Cluster Computing (2018) 21:1797–1820
https://doi.org/10.1007/s10586-018-2811-x

A novel task scheduling approach based on dynamic queues and hybrid meta-heuristic algorithms for cloud computing environment

Hicham Ben Alla[1] · Said Ben Alla[1] · Abdellah Touhafi[2] · Abdellah Ezzati[1]

Received: 20 June 2017 / Revised: 15 March 2018 / Accepted: 9 May 2018 / Published online: 25 May 2018

Abstract
Task scheduling is one of the most challenging aspects to improve the overall performance of cloud computing and optimize cloud utilization and Quality of Service (QoS). This paper focuses on Task Scheduling optimization using a novel approach based on Dynamic dispatch Queues (TSDQ) and hybrid meta-heuristic algorithms. We propose two hybrid meta-heuristic algorithms, the first one using Fuzzy Logic with Particle Swarm Optimization algorithm (TSDQ-FLPSO), the second one using Simulated Annealing with Particle Swarm Optimization algorithm (TSDQ-SAPSO). Several experiments have been carried out based on an open source simulator (CloudSim) using synthetic and real data sets from real systems. The experimental results demonstrate the effectiveness of the proposed approach and the optimal results is provided using TSDQ-FLPSO compared to TSDQ-SAPSO and other existing scheduling algorithms especially in a high dimensional problem. The TSDQ-FLPSO algorithm shows a great advantage in terms of waiting time, queue length, makespan, cost, resource utilization, degree of imbalance, and load balancing.

Keywords Task scheduling · Cloud computing · Fuzzy logic · PSO · Dynamic queues · TSDQ-FLPSO · TSDQ-SAPSO

1 Introduction

Cloud computing is an emerging computing paradigm which provides flexible and on-demand infrastructures, platforms and software as services. NIST [1] defines Cloud Computing as a model that refers to the concept of allowing users to request a variety of services like storage, computing power, applications at anytime, anywhere and in any quantity. However, consumers use the services through the cloud service delivery model and do not know where the services are located in the infrastructure [2]. There are usually three models of cloud: SaaS (Software as a Service), PaaS (Platform as a Service) and IaaS (Infrastructure as a Service).

Scheduling is an important concept in cloud computing and has attracted great attention. This process refers to the concept of deciding the distribution of resources between varieties of possible jobs/tasks. In other words, scheduling handles the problem of which resources need to be assigned to the received task. According to the needs of optimal allocation of resources and achieving good quality of service (QoS), these tasks are assigned to the appropriate resources. In cloud computing, independent tasks from different users may need to be handled and scheduled on different virtual machines (VMs). Task scheduling is known as an NP-complete problem due to the different task's characteristics and dynamic nature of heterogeneous resources. In this process, the task scheduler receives the tasks from the users and maps them to the available resources, taking into consideration the task's characteristics and the resource's parameters. Therefore, an efficient/

✉ Said Ben Alla
saidb_05@hotmail.com

Hicham Ben Alla
hich.benalla@gmail.com

Abdellah Touhafi
abdellah.touhafi@vub.ac.be

Abdellah Ezzati
abdezzati@gmail.com

1 LAVETE Laboratory, Mathematics and Computer Science Department, Science and Technical Faculty, Hassan 1 University, 26000 Settat, Morocco

2 Department of Electronics and Informatics (ETRO), Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium

optimal task scheduling algorithm should consider the system load balancing by achieving a good and efficient utilization of resources with maximum profit and a high-performance computing.

Recently, there is a huge interest regarding the use of artificial intelligence techniques. Several meta-heuristic algorithms have been applied to address the challenges of tasks scheduling in cloud computing. Meta-heuristics can be classified into population-based such as the particle swarm optimization (PSO) [3] and trajectory-based such as the simulated annealing (SA) [4]. In this paper, fuzzy logic theory [5], PSO and SA algorithms are used to solve the contribution problem.

Based on an investigation of the key characteristics and properties of the scheduling problem in cloud computing environment, the proposed work can determine good approximate solutions for this complicated problem. In this paper, we focus on task scheduling optimization using a novel approach based on dynamic dispatch queues (TSDQ) and two hybrid meta-heuristic algorithms. We extend and improve our previous work [6] based on fuzzy logic theory with particle swarm optimization algorithm (FLPSO), and we present a new algorithm based on simulated annealing with particle swarm optimization algorithm (SAPSO). We have focused on the concept of hybridizing meta-heuristic algorithms for several reasons: due to the NP complexity of the considered problem, overcoming the inherent limitations of single meta-heuristic, combining the advantages of these algorithms in order to enhance the convergence as well as the effectiveness of the solution space search. Several experiments representing different scenarios were done to compare both the efficiency and the effectiveness of our proposed algorithms with other works from literature. The simulation experiments were conducted with different settings and using synthetic and real data sets from the parallel workloads archive (PWA) [7]. The multi-objectives of the considered performance optimization are as follow: the waiting time, the makespan, the load balancing, the execution cost and the resources utilization. The rest of the paper is organized as follows: in Sect. 2, relevant literatures are briefly described. In Sect. 3, the background of the proposed work is presented. Next, the scheduling problem and the details of the proposed work are described in Sect. 4. The experiment setup and simulation results are discussed in Sect. 5. The paper gives a conclusion in Sect. 6.

2 Related works

Recently, there have been many studies in the literature which have already discussed tasks scheduling in cloud computing is order to achieve and ensure a good performance and maximum utilization of resources on the basis of users and cloud provider requirements. Meta-heuristics have gained huge popularity and has been tried by many researchers to solve the task scheduling problem. Most of the researches in this field try to optimize some objectives that can influence tasks scheduling process such as cost, energy consumption, waiting time, deadline, and makespan, etc. [8–21]. Moreover, queueing theory and multiple queues concept are also applied in many studies and frameworks [22–26]. In the following, we present a brief description of some works.

Many research papers have been published on the tasks scheduling based on PSO algorithm [11, 14, 18]. Guo et al. [11] formulate a model for task scheduling and propose a particle swarm optimization algorithm which is based on small position value rule in order to minimize the cost of the processing. By virtue of comparing the PSO algorithm embedded in crossover and mutation and in the local research with PSO algorithm, the results show the faster convergence of the PSO algorithm in a large scale and prove that is more suitable for cloud computing. In the paper [14], Khalili et al. present a single objective PSO algorithm for workload scheduling in cloud computing. The PSO algorithm is combined with different inertia weight strategies in order to minimize the makespan. The result shows that the PSO combined with linearly decreasing inertia weight (LDIW) achieves better performance and improves the makespan. Al-Olimat et al. [18] introduce a solution for improving the makespan and the utilization of cloud resources through the use of PSO and Random Inertia Weight (RIW). The experimental results illustrate the maximization of resource utilization and minimization of the makespan.

Various studies based on other meta-heuristic techniques such as genetic algorithm (GA), ant colony optimization (ACO) and cat swarm optimization (CSO) have been proposed [13, 17, 19]. Keshk et al. [13] present a cloud task scheduling policy based on ant colony optimization algorithm for load balancing named MACOLB. The main contribution of the proposed work is to balance the system load while trying to minimize the makespan of a given tasks set. The load balancing factor related to the job finishing rate is proposed to improve the ability of the load balancing. In the paper [17], Kaur et al. introduce a modified genetic algorithm for single user jobs in which the fitness is developed to encourage the formation of solutions to minimize the execution time and execution cost. Experimental results show that under heavy loads, the proposed algorithm shows a good performance. Gabi et al. in the paper [19] propose an algorithm called orthogonal Taguchi based-cat swarm optimization (OTB-CSO) to minimize total task execution time. The proposed OTB-CSO explored local search ability of Taguchi optimization method to improve the speed of convergence and the quality of solution by achieving a minimum makespan. The experimental results showed that OTB-CSO is effective to optimize task scheduling and improve overall cloud computing performance.

Some works have tried to improve the scheduling process by using fuzzy logic theory [15, 16]. For example, Nine et al. [15] propose an efficient dynamic fuzzy load balancing algorithm based on fuzzy system. The authors model the imprecise requirements of memory, bandwidth, and disk space through the use of fuzzy logic. Then, an efficient dynamic fuzzy load balancing algorithm is designed which could efficiently predict the virtual machine where the next job will be scheduled. The authors claim that the proposed algorithm outperforms other scheduling algorithms with respect to response time and data center processing time.

Many other techniques have been proposed to improve the scheduling performance [9, 20–22]. For example, Gupta et al. [9] present an efficient task scheduling algorithm for the multi-cloud environment which considers the makespan and resource utilization. This algorithm categorizes the complete set of tasks into three categories (long tasks, medium tasks and small tasks) on the basis of a threshold value which is calculated based on the execution time of the tasks on different clouds. Dinesh et al. [20] introduce a new approach called Scheduling of jobs and Adaptive Resource Provisioning (SHARP) in cloud computing. The SHARP approach embeds multiple criteria decision analysis to preprocess the jobs, multiple attribute job scheduling to prioritize the jobs and adaptive resource provisioning to provide resources dynamically. The authors claim that their approach can mitigate the number of jobs violating their deadline in order to improve user satisfaction. A multi-queue interlacing peak scheduling method is proposed by Zuo et al. [22]. In this method, the tasks are divided into three queues based on CPU intensive, I/O intensive and Memory intensive. The resources are sorted according to CPU utilization loads, I/O wait times, and memory usage. Then, based on these queues, the tasks are scheduled to the resources. The authors declare that their method can balance loads and improve the effects of resource allocation effectively. Zhang et al. [21] present a cloud task scheduling framework based on a two-stage strategy. The proposed work aims at maximizing tasks scheduling performance and minimizing non reasonable tasks allocation in clouds. In the first stage, a job classifier is utilized to classify tasks based on historical scheduling data. In the second stage, tasks are matched with concrete VMs dynamically. Experimental results show that the proposed work can improve the cloud's scheduling performance and achieve the load balancing of cloud resources.

In all the works presented above, authors have proposed different methods and techniques for the scheduling problem in cloud computing in various aspects. However, there is still a pressing need for a flexible architecture and scheduling approach that covers important requirements which may have a great impact on system performance. Moreover, some works do not consider tasks characteristics and resources properties dynamically. Others focus on optimizing the execution time or response time without incorporating other performance measures such as load balancing and resources utilization. Tasks waiting time is an important performance metrics. However, very few works have considered and discussed this parameter. Another issue is identified relating to the fitness function formulation which still needs to consider both tasks and resources features. Furthermore, other methods are single user type, thus, they are not suitable for general cloud applications.

To address the above-mentioned issues, we design an efficient approach based on dynamic queues and metaheuristic algorithms. We propose a new algorithm called waiting time optimization based on PSO (WTO-PSO) to optimize waiting time and decrease the queue length. Also, we introduce the dynamic queue dispatcher which creates multiple queues and dispatch the tasks among them. The decision process to create an optimal number of queues is based on the optimal result obtained in previous stage by WTO-PSO, tasks and resources characteristics. Further, in order to find an optimal mapping of tasks to VMs and optimize the performance metrics, two hybrid algorithms based on Fuzzy Logic and Simulated Annealing with PSO are proposed. The proposed TSDQ-FLPSO and TSDQ-SAPSO algorithms increase the convergence speed of PSO and enhance the solution quality by not only incorporating tasks characteristics and resources capabilities, but also considering dynamic characteristics of the search space. To achieve a good QoS for both cloud users and providers, the proposed algorithms seek to minimize the makespan as well as the execution cost, maximize the resources utilization and provide a good load balancing. Moreover, we have considered evaluating our proposed work using not only synthetic data, but also different real data workloads in order to strengthen the simulation results. Since the workloads can be unpredictable, using different real workloads to do experiments is extremely significant for tasks scheduling problem in cloud computing.

3 Background

3.1 Particle swarm optimization (PSO)

Particle swarm optimization (PSO) algorithm is one of the meta-heuristic algorithms which are used to solve optimization problems and it was successfully used in several single-objective optimization problems. This algorithm was first proposed by Kennedy and Eberhart [3]. PSO is a population based stochastic optimization technique based on the social behaviors of birds flocking or fish schooling. This algorithm consists of a set of potential solutions which evolves to approach a convenient solution (or set of solutions) for a problem. It is used to explore the search space of a given problem to find the settings or the required parameters to maximize a particular objective [27]. An advantage of using the PSO algorithm is that it could be implemented and applied easily to solve various function optimization problems which can be treated as function minimization or maximization problem.

