J Supercomput (2017) 73:4796–4822
DOI 10.1007/s11227-017-2050-6

A multi-parameter scheduling method of dynamic workloads for big data calculation in cloud computing

Ali Hanani[1] · **Amir Masoud Rahmani**[1] · **Amir Sahafi**[2]

Published online: 24 April 2017

Abstract Workload scheduling in cloud computing is currently an active research field. Scheduling plays an important role in cloud computing performance, especially when the platform is used for big data analysis and as less predictable workloads dynamically enter the clouds. Finding the optimized scheduling solution with different parameters in different environments is still a challenging issue. In dynamic environments such as cloud, scheduling strategies should feature rapid altering to be able to adapt more easily to the changes in input workloads. However, achieving an optimized solution is an important issue, which has a trade-off with the speed of finding the solution. In this article, an ordinal optimization method is proposed that considers the volume of workloads, load balancing and the volume of exchanged messages among virtual clusters, considering the replications. The algorithm in the present paper is based on ordinal optimization (OO) and evolutionary OO. In any time periods, a criterion is calculated to determine the similarity of workloads in two-consequence time periods, which is appropriate for timely changes in the scheduling procedure. In this paper, considering more than one parameter, a proper scheduling would be created for each time period. This scheduler is an organization for the number of virtual machines for each virtual cluster, but if there is a desirable similarity between

✉ Amir Masoud Rahmani
rahmani@srbiau.ac.ir

Ali Hanani
ali.hanani@srbiau.ac.ir

Amir Sahafi
sahafi@iau.ac.ir

[1] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran

[2] Department of Computer Engineering, South Tehran Branch, Islamic Azad University, Tehran, Iran

workloads of two-consequence time periods, this procedure would be ignored. The results show that a more optimized solution is obtained in comparison with the rated methods, such as blind pink, OO, Monte Carlo and eOO in a reasonable time. The suggested method is flexible and it is possible to change the weight ratio of the proposed criteria in different environments to be consistent with different environmental conditions. The results show that proposed method achieved up to 28% performance improvement in comparison with eOO.

Keywords Cloud computing · Workload scheduling · Big data · Optimization · Virtual clustering · Throughput

Abbreviations

OO	Ordinal optimization
eOO	Evolutionary ordinal optimization
VM	Virtual machine
LAN	Local area network
JAWS	Job-aware workload scheduling
PB	Petabytes
RFOH	Resource fault occurrence history
IOO	Iterative ordinal optimization
MOS	Multi-objective scheduling
DVS	Dynamic voltage scaling
VC	Virtual cluster
BIG	Big workflow generation
VBIG	Very big workflow generation
HUGE	Huge workflow generation
MEOO	Multi-parameter evolutionary algorithm
FIFO	First in first out

1 Introduction

The volume of data that are generated by the existing virtual community is rapidly increasing. In other words, a large volume of data is generated and stored by various organizations [1]. With the advent of social network Web sites, users create a lot of data about lives by daily posting details of the activities they perform, events they attend and places they visit [2]. Such data are referred to as big data [2] that require high volumes of storage and computational power for its storage and mining. Cloud computing is a type of computation, which offers scalable, computing capabilities as a service to customers [3]. The appearing cloud-based environments with distributed data centers, while providing strong processing resources, bring about the need for parallel/distributed algorithms [4]. For the selfsame reason, many organizations can present their large calculations on cloud platforms [2]. Moreover, virtualization technology conceals heterogeneous computing resources and cloud customers can meet their alternate demands in cloud compliance [5].

Cloud computing provides high-performance computing in scientific and engineering applications [6,7]. Therefore, many researchers have tried different methods to increase the efficiency in various cloud systems. One of the topics, which are paid more attention to, is scheduling. An important aspect of scheduling paradigm is providing appropriate computing resources for the requests that are received by cloud, regarding the high volume of workload streams with large data and at the right time and right place [8]. The type of scheduling big data work stream in the available resources in the cloud environment is an NP-hard problem [9,10]. The main challenge is to maintain an acceptable trade-off between scheduling overhead and scheduling accuracy [10].

In this paper, a new method is proposed to perform workloads–jobs that have been made up of numerous interrelated tasks scheduling. The proposed method, like the other heuristic ones, attempts to find a suboptimal solution in the whole answer environment instead of a completely optimized answer, yet in a shorter time. The algorithm is based on a method called ordinal optimization (OO) [10], which searches for suboptimal solutions in a shorter period of time. In addition, OO can analyze the input workloads to reach a better result. This article investigates a recent work of the authors of [11], called evolutionary ordinal optimization (eOO) [10], to propose a new and more complete method in workloads analysis for calculating their similarity formula. We have also covered other important parameters in the cloud environment, such as the problem of load balancing or message passing between virtual clusters, to find the appropriate scheduling. The basis of involving workload similarities in finding the solution is for the scheduler to have the ability to adapt itself to different environments where scheduling policy should change constantly for more optimization purposes. On the other hand, when the fluctuation of the work environment is low, frequent changing in scheduling policy is not needed, as it is feasible to impose fewer overloads to the scheduler.

eOO method provides a random solution. It attempts to choose a more suitable solution by n iterations (where n is much less than the total number of possible solutions) and selects the best solution for the scheduling problem. Then, it calculates workload similarities with respect to the volume of workload. In the next step, if the similarity reaches a specific threshold during the time intervals, the system continues the former strategy; otherwise, it calculates the random solution again and changes the strategy.

