafting sections of the manuscript, editing and approving the final, submitted version.

Funding This work has been sponsored by the General Directorate for Scientific Research and Technological Development, Ministry of Higher Education and Scientific Research DGRSDT, Algeria.

Data availability This declaration is not applicable since we have not used any external resource or dataset.

Declarations

Ethical approval This declaration is not applicable for the purpose of our work.

Competing interests The authors have no competing interests as defined by Springer or other interests that might be perceived to influence the results and/or discussion reported in this paper.

References

1. Gopala M, Sriram K (2022) Edge computing vs. cloud computing: an overview of big data challenges and opportunities for large enterprises
2. Kavitha T, Hemalatha S, Saravanan T, Singh AK, Alam MI, Warshi S (2022) Survey on cloud computing security and scheduling. 1–4
3. Alazzam H, Mardini W, Alsmady A, Enizat A (2019) Load balancing in cloud computing using water flow-like algorithm. In: ACM International Conference Proceeding Series
4. Tawfeeg TM, Yousif A, Hassan A, Alqhtani SM, Hamza R, Bashir MB, Ali A (2022) Cloud dynamic load balancing and reactive fault tolerance techniques: a systematic literature review (SLR). IEEE Access 10:71853–71873

5. Kumar P, Kumar R (2019) Issues and challenges of load balancing techniques in cloud computing: a survey. ACM Comput Surv 51
6. Souravlas S, Anastasiadou SD, Tantalaki N, Katsavounis S (2022) A fair, dynamic load balanced task distribution strategy for heterogeneous cloud platforms based on Markov process modeling. IEEE Access 10:26149–26162
7. Kashikolaei SMG, Hosseinabadi AAR, Saemi B, Shareh MB, Sangaiah AK, Bian GB (2020) An enhancement of task scheduling in cloud computing based on imperialist competitive algorithm and firefly algorithm. J Supercomput 76:6302–6329
8. Qaisar F, Shahab H, Iqbal M, Sargana H, Aqeel M, Qayyum M (2023) Recent trends in cloud computing and IoT platforms for it management and development: a review. Pak J Eng Technol 6:98–105
9. Hong J, Dreibholz T, Schenkel JA, Hu JA (2019) An overview of multi-cloud computing. 1055–1068
10. Kumar M, Sharma SC, Goel A, Singh SP (2019) A comprehensive survey for scheduling techniques in cloud computing. J Netw Comput Appl 143:1–33
11. Rehman AU, Aguiar RL, Barraca JP (2022) Fault-tolerance in the scope of cloud computing. IEEE Access 10:63422–63441
12. Deepa T, Cheelu DD (2017) A comparative study of static and dynamic load balancing algorithms in cloud computing. In: Proceedings of International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS-2017)
13. Kokilavani T, Amalarethinam DIG (2011) Load balanced min–min algorithm for static meta-task scheduling in grid computing. Int J Comput Appl 20:975–8887
14. Liu G, Li J, Xu J (2012) An improved min-min algorithm in cloud computing. AISC 191:47–52
15. Miao Z, Yong P, Mei Y, Quanjun Y, Xu X (2021) A discrete pso-based static load balancing algorithm for distributed simulations in a cloud environment. Future Gener Comput Syst 115:497–516
16. Arulkumar V, Bhalaji N (2021) Performance analysis of nature inspired load balancing algorithm in cloud environment. J Ambient Intell Hum Comput 12:3735–3742
17. Milan ST, Rajabion L, Ranjbar H, Navimipour NJ (2019) Nature inspired meta-heuristic algorithms for solving the load-balancing problem in cloud environments. Comput Oper Res 110:159–187
18. Babu KRR, Samuel P (2016) Enhanced bee colony algorithm for efficient load balancing and scheduling in cloud, vol 424. Springer, Berlin, pp 67–78
19. Adhikari M, Nandy S, Amgoth T (2019) Meta heuristic-based task deployment mechanism for load balancing in IAAS cloud. J Netw Comput Appl 128:64–77
20. Shafiq DA, Jhanjhi NZ, Abdullah A, Alzain MA (2021) A load balancing algorithm for the data centres to optimize cloud computing applications. IEEE Access 9:41731–41744
21. Vanitha M, Marikkannu P (2017) Effective resource utilization in cloud environment through a dynamic well-organized load balancing algorithm for virtual machines. Comput Electric Eng 57:199–208
22. Kumar M, Sharma SC (2018) Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment. Comput Electric Eng 69:395–411
23. Priya V, Kumar CS, Kannan R (2019) Resource scheduling algorithm with load balancing for cloud service provisioning. Appl Soft Comput J 76:416–424
24. Tong Z, Chen H, Deng X, Li K, Li K (2020) A scheduling scheme in the cloud computing environment using deep q-learning. Inf Sci 512:1170–1191
25. Jena UK, Das PK, Kabat MR (2022) Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. J King Saud Univ Comput Inf Sci 34:2332–2342
26. Gamal M, Rizk R, Mahdi H, Elnaghi BE (2019) Osmotic bio-inspired load balancing algorithm in cloud computing. IEEE Access 7:42735–42744
27. Ragmani A, Elomri A, Abghour N, Moussaid K, Rida M (2020) Faco: a hybrid fuzzy ant colony optimization algorithm for virtual machine scheduling in high-performance cloud computing. J Ambient Intell Hum Comput 11:3975–3987
28. Chaudhary D, Kumar B (2019) Cost optimized hybrid genetic-gravitational search algorithm for load scheduling in cloud computing. Appl Soft Comput J 83:10
29. Satpathy A, Addya SK, Turuk AK, Majhi B, Sahoo G (2018) Crow search based virtual machine placement strategy in cloud data centers with live migration. Comput Electric Eng 69:334–350
30. Abohamama AS, Alrahmawy MF, Elsoud MA (2018) Improving the dependability of cloud environment for hosting real time applications. Ain Shams Eng J 9:3335–3346

31. Han H, Bao W, Zhu X, Feng X, Zhou W (2018) Fault-tolerant scheduling for hybrid real-time tasks based on CPB model in cloud. IEEE Access 6:18616–18629
32. Sun H, Yu H, Fan G, Chen L (2020) Qos-aware task placement with fault-tolerance in the edge-cloud. IEEE Access 8:77987–78003
33. Chinnathambi S, Santhanam A, Rajarathinam J, Senthilkumar M (2019) Scheduling and checkpointing optimization algorithm for byzantine fault tolerance in cloud clusters. Clust Comput 22:14637–14650
34. Ghasemi A, Haghighat AT (2020) A multi-objective load balancing algorithm for virtual machine placement in cloud data centers based on machine learning. Computing 102:2049–2072
35. Xu X, Mo R, Dai F, Lin W, Wan S, Dou W (2020) Dynamic resource provisioning with fault tolerance for data-intensive meteorological workflows in cloud. IEEE Trans Ind Inform 16:6172–6181
36. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
37. Goyal T, Singh A, Agrawal A (2012) Cloudsim: simulator for cloud computing infrastructure and modeling. Proc Eng 38:3566–3572
38. 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: Proceedings of the IM 2017–2017 IFIP/IEEE International Symposium on Integrated Network and Service Management
39. Vasile MA, Pop F, Tutueanu RI, Cristea V, Kołodziej J (2015) Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. Future Gener Comput Syst 51:61–71