PSO is a computational method that aims at optimizing a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. In this algorithm, the population of the feasible potential solutions of the optimization problem is often called a swarm. The feasible potential solutions are called particles. The key concept of PSO is initialized by creating a group of random particles and then searches for optimum in the problem space by updating generations. We consider that the search space is d-dimensional. During all iterations, each particle is updated by following two "best" values, p_i called personal best (pbest), is the best position achieved so far by particle i and p_g called global best (gbest), is the best value tracked by the particle swarm optimizer, obtained so far by any particle in the population. After finding the two best values, the particle updates its velocity and positions with following Eqs. (1) and (2):

$$v_i^{t+1} = \omega \cdot v_{id}^t + c_1 \cdot r_1 \cdot \left(p_i^t - x_i^t\right) + c_2 \cdot r_2 \cdot \left(p_g^t - x_i^t\right) \quad (1)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (2)$$

where ω is the inertia weight, v_i^t and x_i^t are the component in dimension d of the ith particle velocity and position in iteration t respectively. c_1, c_2 are acceleration coefficients (learning factors). r_1, r_2 are random variables in the [0,1] interval. The particle used to calculate p_g depends on the type of neighborhood selected. In the basic PSO either a global (gbest) or local (lbest) neighborhood is used. In the case of the local neighborhood, neighborhood is only composed by a certain number of particles among the whole population. The local neighborhood of a given particle does not change during the iteration of the algorithm. The parameters ω, c_1 and c_2 must be selected properly to increase and enhance the capabilities of PSO algorithm [28]. The inertia weight is an important parameter for controlling and enhancing the PSO global search capability by providing balance between exploration and exploitation process, which means that the inertia weight parameter can affects the overall performance of the algorithm in finding a potential optimal solution in less computing time. However, numerous strategies have been introduced for adjusting the inertia weight and choosing the proper value, such as chaotic inertia weight (CIW) [29], The linearly decreasing inertia weight strategy (LDIW) [30], random inertia weight (RIW) [31] and fuzzy particle swarm optimization (FPSO) [32].

However, the original PSO version is not designed for discrete function optimization problems but for continuous function optimization. Therefore, in order to solve this issue, a Binary version of PSO (BPSO) algorithm was developed [33]. BPSO implements the decision model of a particle based on discrete decision i.e., "true" or "false", "yes" or "no", etc. The main difference between binary PSO and continuous PSO is that the velocities of particles in binary version are defined as probabilities that a bit take 0 or 1 value. Thus, a velocity must be restricted within the range [0,1]. The logistic sigmoid function of the particle velocity is used as the probability distribution to identify new particle position based on binary values.

The logistic sigmoid function shown in Eq. (3) is used to limit the speed of the particles as the probability stays in the range of [0,1]:

$$S\left(v_i^{t+1}\right) = \frac{1}{1 + e^{-v_i^{t+1}}} \quad (3)$$

The Equation that updates the particle position is given in (4):

$$x_i^{t+1} = \begin{cases} 1 & \textit{if } r_3 S(v_i^{t+1}) \\ 0 & \textit{otherwise} \end{cases} \quad (4)$$

where r$_3$ is a random factor in the [0,1] interval.

3.2 Simulated annealing and random inertia weight

Simulated annealing (SA) is trajectory-based search algorithm that belongs to the field of stochastic optimization and meta-heuristics. It is based on probabilistic methods that avoid being stuck at local optimal solution. This technique is inspired by the process of annealing in metallurgy. SA is based on the idea of searching for feasible solutions and converging to an optimal solution. The strategy called random inertia weight (RIW) proposed in [31] uses the idea of SA and the fitness of the particles to design another inertia weight to improve the global search ability of PSO and to increase the probability of finding a near-optimal solution in fewer iterations and computing time. A cooling temperature was used in SA mechanism to adjust the inertia weight according to certain probability so as to be adapted to the complex condition. The simulated annealing probability is expressed in (5):

$$p = \begin{cases} 1, & \min_{1\le i\le m} f_i^{t-k} \min_{1\le i\le m} f_i^t \\ \exp\left(-\frac{\min_{1\le i\le m} f_i^{t-k} - \min_{1\le i\le m} f_i^t}{T_i}\right), & \min_{1\le i\le m} f_i^{t-k} \min_{1\le i\le m} f_i^t \end{cases} \tag{5}$$

where the adaptive cooling temperature used to jump off the local optimal solution is given in (6):

$$T_t = \frac{f_{avg}^t}{f_{best}^t} - 1 \tag{6}$$

where T_t is the cooling temperature in the tth iteration, f_{avg}^t is the average fitness value in the tth iteration using the formula (7), f_{best}^t is the current best fitness value, m is the number of particles, and f_i^t is fitness value of the i particle in the tth iteration.

$$f_{avg}^t = \frac{1}{m}\sum_{i=1}^{m} f_i^t \tag{7}$$

The inertia weight is defined in Eq. (8):

$$\omega_t = \begin{cases} \alpha_1 + \frac{r}{2.0}, & p \ge r \\ \alpha_2 + \frac{r}{2.0}, & p < r \end{cases} \tag{8}$$

where r is random number in range [0,1], α_1, α_2 are constants in range [0,1], with $\alpha_1 > \alpha_2$. The RIW combined with SA method is used in this work, as it can achieve best convergence velocity and precision, and can help to keep swarm variety.

3.3 Fuzzy logic theory

Fuzzy logic is a powerful mathematical tool to deal with uncertainty and imprecision. Fuzzy Logic attempts to solve problems by considering all available information and making the best possible decision given the input. The fuzzy logic theory is often applied by advanced trading models/systems that are designed to react to the changing markets. The fuzzy controller uses a form of quantification of imprecise information or input fuzzy sets to generate an inference scheme, which is based on a knowledge base of control, a precise control force to be applied on the system. The advantage of using the quantification is that the fuzzy sets can be represented by linguistic expressions such as low, medium or high. The linguistic expression of fuzzy sets is known as term, and a collection of such terms defines a library of fuzzy sets. Therefore, a fuzzy controller converts the linguistic control strategy typically based on expert knowledge into an automatic control strategy. The logical controller has three main components as shown in Fig. 1: (i) a Fuzzifier component where the information (crisp set of input data) is quantified and converted to a fuzzy set using fuzzy linguistic variables, fuzzy linguistic terms and membership functions [34]. (ii) a fuzzy inference engines component which converts the input fuzzy sets into control force fuzzy sets through rules collected in the knowledge base and aggregates the resulting fuzzy sets and (iii) a Defuzzifier component where the output fuzzy information is converted into a precise value.

4 Proposed architecture

4.1 Scheduling problem

Nowadays, much attention has been devoted to the task scheduling as an interesting aspect of cloud computing, due to the big overlap between the cloud provider requirements and user preferences such as the quality of service, the priority of users, the cost of services, etc. Thus, the user's satisfaction, optimization of profit for the providers and other factors should be taken into consideration when scheduling the tasks. Therefore, as this process is NP-complete problem, an efficient approach should not only optimize some objectives of cloud user and provider, but also take into consideration the dynamic characteristics of cloud computing environment.

Cloud computing is composed of a large number of datacenters containing multiple physical machine (Host). Each host runs several virtual machines (VMs) which are responsible for executing user's tasks with different QoS. The scheduling process in cloud computing environment is depicted in Fig. 2. Suppose that N users submit their tasks i.e., $T_1, T_2, \ldots, T_N$ to be scheduled to M Virtual Machines i.e., $VM_1, VM_2, \ldots, VM_M$. We assume that tasks are mutually independent i.e., there is no precedence constraint between tasks, they are not pre-emptive and they cannot be interrupted. Moreover, these tasks have different characteristics such as length, arrival time, burst time, deadline, etc. We assume also that VMs are heterogeneous in terms of CPU speed, RAM, bandwidth, etc. The cloud broker

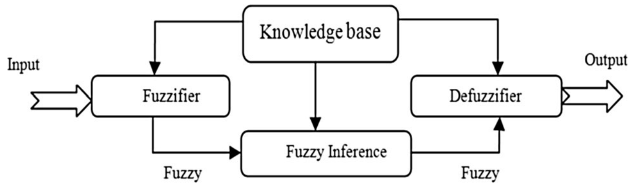

Fig. 1 Fuzzy logic system architecture

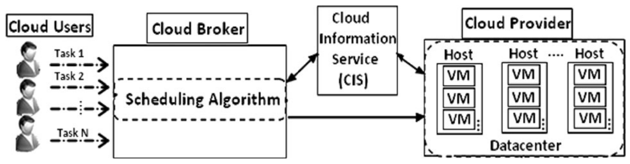

Fig. 2 Cloud scheduling environment

queries the cloud information service (CIS) to provide information about the services required to execute the received tasks, then schedule the tasks on the discovered services. The choice of the tasks to be served is determined by multiple factors and QoS requirements assured by the broker. The cloud broker is the main component of tasks scheduling process, which mediates negotiations between user and provider and is responsible also for making scheduling decisions of task to the specific and particular resource. However, some issues need to be taken into account. Firstly, the tasks submitted by users join the first queue in the system and have to wait while the resources are used. Consequently, this extends the queue length of the system and increases the waiting time. However, this queue has to be managed with an efficient method rather than First Come First Served (FCFS) policy. Secondly, when the provider handles the tasks, many parameters can be considered as single objective optimization or multi-objective simultaneously such as the makespan that has a direct effect on utilization of resources. Therefore, a good tasks scheduling approach should be designed and implemented in the cloud broker to not only satisfy the QoS constraints imposed by cloud users, but also perform good load balancing among virtual machines in order to improve the resources utilization and maximize the profit earned by the cloud provider. Such approach should also have the adaptability to adjust the scheduling process according to dynamic scheduling requirements in cloud environment. Based on the issues mentioned above, the proposed approach is designed to optimize specific performance metrics. Particularly, our goal is to minimize the makespan, minimize the waiting time, maximize the resources utilization, achieve better load balancing and reduce the cost of demanded resources. A detailed description of these parameters is given in the following subsection.

4.2 Measures of effectiveness

In the process of task scheduling, various parameters can be considered as performance metrics such as makespan, cost, flowtime, waiting time, tardiness, turnaround time, throughput, load balancing, resource utilization, etc. In this paper, the optimized performance metrics are defined as follows:

(i) *Makespan* Indicate the time spent for executing all tasks (i.e.: the finishing time of the last task). This metric can be calculated by Eq. (9):

$$\mathrm{Makespan} = \max_{\mathrm{i} \in \mathrm{tasks}}\{\mathrm{FT_i}\} \tag{9}$$

where $\mathrm{FT_i}$ denotes the finishing time of task *i* [35].

(ii) *Waiting time* In this paper, the waiting time of task refers to the time spent in the queue before execution. The minimum value of waiting time indicates the right/optimal order of the tasks that can minimize the waiting time as well as the queue length.

(iii) *Degree of Imbalance (DI)* This metric measures the imbalance among VMs. The Degree of Imbalance metric represents an important QoS metric to show the distribution efficiency of tasks and load balancing among the virtual machines. DI is calculated using the following Eq. (10):

$$DI = \frac{T_{\max} - T_{\min}}{T_{avg}} \tag{10}$$

where $T_{\max}$, $T_{\min}$, T_{avg} are the maximum, minimum and average total execution time of all VMs respectively [36].

(iv) Cost: This metric is calculated using the following formula (11):

$$\mathrm{Cost} = \sum ET_{ij} \times C_{VM_j} \tag{11}$$

where ET_{ij} is the execution time of the task *i* on VM_j and C_{VM_j} is the cost of VM_j per unit time.

(v) *Resource utilization (RU)* The resource utilization can be calculated by Eq. (12):

$$Average\ resource\ utilization = \frac{\sum_{i=1}^{N} T_{VM_i}}{Makespan \times N} \tag{12}$$

where $\mathrm{T_{VM_i}}$ is the time taken by the $\mathrm{VM_i}$ to finish all tasks, and N is the number of resources [35].

4.3 Proposed approach

Cloud computing environment has seen a rapid and enormous growth over the last few years. Notwithstanding, task scheduling is considered as one of the challenges which have to be solved. However, an efficient approach of tasks scheduling remains a long-standing problem in terms of the task's characteristics, the cloud provider's requirements, user's preferences and the type of optimization problem to be solved. In general, the proposed approach stems from the need to incorporate the solutions of these issues in simplified design architecture. In fact, the motivation of our work is the following: Firstly, tasks waiting time and waiting queue length are important indicators of quality of services offered by cloud providers; therefore, it is very important to consider these constraints when designing a scheduling algorithm. However, traditional FCFS strategy completely ignores these indicators. To overcome this issue, a Waiting Time Optimization algorithm based on PSO (WTO-PSO) is proposed. It aims at finding an optimal solutions (optimal order) which can effectively minimize the tasks waiting time and reduce the queue length. Secondly, we introduce a technique to dispatch the tasks among dynamic queues. Dynamic queue dispatcher is designed to analyze different characteristics of the search space problem under consideration and to create an optimal number of queues which can improve the ability of TSDQ-SAPSO and TSDQ-FLPSO algorithms in the next step to schedule tasks efficiently. Dynamic dispatch queues algorithm makes its decision to create an optimal number of queues and dispatch the tasks among them based on the waiting time result, task lengths and resource computing capabilities. Finally, two hybrid meta-heuristic algorithms are proposed in order to find an optimal mapping of tasks to resources and optimize the performance metrics under consideration such as minimize the makespan, achieve a good load balancing, reduce the execution cost and increase the resources utilization. The proposed algorithm combines the advantages of Fuzzy Logic and SA with PSO algorithm. The idea was to use different algorithms to enhance the convergence performance of PSO and to improve the search ability as well as the solution quality. Based on the observation that inertia weight is a crucial parameter that affects the performance of PSO significantly, both proposed algorithms adjust this parameter dynamically in order to improve the search ability of PSO. Moreover, after analyzing dynamic characteristics of the problem in hand, the proposed solution based on fuzzy model incorporates also tasks characteristics and resources capacities. The main difference between these hybrid algorithms lies in the way to adjust inertia weight and in the fitness calculation model. Therefore, in TSDQ-SAPSO algorithm, we adjusted the inertia weight parameter using SA algorithm. However, in TSDQ-FLPSO algorithm, Fuzzy Logic is applied not only to adjust the inertia weight, but also to calculate fuzzy fitness which is modeled to consider both tasks length and resources features such as CPU speed, RAM memory, bandwidth and status of the VM.

The main purpose of our contribution is to satisfy the performance requirements, enhance the cloud provider's profit and provide a good QoS for users. Therefore, we solve the task waiting time problem by managing the task waiting queue using a meta-heuristic algorithm rather than basic policy such as FCFS which completely ignores waiting time and queue length. FCFS schedules the tasks according to their arrival time and does not consider any other factors that may have great impacts on system performance. As a result, this will create a long queue in the system which may be an indicator of slow service. Moreover, this can have a negative impact on user's satisfaction especially when the estimated waiting time or the queue length is too long. Thus, we use an efficient algorithm based on PSO algorithm to optimize the average waiting time and to keep the task waiting queue in the system as short as possible. As shown in Fig. 3, in the Task Waiting Queue, the tasks are queued based on their arrival time. Then, the waiting time optimization WTO-PSO algorithm is applied to manage the batch of tasks $T_1, T_2, \ldots, T_n$ stored in this queue. The proposed algorithm jointly considers two issues to optimize the waiting time and reduce the queue length. Therefore, it starts to calculate the waiting times of all possible tasks sequences to get the best or optimal sequence, then, return the minimum value, which refers to the right order of the tasks that can minimize the waiting time and reduce the queue length.

To optimize the waiting time, Eq. (13) is defined as the objective function of this problem. Therefore, the fitness function used to calculate the solution of each particle in PSO algorithm, and examine the solutions to find an optimal solution for the problem under consideration is given in Eqs. (14) and (15).

$$\textit{objective function} = \text{minimize}\,\{\text{Waiting Time}\} \tag{13}$$

$$Fitness = MinT_{Wt} \tag{14}$$

where the total waiting time of tasks T_{WT} can be expressed as follow:

$$T_{WT} = \frac{1}{n}\sum_{i=1}^{n} WT_{Task\,i} \tag{15}$$

where $WT_{Task\,i}$ is the waiting time of task i and n is the number of tasks in the queue.

When the minimum value is returned by WTO-PSO algorithm, we ultimately get the best/optimal order of tasks. Next, based on this result, the dynamic dispatch queues algorithm TSDQ which manages the tasks automatically is applied. The main goal of the TSDQ algorithm is to create an optimal number of queues and dispatch the received tasks to these queues based on an efficient

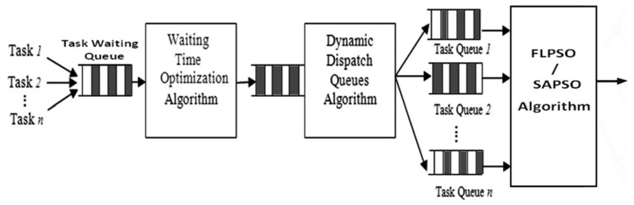

Fig. 3 The proposed approach model

strategy. TSDQ algorithm calculates the best threshold value and starts to calculate the sum of task length until the threshold is reached, then create a new queue and dispatch the appropriate tasks to this queue. Next, it restarts the calculus again until it dispatches all arrived tasks. Consequently, applying the TSDQ can create dynamic queues on the basis of a decision threshold.

The pseudo code of the proposed work is presented as Algorithm 1:

decided to develop an intelligent algorithm that integrates the concept of queueing model which is increasingly used for the analysis and design of complex service systems. The proposed strategies based on dynamic queues model is designed to be able to analyze different characteristics of the search space problem under consideration. Moreover, it can change dynamically the numbers of needed queues and the number of tasks they stored on the basis of tasks characteristics and resources requirements and availability.

Algorithm 1. Pseudo-code of TSDQ-SAPSO/FLPSO Algorithm

```
1.  create list of received tasks {Ltask} and determine all required parameters
2.  create list of VMs {LVMs} and determine all required parameters
3.  all tasks {Ltasks} are at first stored into Tasks Waiting Queue
4.  Waiting Time Optimization based on PSO (WTO-PSO) is applied to get the
    minimum waiting time. {Fitness solution of PSO: Equations (14,15) }
5.  return the best obtained sequence (minimum waiting time) as Ltask'
6.  initialize BestDecisionTreshold
7.  for each task in Ltask'   do
8.          Calculate the Decision Distribution Threshold ({Ltask'},{LVMs})
9.  end for
10. Return the best Decision Threshold as BestDecisionTreshold
11. Create an empty queue Q;
12. for each task in {Ltask'}   do
13.         if  SumTaskLenght(task) < bestDecisionTreshold   then
14.                 put the task into the queue Q
15.         else
16.                 create the next queue Q
17.                 initialize SumTaskLenght
18.         end if
19. end for
20. return list {LQueue:$Q_1$,..$Q_N$} of all created queues
21. for each queue Q in {LQueue}    do
22.         FLPSO or SAPSO algorithm is applied to get the best fitness solution.
23.         keep track of the best solution
24. end for
```

In the cloud, the resources and users requests can change dynamically. Therefore, a scheduling algorithm should be smart enough to make real-time responses to a changing environment. Consequently, there is a need to develop an approach in order to respond correctly to the QoS requirements. Building upon these observations, we Thus, such representation allows not only to minimize the search space, but also to partition it in order to improve the search ability of the proposed TSDQ-SAPSO and TSDQ-FLPSO algorithms in high-dimensional problem. In other words, dynamic dispatch queues algorithm makes decisions in the best possible ways to create the optimal number of queues which can guarantee to schedule the tasks efficiently. For this reason, it exerts an intelligent strategy so as to enforce that the optimization waiting time result (the optimal sequence order which can reduce the waiting time and decrease the queue length) is used. The importance of this step when dispatching the tasks among dynamic queues comes from the reason that the waiting time of some tasks may be too long in these queues, especially for those coming first, because their arrival order is changed. Consequently, their completion time could be significantly increased, which has negative impact on the system's performance and QoS constraints. Thus, to overcome this problem, the decision made by dynamic queue dispatcher takes into account also the obtained waiting time result. This strategy helps to avoid performance degradation when the number of tasks scales up. In addition, it gives the proposed hybrid algorithms the flexibility to analyze and explore the tasks set stored in these queues in order to find the best mapping of tasks to the computational resources.

After the TSDQ is finished, FLPSO or SAPSO algorithm selects each queue and schedules the tasks to the appropriate resources. In fact, an efficient process of tasks scheduling requires an optimal/efficient algorithm that takes into consideration the tasks properties and resources configuration. Therefore, the main objective of these algorithms is to assign the most suitable resources for tasks based on the computational capabilities of the resources and the tasks characteristics. Our proposed TSDQ-SAPSO and TSDQ-FLPSO algorithms use PSO as the main search algorithm, while SA and Fuzzy Logic are used to improve the fitness value and promote the convergence rate. There are some reasons for using these algorithms. First, we need an algorithm that is based on a population that can search the entire cloud space for the problem in hand. Second, the scheduling algorithm must be fast enough to adapt with dynamic characteristics of cloud and must be able to converge faster than other algorithms. Third, due to complex structure of search space and critical role of inertia weight in the convergence behavior of PSO, the standard PSO is not performing well. That is why we combined PSO with SA and Fuzzy Logic algorithms that are strong in local searches and used practical rules to address these challenges. The two hybrid algorithms TSDQ-FLPSO and TSDQ-SAPSO are used to minimize the makespan, minimize the execution cost, achieve a good load balancing and a high utilization of resources. The details of our algorithms used in this paper are described in the next subsections: in Sect. 4.3.1, we describe the SAPSO algorithm based on Simulated Annealing and PSO, and in Sect. 4.3.2 we explain the FLSPO algorithm based on Fuzzy Logic and PSO.

4.3.1 SAPSO algorithm: simulated annealing and PSO algorithms

In the following, we propose a new algorithm to address the scheduling problem. For this, we develop a hybrid algorithm named SAPSO based on PSO algorithm combined with SA (Algorithm 2 and 3) to achieve the best convergence speed of PSO algorithm. The Inertia weight is one of the most critical factors affecting the convergence performance of PSO. Therefore, SA algorithm is used to adjust this factor based on certain probability to facilitate jumping off local optimal solutions. In other words, SA employs certain probability to avoid becoming trapped in a local optimal and the search process can be controlled by the cooling schedule. Thus, the hybrid algorithm SAPSO combines the advantages of both algorithms to improve the robustness and accuracy. In the proposed algorithm, an initial swarm of N particles is randomly generated. Each particle represents a possible solution for the task scheduling problem. So, according to collective experiences, the particle vectors will be iteratively modified in order to improve their solution quality [37]. According to a particular objective function, the group of particles moves in the given problem space to search for an optimum solution. The main objective in our case is to minimize the makespan using Eq. (16). Therefore, the fitness value of each particle defined in Eqs. (17) and (18) are used to calculate the executions times of all possible tasks sequences on each cloud resource.

$$\begin{aligned} \mathit{Objective\,function} &= \text{minimize}\,\{\text{Makespan}\} \\ &= \min\left[\mathit{Max}\left\{\varphi_{vm_1},\varphi_{vm_2},\ldots,\varphi_{vm_j},\ldots,\varphi_{vm_m}\right\}\right] \end{aligned} \tag{16}$$

$$\begin{aligned} \mathit{Fitness} &= \mathit{Max}\left\{\varphi_{vm_1},\ldots,\varphi_{vm_m}\right\} \\ &\quad \mathit{with}\ \varphi_{vm_j} = \sum_{i=0}^{n}\delta_j(\mathit{task}_i) \end{aligned} \tag{17}$$

$$\begin{aligned} \delta_j(\mathit{task}_i) &= \frac{L(\mathit{task}_i)}{\mathit{npe}_j \times \mathit{Vmips}_j} \\ &\forall i = \{1,2,\ldots,n\},\ j = \{1,2,\ldots,m\} \end{aligned} \tag{18}$$

where is the total execution time of a set of tasks running on vm_j, $\delta_j(task_i)$ is the execution time of task i on vm_j, n is the number of tasks and m is the number of VM, $L(task_i)$ is the length of a task in million instruction (MI), npe_j is the number of processing elements and $Vmips_j$ is the VM speeds in million instructions per second (MIPS).

The pseudo-code of SAPSO is described in Algorithm 2.

This algorithm starts with initialization part which consists of generating the swarm with random position and velocity. Then successively updating velocity, position, local best and global best solution in each iteration. The process ends when the termination criteria are met: either the number of iterations is exceeded or the optimal solution is obtained. We used the SA algorithm for optimizing the inertia weight to speed up the convergence of PSO towards the optimal solution.

The pseudo-code of simulated annealing method is shown as follow:

Algorithm 2. Pseudo-code of SAPSO Algorithm

```
1.  Initialize particles with random position and velocity
2.  repeat
3.      for each particle    do
4.          calculate the inertia weight using SA combined with RIW(Algorithm3)
5.          calculate the fitness value using Equation (17)
6.          if    the fitness value is better than the best fitness
                  value (pbest-old)    then
7.                set current value as the new pbest
8.          end if
9.      end for
10.     select the particle with the best fitness value of all the particles as gbest
11.     for each    particle    do
12.         update particle velocity using Equation(1)
13.         update particle position using Equation(4)
14.     end for
15. until termination criterion is met
16. output the best solution obtained
```

Algorithm 3. Pseudo-code of Simulated Annealing (SA)

```
1.  S0 = Generate_initial_solution()
2.  Sbest = S0
3.  Temperature = Initialize_Temp()
4.  repeat
5.      repeat
6.          Temperature = calculate_temperature()
7.          Snew = Create_neighbor_solution(S0)
8.          if (Fitness(Snew) < Fitness(S0))    then
9.              Sbest = Snew
10.         else if (rand() < $e^{\frac{(Fitness(S_0)-Fitness(S_{new}))}{Temp}}$ )
11.             S0 = Snew
12.         end if
13.     until thermal equilibrium is reached
14.     decrease Temperature according with the annealing schedule;
15. until stopping criterion is met
16. return  Sbest
```

In this process, an initial *S0* is generated and the temperature is set at initial *Temperature*. Iterations are performed until a certain stopping criterion is met. In each step, SA considers some neighboring solution *Snew* of the current solution *S0* and decides between moving to *Snew* or staying in *S0* with some probability. The new solution (*Snew*) will be accepted if it has a better fitness compared to the current solution (*S0*). However, if the new solution is worse than the current one, it will be accepted with the probability showed in line 10 of the pseudo code. Once thermal equilibrium is reached, the temperature is reduced according with the annealing schedule. The main advantage of the simulated annealing algorithm is its strong local search ability to avoid falling into local optimal solution.

4.3.2 FLPSO algorithm: fuzzy Logic and PSO algorithm

In this section, the proposed algorithm FLPSO based on fuzzy logic with PSO algorithm is discussed. Fuzzy logic employs linguistic terms which deal with the causal relationship between input and output variables. Hence, the approach makes it easier to handle and solve problems. Therefore, our proposed algorithm is designed to adjust dynamically the inertia weight and to enhance the fitness function value using a mechanism based on fuzzy rules. The fuzzy logic theory is used in two stages: Firstly, to calculate the inertia weight parameter. A suitable value for inertia weight provides the desired balance between the global and local exploration ability of the swarm and, consequently, improves the effectiveness of FLPSO algorithm. Secondly, to calculate the fitness value of each particle used in FLPSO algorithm to get a better chance to find an optimal solution in a reasonable number of iterations. Accordingly, using Fuzzy Logic controller aids FLPSO to converge to the optimum solution based on the fitness value defined by combining the satisfaction of task requirements and VM capabilities. The flowchart of FLPSO algorithm is described in Fig. 4.

Using FLPSO algorithm, we seek to assign the most suitable resources to tasks based on the computational capabilities of the resources and the tasks characteristics, taking into account the performance metrics. Thus, we need an intelligent algorithm which finds the best solution based not only on the search space structure, but also on the dynamic tasks and resources characteristics. Therefore, fuzzy logic is applied to calculate the inertia weight w as an output based on iteration parameter input. The inertia weight was adaptively adjusted to increase the convergence speed and accuracy of our algorithm. Moreover, Fuzzy Logic is used to calculate the fitness value of particles

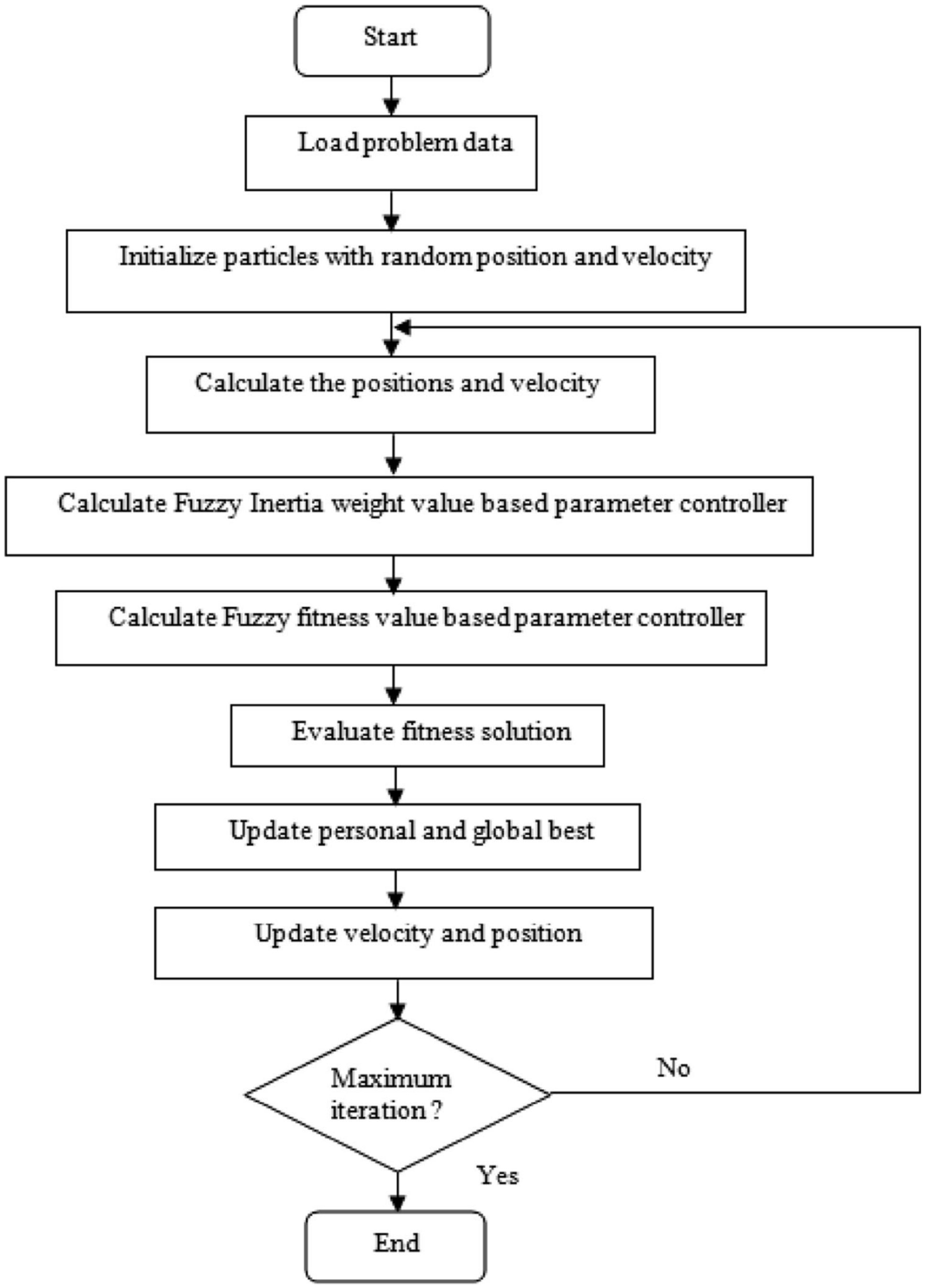
Start
Load problem data
Initialize particles with random position and velocity
Calculate the positions and velocity
Calculate Fuzzy Inertia weight value based parameter controller
Calculate Fuzzy fitness value based parameter controller
Evaluate fitness solution
Update personal and global best
Update velocity and position
Maximum iteration ?
No
Yes
End

Fig. 4 Flowchart of FLPSO algorithm

based on the following input parameters: task length, RAM memory, CPU speed and status of the resource (i.e., the occupancy rate of the VM). Therefore, the objective function is to minimize the makespan which means to get the best solution of all fitness values calculated by Fuzzy Logic for all possible tasks sequences on cloud resources. This means to find the most suitable resources to process the tasks.

To get the output value of fuzzy logic, the fuzzy inference converts the input fuzzy sets into control force fuzzy sets, through rules collected in the knowledge base.

In this paper, we use Mamdani [38] inference system to derive the fuzzy outputs from the inputs fuzzy sets according to the relation defined through fuzzy rules. We construct the fuzzy inference system (FIS) for calculating the inertia weight value with the rules shown in Table 1. For the fitness value, the FIS is constructed with the rules shown in Table 2.

The Fig. 5(a) and (b) show the fuzzy sets for the task length and CPU speed of VM parameters. The Fig. 6(a) and (b) show the fuzzy sets for the status of VM and RAM parameters. The Fig. 7(a) and (b) show the fuzzy

Table 1 Fuzzy rules for calculating the inertia weight

Iteration	Result
Low	High
Medium	Medium
High	Low

Table 2 Fuzzy rules for calculating the fitness value

Task length	CPU speed	RAM	Status of VM	Result
Low	Low	Low	Low	High
Low	Low	Medium	Low	High
Low	Medium	Low	Medium	Medium
Low	Medium	Medium	High	Medium
Low	Medium	High	Medium	Medium
Low	High	Medium	Low	High
Low	High	Medium	Medium	Medium
Low	High	High	High	Low
Medium	Low	Low	Low	High
Medium	Low	Medium	High	Low
Medium	Low	High	Medium	Medium
Medium	Medium	Medium	Medium	Medium
Medium	Medium	Medium	Low	High
Medium	High	Low	Low	High
Medium	High	Medium	High	Low
Medium	High	Low	Medium	Medium
High	Low	Low	Low	Medium
High	Low	Medium	Medium	Low
High	Low	High	High	Low
High	Medium	Low	Low	High
High	Medium	Medium	Medium	Medium
High	Medium	Medium	High	Low
High	High	Low	Low	High
High	High	High	Medium	Medium

sets for the fitness value and Center of Gravity. The Fig. 8(a) and (b) show the fuzzy sets for the inertia weight and the iteration parameters. The implementation of FIS and the creation of membership functions have been done by using the JFuzzyLogic [39] library for the Fuzzification, Defuzzification and for defining the rule blocks. This library allows designing and developing FLCs according to the standard IEC 61131 [40]. JFuzzyLogic offers a fully functional and complete implementation of a FIS according to this standard and provides a programming interface and plug-into easily write and test code for fuzzy control applications.

4.3.3 Running example of proposed approach

We now provide a simple example as shown in Tables 3 and 4(a) and (b) to illustrate the functionality of the proposed work. Our example consists of a case where we schedule 10 tasks. We compare our proposed algorithms with the basic policy FCFS. This example is done under the same settings presented in Table 6, except the change of CPU speed of VMs to 500 MIPS. We assume that the cost of using the full computing power of the resources is \$20 per unit time. To simplify our example, we assume that users send 10 tasks {T1, T2, T3,…T10} with the length and burst time characteristics shown in Table 3. These tasks are at first stored into the waiting time queue. Therefore, to optimize the waiting time of tasks set, WTO-PSO algorithm is applied as shown in Table 4(a). The best optimization is obtained by WTO-PSO with minimum waiting time which is 62.9 and the optimal order is T6-T7-T1-T3-T4-T9-T2-T8-T5-T10. However, the waiting time result is 88.7 in the case of using FCFS which keep the same order of tasks as in the waiting time queue. The best sequence which can minimize the waiting time and decrease the queue length is used now as an input of the next part. Therefore, based on the optimal tasks sequence obtained, the dynamic dispatch queues algorithm is applied in order to dispatch these tasks among dynamic queues. The optimal number of queues in the case of TSDQ-FLPSO is 5 queues and 4 queues are created in the case of TSDQ-SAPSO. The difference in the dispatching queues strategy

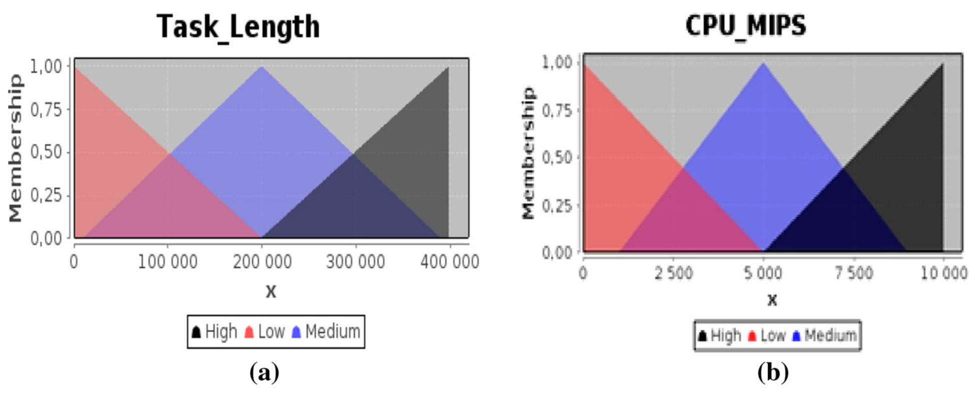

Fig. 5 **a** Fuzzy sets for task length parameter. **b** Fuzzy sets for CPU parameter

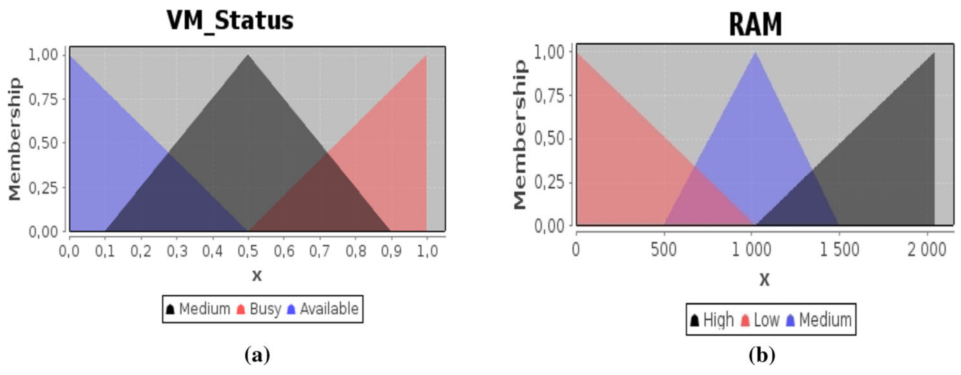

Fig. 6 **a** Fuzzy sets for status of VM parameter. **b** Fuzzy sets for RAM of VM parameter

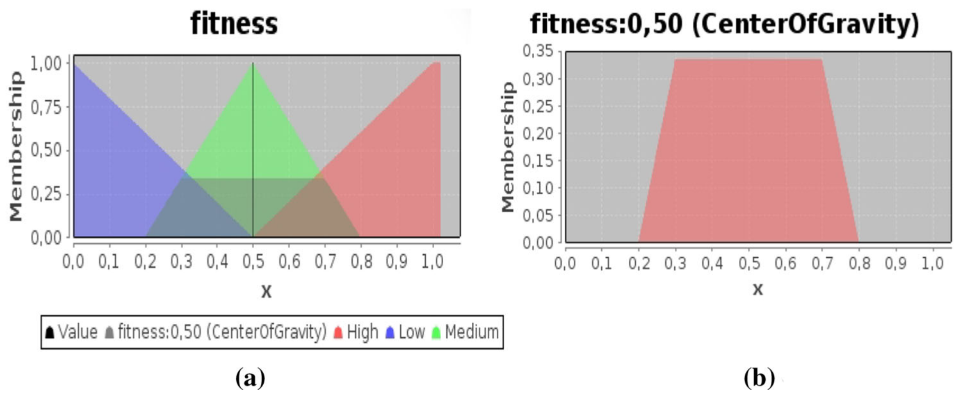

Fig. 7 **a** Fuzzy sets for fitness value. **b** Center of gravity of fitness value

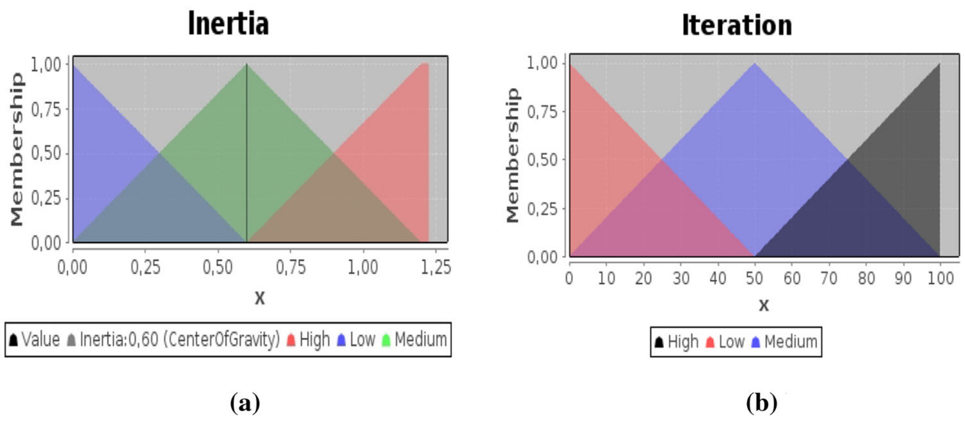

Fig. 8 **a** Fuzzy sets for inertia value. **b** Fuzzy sets for Iteration value

is shown clearly in Table 4(b). For example, the first queue Q_1 in TSDQ-FLPSO store only one tasks (T6) which is the first one in task sequence processed but TSDQ-SAPSO store two tasks (T6 and T7) in the first queue Q_1. This process continues until it dispatches all arrived tasks. After the dynamic dispatch queues process is finished, FLPSO/SAPSO algorithm selects each queue and schedules the tasks to the appropriate resources. The final results show different performance metrics of our algorithms compared to FCFS. We can detect that there is a very slight difference between the proposed algorithms TSDQ-FLPSO and TSDQ-SAPSO. Both proposed algorithms can achieve better results than FCFS in terms of waiting time, makespan, execution cost, Degree of imbalance (DI) and Resource Utilization (RU). Finally, we conclude that the proposed algorithms achieve the goal of scheduling optimization and exhibit favorable performance.

5 Performance evaluation of simulation results

5.1 Simulation environment

In order to evaluate the effectiveness of the proposed approaches TSDQ-SAPSO and TSDQ-FLPSO, the

Table 3 Tasks characteristics

Task ID	Task burst time	Task length
1	10	800
2	30	100
3	13	3500
4	19	3000
5	44	2326
6	4	3300
7	8	4000
8	36	5000
9	20	6000
10	50	7000

performance evaluations and the comparison with other algorithms were implemented on the CloudSim Cloud Simulator [41]. This simulator allows modeling and simulating extensible clouds, and testing the performance of developed application service in a controlled environment. CloudSim toolkit supports modeling of cloud system components such as cloud data centers, users, virtual machines and resource provisioning policies. Cloudsim gives several functionalities such as generating a different workload with different scenarios and performing robust tests based on the custom configurations.

5.2 Experimental results

Several experiments with different parameters settings were performed to evaluate the efficiency of our work. Therefore, to objectively assess the performance of the proposed algorithms, we have compared our algorithms TSDQ-SAPSO and TSDQ-FLPSO with other works proposed in [13, 14, 17, 18]. The evaluation has been done using synthetic and real data sets from workload data generated from real systems available from the parallel workload archive (PWA) [7]. We have evaluated the proposed work as following:

(1) First experiment: FCFS versus WTO-PSO: We compare our algorithm WTO-PSO with FCFS in terms of waiting time.
(2) Second experiment: TSDQ-SAPSO versus TSDQ-FLPSO versus FCFS versus PSO-LDIW versus PSO-RIW: In this experiment, we compare our algorithms with PSO-LDIW [14], PSO-RIW [18] and FCFS in terms of average makespan.
(3) Third experiment: TSDQ-SAPSO versus TSDQ-FLPSO versus SGA versus MGA: In this experiment, the simulation results of our proposed algorithms is compared with SGA and MGA results reported in literature [17] in terms of average makespan and execution cost.
(4) Fourth experiment: TSDQ-SAPSO versus TSDQ-FLPSO versus MACOLB: we compare our algorithms results with MACOLB results reported in literature [13] in terms of average makespan and average degree of imbalance.
(5) Fifth experiment: TSDQ-FLPSO versus TSDQ-SAPSO versus SA_PSO versus PSO: In the last experiment, we test TSDQ–FLPSO, TSDQ-SAPSO, SA_PSO (Simulated Annealing combined with PSO) and PSO in terms of makespan, degree of imbalance, resource utilization and cost. Finally, we compare TSDQ–FLPSO and TSDQ-SAPSO in terms of average number of dynamic queues created during the execution of the algorithms in this experiment.

5.2.1 First experiment: FCFS versus WTO-PSO

In the first experiment, we compared the proposed algorithm WTO-PSO with FCFS algorithm. The purpose of this simulation is to find the best minimum waiting time sequence in the tasks waiting queue. To illustrate the case, we provide a simple example. We assume that 3 independent tasks (T1, T2, T3) are submitted by users to be handled by a provider and are at first stored into waiting time queue. In this simulation, 10 experiments have been done where the burst time has changed for each task in these experiments. The parameter settings of WTO-PSO are described in Table 7.

Table 5 shows the comparison results in terms of the waiting time of each sequence and the total average waiting time. For example, in the last Serial No. 10, there are three tasks (T1,T2,T3) which require processing time (16,10,14) respectively. Using FCFS algorithm, the waiting time in this case is 14,00. However, by using WTO-PSO algorithm which has obtained the T2,T3,T1 sequence which is the best solution that gives the minimum waiting time, the result becomes 11.33 which is less than using FCFS algorithm. The results show that WTO-PSO can optimize the waiting time of different serials and reduce the total average waiting time. In the case of FCFS algorithm, it is shown clearly that it cannot provide an optimized performance which consequently leads to long waiting times and long waiting queue, particularly when the number of tasks is very large. This important difference between the two algorithms shows that managing the tasks waiting queue by WTO-PSO instead of FCFS can effectively minimize the waiting time and decrease the queue length. As a result, this can improve the system performance significantly, especially when the number of tasks scales up in high dimensional problem.

Table 4 Comparison results of the running example

(a)

	Waiting Time Queue	Waiting Time Optimization Algorithm	
		Waiting time	Optimal order
TSDQ-FLPSO Algorithm	→ T10 | T9 | T8 | T7 | T6 | T5 | T4 | T3 | T2 | T1 →	62.9	→ T10 | T5 | T8 | T2 | T9 | T4 | T3 | T1 | T7 | T6
TSDQ-SAPSO Algorithm	→ T10 | T9 | T8 | T7 | T6 | T5 | T4 | T3 | T2 | T1 →	62.9	→ T10 | T5 | T8 | T2 | T9 | T4 | T3 | T1 | T7 | T6
FCFS Algorithm	→ T10 | T9 | T8 | T7 | T6 | T5 | T4 | T3 | T2 | T1 →	88.7	→ T10 | T9 | T8 | T7 | T6 | T5 | T4 | T3 | T2 | T1

(b)

	Waiting Time Optimization Queue	Dynamic Dispatch Queues Algorithm	Performance metrics				
			Fitness value	Makespan	DI	RU	Cost
TSDQ-FLPSO Algorithm	T10 | T5 | T8 | T2 | T9 | T4 | T3 | T1 | T7 | T6 →	Q1 → T6 → Q2 → T1 | T7 → Q3 → T4 | T3 → Q4 → T9 → Q5 → T10 | T5 | T8 | T2 →	1.596	14.09	0.92	82.1%	$281,8
TSDQ-SAPSO Algorithm	T10 | T5 | T8 | T2 | T9 | T4 | T3 | T1 | T7 | T6 →	Q1 → T7 | T6 → Q2 → T1 → Q3 → T4 | T3 → Q4 → T10 | T5 | T8 | T2 | T9 →	1.701	14.15	0.93	82 %	$283
FCFS Algorithm	T10 | T9 | T8 | T7 | T6 | T5 | T4 | T3 | T2 | T1 →	---	---	18.75	1.09	75%	$375

Figure 9 shows that the proposed algorithm WTO-PSO outperforms FCFS algorithm. Our algorithm can effectively overcome the shortcoming of FCFS and give an optimized solution, which increases speed and efficiency of managing the task waiting queue by minimizing the tasks waiting time spent in the queue, reducing the queue length and keeping it as short as possible.

5.2.2 Second experiment: TSDQ-SAPSO versus TSDQ-FLPSO versus PSO-LDIW versus PSO-RIW versus FCFS

For the second experiment, we compare TSDQ-SAPSO and TSDQ-FLPSO algorithms with PSO-LDIW [14], PSO-RIW [18] and FCFS in terms of makespan. In this simulation, the standard formatted workload of a High-Performance Computing Center North called (HPC2N) log was

Table 5 Results comparison of WTO-PSO and FCFS

Serial no.	Burst time of tasks			FCFS	WTO-PSO
	T1	T2	T3		
1	6	4	8	5.33	4.67
2	34	12	20	26.67	14.67
3	43	15	30	33.67	20.00
4	24	25	10	24.33	24.33
5	13	5	20	10.33	7.67
6	30	12	18	24.00	14.00
7	40	20	33	33.33	24.33
8	14	24	21	17.33	16.33
9	8	3	14	6.33	4.67
10	16	10	14	14.00	11.33
Total average waiting time				195.33	142.00

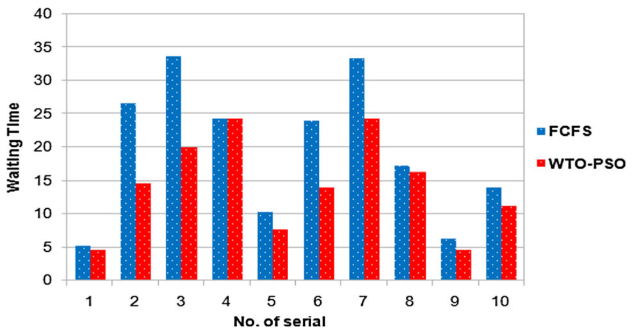

Fig. 9 Average waiting time using FCFS and WTO-PSO

Table 6 Resource parameters

Parameters	Values
Datacenter	
Number of datacenters	1
Number of hosts	2
Virtual machine	
Number of VMs	5
MIPS	9726
VM memory (RAM)	0.5 GB
Bandwidth	1 GB
Tasks source	Workload HPC2N log

Table 7 Parameter settings for PSO

Parameters	Values
Particle size	100
Maximum iteration	1000
Acceleration coefficients (C1,C2)	1.49445

used to generate different workloads [42]. This log contains three and a half years' worth of accounting records from the High-Performance Computing Center North in Sweden. HPC2N is a joint operation with several universities and facilities, and containing 527,371 jobs. The simulation has been done based on the parameter settings of PSO-LDIW [14] and PSO-RIW [18] described in Tables 6 and 7.

In Fig. 10, performances were compared in terms of average makespan for different numbers of tasks. The results obtained show that the makespan of the proposed algorithms TSDQ-SAPSO and TSDQ-FLPSO is better and increases more slowly than the other algorithms. It can be seen that the proposed algorithms take less time to execute all tasks and outperform PSO-LDIW and PSO-RIW in terms of both quality of solutions and the convergence speed. From the figure, we can see that, when the search space expands, both proposed algorithms show more stability. But in the case of PSO-LDIW and PSO-RIW algorithms, the chance of finding an improved optimal solution becomes harder. TSDQ-FLPSO algorithm shows good convergence characteristics. As a result, the makespan is reduced significantly especially when the number of tasks increases. This is because, as mentioned earlier, based on dynamic queues and Fuzzy Logic in two stages, TSDQ-FLPSO algorithm assigns tasks to the most suitable resources by taking into consideration task length and VM capabilities such as CPU speed, occupancy rate, RAM and bandwidth. This helps find an optimal mapping and complete tasks in a shorter time. Besides, in every iteration while searching for the optimum, the algorithm assesses the solutions returned to choose the optimal one. In addition, as the VM status or the occupancy rate of the VM is one of the input parameters of the proposed algorithm, all this gives TSDQ-FLPSO the ability not only to make the best decision to assign the most suitable resources to the received tasks, but also to keep the resources as busy as possible.

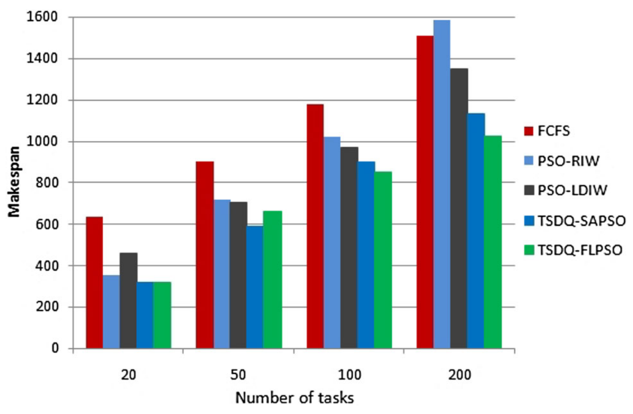

Fig. 10 Average makespan with different number of tasks

This effectively decreases the makespan by increasing the execution time of tasks.

5.2.3 Third experiment: TSDQ-FLPSO versus TSDQ-SAPSO versus SGA versus MGA

In the third simulation experiment, we compare the simulation results of the proposed algorithm TSDQ-FLPSO and TSDQ-SAPSO with SGA and MGA results reported in literature [17]. The performance evaluation is compared in terms of the average makespan and execution cost with a different number of tasks. The simulation has been done under the same conditions described in Tables 8 and 9 as specified in [17]. Table 8 shows the resources parameters and Table 9 presents the processor capacities and the costs of using the resources for processing which are selected randomly.

The Figs. 11 and 12 present the average makespan and execution cost of the various task scheduling algorithm compared under the same conditions, from which we can conclude that the proposed algorithms show significant improvements over SGA and MGA algorithms in minimizing makespan and minimizing execution cost respectively. Figure 11 shows that the makespan of TSDQ-FLSPO and TSDQ-SAPSO increases more slowly than SGA and MGA algorithms. This improvement is attributed to the proposed model which considers the tasks requirements and resource capabilities. Therefore, it can dynamically create the needed queues to manage the tasks and search for the solution based on the fitness function in order to find the efficient mapping of tasks to the most suitable resources. In general, TSDQ-SAPSO and TSDQ-FLPSO rapidly converge to the best solution in less

Table 8 Resource parameters

Parameters	Values
Datacenter	
Number of datacenters	2
Number of hosts	2
Virtual machine	
Number of VMs	10
MIPS	100–500
VM memory(RAM)	512–2048 MB
Bandwidth	500–1000 MB
Tasks source	
Number of tasks	10–30
Length of tasks	10,000–100,000 MIPS

Table 9 Resource parameters

Processor capacity	Per unit cost
100	15
200	20
300	25
400	30
500	40

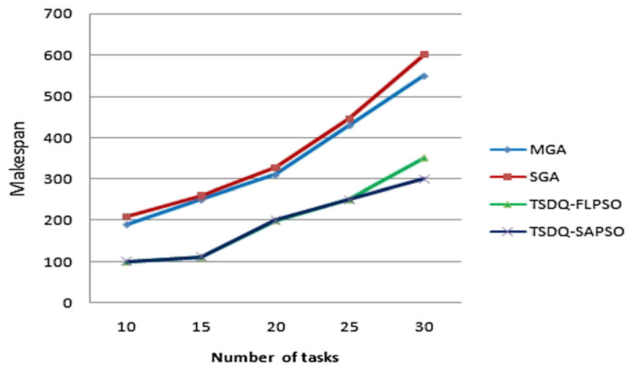

Fig. 11 Average makespan of our algorithms compared with SGA and MGA Reproduced with permission from [17]

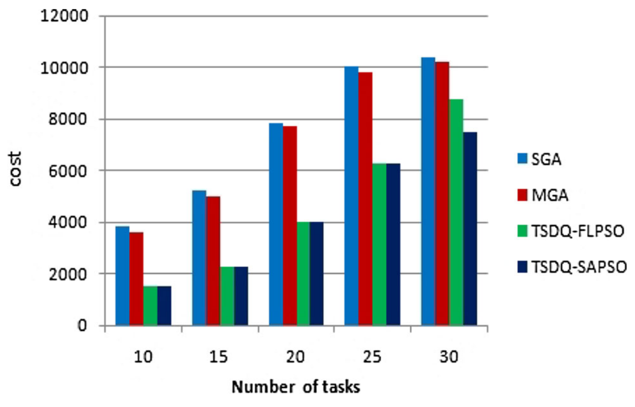

Fig. 12 Average execution cost of our algorithms compared with SGA and MGA Reproduced with permission from [17]

iteration. The results demonstrate also that the proposed algorithms solve two conflicting objectives by optimizing not only the makespan but also the execution cost. In the proposed algorithms, the resource is selected in a way that the task execution takes the minimum time and without ignoring the other metrics such the resource utilization and execution cost. The superiority of TSDQ-SAPSO and TSDQ-FLPSO is shown clearly in Fig. 12. Because the goal is to find a compromise between better execution time and low cost. In other words, to use the resources with lower processor capacity as much as possible while the solution obtained is reasonably good. From these results, we can outline that the proposed algorithms have the flexibility to dynamically optimize the resources allocation, improve the execution time and reduce the execution cost.

5.2.4 Fourth experiment: TSDQ-FLPSO versus TSDQ-SAPSO versus MACOLB

In this simulation experiment, we compare the simulation results of the proposed algorithms TSDQ-FLPSO and TSDQ-SAPSO with MACOLB results reported in literature [13]. The performance evaluation is compared in terms of the average makespan and average degree of imbalance with a different number of tasks. The simulation has been done under the following conditions described in Table 10 as specified in [13].

The Figs. 13 and 14 plot the comparison of proposed algorithms with MACOLB in terms of the average makespan and average degree of imbalance versus number of tasks. Experimental results show that the two task scheduling algorithms based on dynamic queues have obvious efficiency advantages and show approximately the same performances for all datasets. The average makespan and average DI of our algorithms is better than MACOLB algorithm. As can be seen, there is a significant difference between the proposed algorithms and MACOLB in terms of execution time and system load balancing. The proposed algorithms minimize the makespan, but meanwhile, they also pay much more attention to the system load balancing. It can be observed that the solution quality of our algorithms improves significantly when the number of tasks increases. Because important factors such as VM status and CPU speed are taken into account during the scheduling process. Our algorithms use all possible capabilities of the resources while scheduling tasks. Furthermore, TSDQ-SAPSO and TSDQ-FLPSO show excellent ability to determine the optimal load balancing between resources which prevent the makespan from growing too large and avoid unbalanced workload of VMs. In other words, this means that the resources have an optimal amount of load; none of them is too busy while some others are idle or not used at any given time. Hence, the optimal and good load balancing leads to the highest resource utilization.

Table 10 Resource parameters

Parameters	Values
Datacenter	
Number of datacenters	10
Number of hosts	2–6
Virtual machine	
Number of VMs	50
MIPS	500–2000
VM memory (RAM)	256–2048 MB
Bandwidth	500–1000 MB
Number of PEs requirement	1–4
Tasks source	
Number of tasks	100–1000
Length of tasks	1000–20,000 MIPS

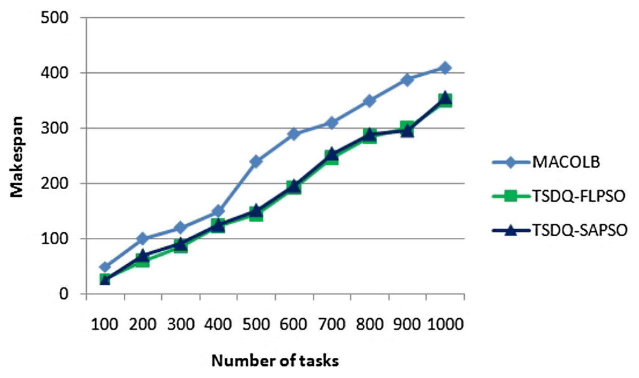

Fig. 13 Average makespan of our algorithms compared with MACOLB Reproduced with permission from [13]

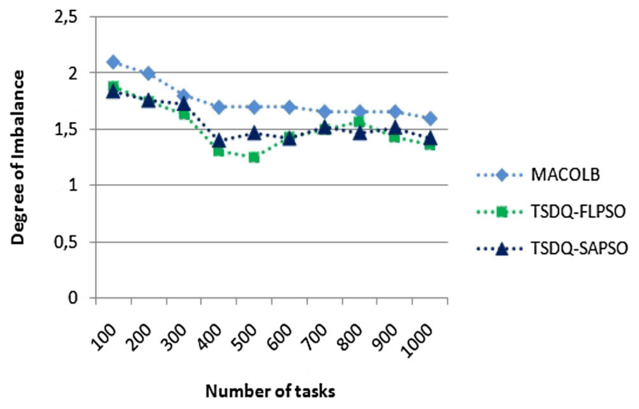

Fig. 14 Average degree of imbalance (DI) of our algorithms compared with MACOLB Reproduced with permission from [13]

5.2.5 Fifth experiment: TSDQ-FLPSO versus TSDQ-SAPSO versus SA_PSO versus PSO

In the fifth experiment, exhaustive simulations have been done to compare the following algorithms TSDQ-FLPSO, TSDQ-SAPSO, SA_PSO and PSO (PSO with constant inertia weight [43, 44]). The goal of this experiment was to examine and evaluate the performance of our algorithms to determine which algorithm works better and more stable under different scenarios. Compared with the previous experiments, we have increased significantly the overall tasks set, resources and their characteristics. Therefore, the performance evaluation is compared in terms of the average makespan, degree of imbalance, cost and resource utilization. In addition, we used a large real dataset generated from the NASA Ames iPCS/860 log [45]. The advantage of using this log directly as the input of this simulation is that it is reflects a real workload precisely, with all its complexities. This log contains 3 months' worth of sanitized accounting records for the 128-node iPSC/860 located in the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center. The workload on the iPSC/860 is a mixture of interactive and batch jobs (development and production) mainly consisting of computational aeroscience applications. The parameter settings of PSO are described in Table 7. The SA_PSO algorithm is the same SAPSO algorithm used in TSDQ-SAPSO in order to prove the advantage and effectiveness of our approach based on the dynamic dispatch queues. The simulation has been done 100 times under the following conditions described in Table 11.

In the Fig. 15, there is a clear correlation between the number of tasks and makespan, as the task quantity affects the efficiency of execution time of the set of tasks. PSO and SAPSO algorithms search for an optimal solution in the entire given search space without considering VMs capabilities and tasks lengths dynamically. Moreover, the choice of static inertia weight in PSO influences its convergence rate and often leads to premature convergence. As a result, PSO makespan is higher than all the other algorithms and has the worst makespan in all workload sets. We can also observe a slow increase in TSDQ-FLPSO makespan, which means that the execution time is significantly faster and better than other algorithms. When the number of tasks increases, the TSDQ-FLPSO makespan becomes very slower than the TSDQ-SAPSO because the TSDQ-FLPSO selects the resource on the basis of the task length, RAM memory, CPU speed and the status of the resources in the model of fuzzy logic. This model has the potential to improve the convergence speed and optimization efficiency of TSDQ-FLPSO algorithm. The TSDQ-FLPSO algorithm shows a good ability to evaluate the obtained results and to find the best fitness value and come up with the best decision even if the number of tasks increases. The results show that the TSDQ-FLPSO has the characteristics of fast convergence speed and can achieve good performances to reduce the makespan in comparison with other algorithms, notably the TSDQ-SAPSO algorithm based also on dynamic queues.

Table 11 Resource parameters

Parameters	Values
Datacenter	
Number of datacenters	10
Number of hosts	2–6
Virtual machine	
Number of VMs	5–50
MIPS	10,000–30,000
VM memory (RAM)	256–8196
Bandwidth	500–5000
Tasks source	Workload NASA iPSC/860 log

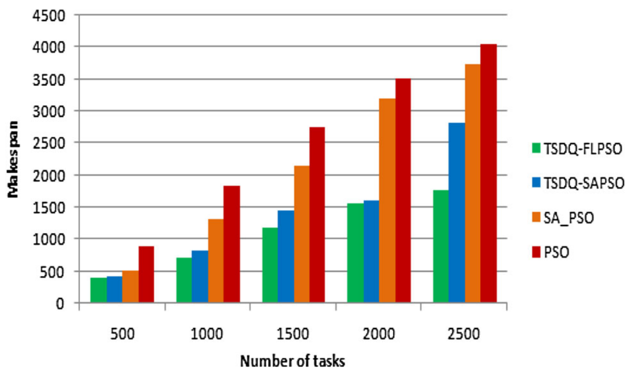

Fig. 15 Average makespan of different algorithms

A comparison of average degree of imbalance with a different number of tasks between different algorithms is depicted in Fig. 16. The results show that the proposed algorithms TSDQ-FLPSO and TSDQ-SAPSO perform better than the two others algorithms, and have a remarkable similarity in some cases. However, TSDQ-FLPSO algorithm has a good stability in the degree of imbalance, as it is maintained at a low level, and it is observed that it has a slight increase when the number of tasks increases. Moreover, all received tasks are distributed between the resources in the form of balancing, this explains the good performances achieved in terms of execution time. The use of SA and fuzzy inferences which are both used not only to dynamically adjust the inertia weight, but also to incorporate both tasks length and VMs capabilities in the fitness formulation. This hybridization provides good flexibility to control the balance between local and global exploration of the problem space and overcome premature convergence of PSO. Moreover, TSDQ-FLPSO shows some improvements

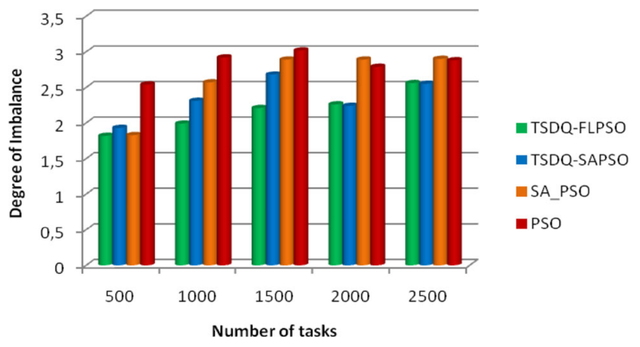

Fig. 16 Average degree of imbalance of different algorithms

over TSDQ-SAPSO which can be explained by the importance of input parameters of TSDQ-FLPSO and the application of the Fuzzy Logic in two stages. This effectively helps TSDQ-FLPSO not only find a potential optimal solution with less iteration and less computing time, but also select the resources with light load and good capabilities to process tasks which have an important length. The TSDQ-FLPSO algorithm shows superior performance and achieves good system load balance in any situation and takes less time to execute tasks.

Figure 17 shows the average resource utilization of the TSDQ-SAPSO, TSDQ-FLPSO, SA_PSO and PSO algorithms. Resource utilization is an essential performance metric as it represents the rate of utilization of resources, and at which level the resources are busy in executing tasks. The result illustrates that the resource utilization of TSDQ-FLPSO is maintained at a high level which means that it has the best resource utilization comparing with TSDQ-SAPSO, SA_PSO and PSO. SA_PSO algorithm shows good performance in comparison with PSO. However, the difference between our proposed algorithms and these algorithms is significant and prove the effectiveness of the proposed algorithms based on dynamic dispatch queues. Therefore, TSDQ-SAPSO and TSDQ-FLPSO have the flexibility to analyze, explore and schedule the tasks sets stored in the dynamic queues to carry out more intelligent scheduling decisions. The comparison between the two proposed algorithms shows that TSDQ-SAPSO exhibit less performance than TSDQ-FLPSO which shows better performances despite a slight decrease which could be observed before it continues to increase when the received workload becomes important. In addition, it can achieve a good utilization of resources and maintain a high occupancy rate of resources during the process of scheduling task. Indeed, TSDQ-FLPSO algorithm keeps the resources as busy as possible and uses all possible resources capabilities. The resource utilization metric is gaining significance as service providers want to earn a maximum profit by renting a limited number of resources.

Figure 18 illustrates the comparison of the cost between TSDQ-FLPSO, TSDQ-SAPSO, SA_PSO, and PSO with a different number of tasks. This metric presents the total amount of resources utilization that a user needs to pay to the service provider. For example, in this simulation, we assume that the cost of 5000 MIPS computing power of the resource is $10 per unit time. The result of simulation shows that the TSDQ-FLPSO performs better with less cost when the number of tasks increases. Based on the proposed scheduling strategy, TSDQ-FLPSO shows better flexibility to distribute incoming requests to the heterogeneous VMs, e.g., it is able to determine the most appropriate VM to execute each task in order to minimize the cost while considering other QoS requirements. TSDQ-FLPSO algorithm considers dynamically the task length and the resource processor capacity on the basis of the Mamdani inference system and defined rules. In other words, when the proposed algorithm calculates the function fitness, it takes into consideration the task and resource proprieties such as task length and CPU power of resource. Hence, the best fitness means that the TSDQ-FLPSO algorithm selects and uses VMs with higher computing power if the VMs with lower computing power cannot provide good results to tasks or give a non-optimal solution. As a result, the TSDQ-FLPSO algorithm can effectively achieve good performances in terms of cost.

Figure 19 illustrates the average dynamic queues created by TSDQ-FLPSO and TSDQ-SAPSO algorithms in this experiment. The two proposed algorithms search to get

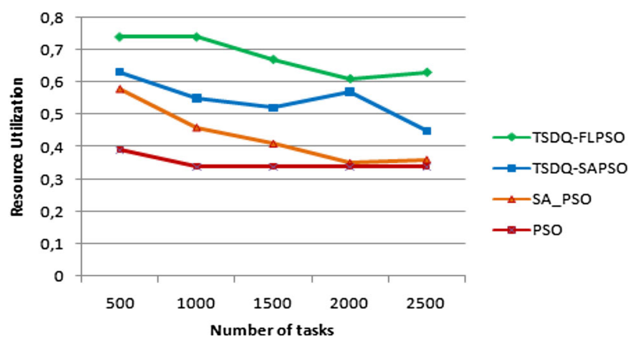

Fig. 17 Average resource utilization of different algorithms

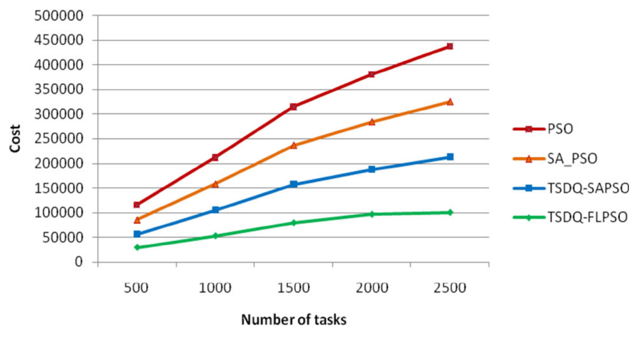

Fig. 18 Cost (in $) with different number of tasks

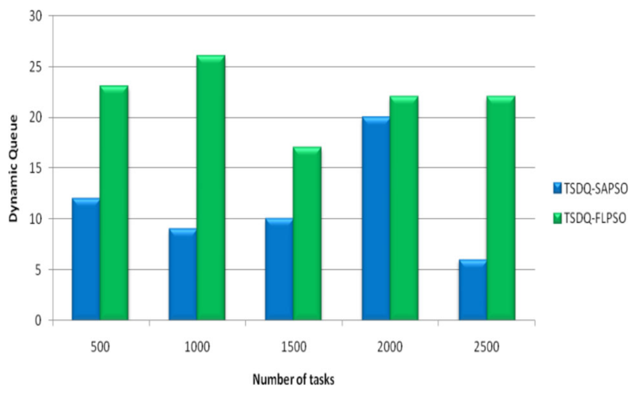

Fig. 19 Average dynamic queues created by TSDQ-FLPSO and TSDQ-SAPSO algorithms

an optimal result by taking into consideration the best decision threshold. Both of the proposed algorithms use the best threshold value that can not only create the queues dynamically, but also give good results while dispatching tasks among queues. The result shows that TSDQ-SAPSO algorithm creates minimum number of queues, but TSDQ-FLPSO algorithm creates more queues while searching for the optimum and attempting to find the best potential solution for the current set of tasks. The experimental results indicate that the optimal number of queues helps TSDQ-FLPSO to exhibit superior performance in terms of the convergence speed and the four performance metrics examined above. The TSDQ-FLPSO algorithm shows also a great advantage in comparison with TSDQ-SAPSO in terms of the number of dynamic queues created while dispatching the tasks among corresponding queues and mapping tasks to appropriate resources.

The experimental results above demonstrate that the proposed algorithms work very well for a different number of tasks and setting. Several experiments have been carried out in order to evaluate the proposed work. As we can see, the proposed approach based on dynamic queues and two hybrid meta-heuristic algorithms generates good solutions. The experiments conducted show that the superiority of our algorithms is not only applicable for synthetic tasks but also for real tasks. Finally, the results conclude that the proposed approach is an intelligent architecture which obtains optimal solutions and achieves better global convergence capability in terms of the performance metrics under consideration.

6 Conclusion

Tasks scheduling is one of the most challenging problems in cloud computing and plays an important role in the efficiency of the whole cloud computing facilities. The goal of task scheduling is to map tasks to the most appropriate and suitable resources to be executed with optimization of different parameters such as the execution time, cost, resource utilization, load balancing and so on. In this paper, we have proposed a novel approach for task scheduling optimization based on dynamic dispatch queues and hybrid meta-heuristic algorithms. We have proposed two hybrid meta-heuristic algorithms, the first one using Fuzzy Logic with Particle Swarm Optimization Algorithm (FLPSO), and the second one using Simulated Annealing with Particle Swarm Optimization algorithm (SAPSO). The objective of this approach is to get the best order of tasks to minimize the waiting time in the task waiting queue and dispatch the tasks among dynamic queues created by TSDQ algorithm, then assign the tasks to the most suitable resources and optimize the performance metrics of cloud. Several experiments have been done in order to evaluate the performances of the proposed approach compared with other works from literature such as PSO-LDIW, PSO-RIW, SGA, MGA and MACOLB. Our algorithms were later compared with PSO with constant inertia weight, and SA_PSO algorithm to show clearly the advantage of using the dynamic dispatch queues process in our approach. Also, to determine which algorithm works best and most stable under different scenarios. The superiority of the proposed work is validated with extensive experiments using CloudSim simulator with synthetic and real data sets from real systems available from the Parallel Workload Archive (PWA). TSDQ-FLPSO and TSDQ-SAPSO achieve good performance in minimizing the waiting time as well as the queue length, reducing the makespan, maximizing resources utilization, minimizing the execution cost and improving the load balancing. Finally, the experiments results prove the effectiveness of our algorithms and the optimal results is provided using TSDQ-FLPSO compared to TSDQ-SAPSO and other existing scheduling algorithms especially in a high dimensional problem.

In the future, we intend to enhance our work, by integrating other optimization algorithms and techniques, improving the robustness of algorithms, considering more QoS parameters such as the priority of tasks, allowing the migration of tasks between the queues, considering the energy consumption and VM migration concept.

References

1. Mell, P., Grance, T.: The NIST Definition of Cloud Computing, p. 800. National Institute of Standards and Technology. The NIST Special Publication, Gaithersburg (2011)
2. Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D.,

Rabkin, A.: A view of cloud computing. Commun. ACM **53**, 50 (2010)
3. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, pp. 1942–1948. IEEE Service Center, Piscataway (1995)
4. Eglese, R.: Simulated annealing: a tool for operational research. Eur. J. Oper. Res. **46**, 271–281 (1990)
5. Lee, C.: Fuzzy logic in control systems: fuzzy logic controller. I. IEEE Trans. Syst. Man Cybern. **20**, 404–418 (1990)
6. Ben Alla, H., Ben Alla, S., Ezzati, A., Mouhsen, A.: A Novel Architecture with Dynamic Queues Based on Fuzzy Logic and Particle Swarm Optimization Algorithm for Task Scheduling in Cloud Computing. Lecture Notes in Electrical Engineering, pp. 205–217. Springer, New York (2016)
7. Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload
8. Sujan, S., Kanniga Devi, R.: An efficient task scheduling scheme in cloud computing using graph theory. Proceedings of the International Conference on Soft Computing Systems. pp. 655–662 (2015)
9. Gupta, J., Azharuddin, M., Jana, P.: An effective task scheduling approach for cloud computing environment. Lecture Notes in Electrical Engineering. pp. 163–169 (2016)
10. Ma, J., Li, W., Fu, T., Yan, L., Hu, G.: A novel dynamic task scheduling algorithm based on improved genetic algorithm in cloud computing. In: Wireless Communications, Networking and Applications. pp. 829–835 (2015)
11. Guo, L., Zhao, S., Shen, S., Jiang, C.: Task scheduling optimization in cloud computing based on heuristic algorithm. J. Netw. **7**, 547–553 (2012)
12. Wu, X., Deng, M., Zhang, R., Zeng, B., Zhou, S.: A task scheduling algorithm based on QoS-driven in cloud computing. Proc. Comput. Sci. **17**, 1162–1169 (2013)
13. Keshk, A., El-Sisi, A., Tawfeek, M.: Cloud task scheduling for load balancing based on intelligent strategy. Int. J. Intell. Syst. Appl. **6**, 25–36 (2014)
14. Khalili, A., Babamir, S.M.: Makespan improvement of PSO-based dynamic scheduling in cloud environment. In: Proceedings of the 23rd Iranian Conference on Electrical Engineering, pp. 613–618 (2015)
15. Zulkar Nine, M., Azad, M., Abdullah, S., Rahman, R.: Fuzzy logic based dynamic load balancing in virtualized data centers. In: Proceedings of the 2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (2013)
16. Chen, Z., Zhu, Y., Di, Y., Feng, S.: A dynamic resource scheduling method based on fuzzy control theory in cloud environment. J. Cont. Sci. Eng. **2015**, 1–10 (2015)
17. Kaur, S., Verma, A.: An efficient approach to genetic algorithm for task scheduling in cloud computing environment. Int. J. Inf. Technol. Comput. Sci. **4**, 74–79 (2012)
18. Al-Olimat, H., Alam, M., Green, R., Lee, J.: Cloudlet scheduling with particle swarm optimization. In: 2015 Fifth International Conference on Communication Systems and Network Technologies (2015)
19. Gabi, D., Ismail, A., Zainal, A., Zakaria, Z.: Solving task scheduling problem in cloud computing environment using Orthogonal Taguchi-Cat Algorithm. Int. J. Electr. Comput. Eng. **7**, 1489 (2017)
20. Komarasamy, D., Muthuswamy, V.: ScHeduling of jobs and adaptive resource provisioning (SHARP) approach in cloud computing. Clust Comput (2017). https://doi.org/10.1007/s10586-017-0976-3
21. Zhang, P., Zhou, M.: Dynamic cloud task scheduling based on a two-stage strategy. IEEE Trans. Autom. Sci. Eng. **15**, 1–12 (2017)
22. Zuo, L., Dong, S., Shu, L., Zhu, C., Han, G.: A Multiqueue interlacing peak scheduling method based on tasks' classification in cloud computing. IEEE Syst. J. **6**, 1–13 (2016)
23. Peng, Z., Cui, D., Zuo, J., Li, Q., Xu, B., Lin, W.: Random task scheduling scheme based on reinforcement learning in cloud computing. Clust. Comput. **18**, 1595–1607 (2015)
24. Karthick, A., Ramaraj, E., Subramanian, R.: An Efficient multi queue job scheduling for cloud computing. In: Proceedings of the 2014 World Congress on Computing and Communication Technologies. (2014)
25. He, T., Cai, L., Deng, Z., Meng, T., Wang, X.: Queuing-Oriented Job Optimizing Scheduling In Cloud Mapreduce. In: Proceedings of the International Conference on Advances on P2P, Parallel, Grid, Cloud and Internet Computing. pp. 435–446 (2016)
26. The Apache Hadoop Project: http://hadoop.apache.org/
27. Blondin, J.: Particle swarm optimization: a tutorial (2009). http://www.cs.armstrong.edu/saad/csci8100/pso_tutorial.pdf
28. Clerc, M., Kennedy, J.: The particle swarm—explosion, stability, and convergence in a multidimensional complex space. IEEE Trans. Evol. Comput. **6**, 58–73 (2002)
29. Feng, Y., Teng, G., Wang, A., Yao, Y.: Chaotic Inertia Weight in Particle Swarm Optimization. In: Proceedings of the Second International Conference on Innovative Computing, Information and Control (ICICIC 2007). (2007)
30. Xin, J., Chen, G., Hai, Y.: A Particle Swarm Optimizer with Multi-stage linearly-decreasing inertia weight. In: Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization. (2009)
31. Yue-lin, G., Yu-hong, D.: A new particle swarm optimization algorithm with random inertia weight and evolution strategy. In: Proceedings of the 2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007). (2007)
32. Kumar, S., Chaturvedi, D.: Tuning of particle swarm optimization parameter using fuzzy logic. In: Proceedings of the 2011 International Conference on Communication Systems and Network Technologies. (2011)
33. Kennedy, J., Eberhart, R.: A discrete binary version of the particle swarm algorithm. In: Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation. (1997)
34. Mendel, J.: Fuzzy logic systems for engineering: a tutorial. Proc. IEEE **83**, 345–377 (1995)
35. Kalra, M., Singh, S.: A review of metaheuristic scheduling techniques in cloud computing. Egypt. Inform. J. **16**, 275–295 (2015)
36. Li, K., Xu, G., Zhao, G., Dong, Y., Wang, D.: Cloud task scheduling based on load balancing ant colony optimization. 2011 In: Proceedings of the Sixth Annual Chinagrid Conference. (2011)
37. Yin, P., Yu, S., Wang, P., Wang, Y.: Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization. J. Syst. Softw. **80**, 724–735 (2007)
38. Mamdani, E.: Application of fuzzy algorithms for control of simple dynamic plant. Proc. Inst. Electr. Eng. **121**, 1585 (1974)
39. Cingolani, P., Alcalá-Fdez, J.: jFuzzyLogic: a Java library to design fuzzy logic controllers according to the standard for fuzzy control programming. Int. J. Comput. Intell. Syst. **6**, 61–75 (2013)
40. Cingolani, P., Alcala-Fdez, J.: jFuzzyLogic: a robust and flexible Fuzzy-Logic inference system language implementation. In: Proceedings of the 2012 IEEE International Conference on Fuzzy Systems. (2012)
41. Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software **41**, 23–50 (2011)

42. The High-Performance Computing Center North (HPC2N) in Sweden, http://www.cs.huji.ac.il/labs/parallel/workload/l_hpc2n/
43. Arumugam, M., Rao, M.: On the performance of the particle swarm optimization algorithm with various inertia weight variants for computing optimal control of a class of hybrid systems. Disc. Dyn. Nat. Soc. **2006**, 1–17 (2006)
44. Umapathy, P., Venkataseshaiah, C., Arumugam, M.: Particle swarm optimization with various inertia weight variants for optimal power flow solution. Disc. Dyn. Nat. Soc. **2010**, 1–15 (2010)
45. Parallel Workloads Archive: NASA Ames iPSC/860, http://www.cs.huji.ac.il/labs/parallel/workload/l_nasa_ipsc/

Hicham Ben Alla is currently a Ph.D. student at Hassan first University, Settat (Morocco). He obtained the Computing Engineering degree from the same University in 2014. His research interests include Cloud Computing, Scheduling Algorithms, Resources Management, High-Performance Computing and Artificial Intelligence.

Said Ben Alla received his M.S. degree in Telecommunications and Networks in 2009 from the University of Cadi Ayyad, Morocco and his Ph.D. from the Hassan 1st University, Faculty of Sciences and Techniques (FSTS) Settat, Morocco in 2013. He is currently a professor at ESTB, Hassan 1st University Settat. His current research interests include Cloud Computing, embedded Real-Time Systems, Wireless Sensor Networks (WSNs), wireless ad hoc networks with main focus on Routing Protocols Development and Mobility Management.

Abdellah Touhafi is obtained his M.Sc. degree in Electronic Engineering from Vrije Universiteit Brussel (Belgium) in 1995 and his Ph.D. from the Faculty of Engineering Sciences from Vrije Universiteit Brussel (Belgium) in 2001. In 2001 he became post-doctoral researcher at Erasmushoge School Brussel where he researched on environmental monitoring systems. In 2003 he became professor and founded his research group on reconfigurable and embedded systems. Since 2009 he is the program coordinator in the Industrial Sciences Department. His current research interests include Cloud Computing, High Performance and Reconfigurable Computing, Embedded Real-time Systems, Security, Software Defined Radio and Digital Communication Circuits.

Abdellah Ezzati received a Research Habilitation degree from the Hassan 1st University, Faculty of Sciences and Techniques (FSTS), Settat, Morocco, in 2012. Since 1994 he has been working as a Professor at the Department of Mathematics and Computer at the Faculty of Sciences and Techniques, Settat, Morocco. His research interests are cloud computing, distributed systems and WSN management. Dr. Abdellah Ezzati is also interested in Protocol Specifications and Mobility Management.