The novel features of our proposed algorithm are as follows:

- Calculation of similarity based on the workloads volume does not seem sufficient. As a result, we suggest that, in addition to volume, replication factors and messaging clusters be considered in workload similarities.
- We added a new step to the scheduling algorithm so that in each time interval the feedbacks of all executions contribute to finding a better solution under the assumption of lack of similarity and the need to change the scheduling strategy. At this stage, the impact of the queue for each virtual cluster has been taken into account to have more balance between the loads on each cluster and the others.
- If a cluster is loaded too much and its queue grew more than others, then the scheduling order should change. By giving more probability to the cluster with more loads, its chance for absorbing more virtual machines (VMs) in the next step

will increase. As a result, that cluster executes more jobs in its queue with more VMs.

Involving load balancing and replication in the scheduling process is very helpful in finding the optimal solution, and the implementation results show the improved performance of the algorithm.

The organization of the rest of this paper is as follows: In Sect. 2, we survey previous studies in the literature; in Sect. 3, the proposed algorithm is delineated. Section 4 is dedicated to implementation and assessing the proposed algorithm, and finally, in Sect. 5, conclusions and suggestions for future research are mentioned.

2 Related work

Many studies have aimed at resolving the scheduling issue in different areas. In this article, we address the recently proposed algorithms for scheduling in grid and cloud computational systems. Many efforts were made to solve the scheduling issue, using meta-heuristic algorithms and the swarm intelligence. Hanani et al. [12] investigated multi-processor systems scheduling, using artificial bee colony algorithm. In that study, better results were obtained rather than genetic algorithm and ant colony optimization algorithm by adding memory for bees in artificial bee colony [12].

Job scheduling is often used in distributed systems, such as smart grid, in order to optimize one or more specific quality-of-service parameters that are often throughput or makespan [13]. Some researchers have also considered other important aspects of distributed computations along with the scheduling issue. In some of the proposed methods, features such as replication, security and load balancing are included besides scheduling [5,14–19]. For example, Mansouri et al. [14] presented a method for improving data availability in the data grid by combining data replication and scheduling algorithm. They made a hierarchical classification of available resources. Interrelated sites were put in a local area network (LAN) and several LANs were organized in a region. Then, based on the site's relationship costs (sites' relationship cost of one region is naturally less than the cost of sites in two different regions) to access the required replica and elapsed time in queue, a function was used to calculate the combined cost; then, the scheduling was done. They also did some work to manage replicas in the case of lack of storage space [14]. As another instance, Rahmati et al. [15] combined data replication and job scheduling. They integrated these two techniques to enhance the performance of data-intensive applications execution. They integrated these two techniques as a single objective, aiming at reducing response time by data access time reduction in cloud computing environment. Simulation results showed the effectiveness of their algorithm in comparison with well-known algorithms. Furthermore, Jiang et al. [18] used security as a parameter in the grid scheduling.

Some other studies similar to our work analyze jobs and other entered data as well as existing resources to provide more optimized scheduling according to the input streams and existing resources. Studies like [20–24] in a grid environment and [25–29] in other environments, such as cloud, focus on input job streams and resources in scheduling issue. Jianhua et al. [20] suggested a method for data-aware scheduling in the data grid. They added a data-aware module to scheduler that puts the available

resources information besides the file's data and performs the scheduling. Mei et al. [21] proposed a novel duplication-based scheduling algorithm to prevent unnecessary duplications that impose a huge cost and overload. Wang et al. presented a technique called job-aware workload scheduling (JAWS). It increases the query throughput for data-intensive scientific database clusters. The investigated dataset poses a volume in petabytes (PB) scale. They divided queries to the 1/0-friendly sub-queries, and then, the data requirements which overlap with workloads were recognized and carried out the scheduling based on that [22]. Khanli et al. presented a fault- tolerant job scheduler in grid computing via a new strategy named resource fault occurrence history (RFOH) in computational grid. In their work, broker maintained the history of fault occurrence of resources in grid information server and it utilized genetic algorithm to find a near-optimal solution for the problem. In addition, it increased the percentage of jobs executed within specified deadline [23]. Kazem et al. proposed a modified simulated annealing algorithm for scheduling-independent tasks in a grid environment. Experimental results showed that their algorithm improved the performance of static instances compared to the results of other algorithms reported in the literature [24]. Zhang et al. [25] presented a method for workflow scheduling on cloud computing platform. Their method, called iterative ordinal optimization (IOO), employs ordinal optimization in a way that it is able to reach suboptimal schedule in each iteration. Their test method has been done on the IBM RC2. Zhang et al. following their work proposed another method called multi-objective scheduling (MOS). This method also employed OO capabilities for cloud environments [26]. Nanduri et al. offered a scheduling method based on map reduce. Their method was also job-aware, and they divided tasks into four categories of CPU-intensive, memory-intensive, disk-intensive and network-intensive. Then, according to the pattern of tasks resource utilization, a vector was achieved from a combination of the four above-mentioned categories. This vector was calculated by map reduce method, and it formed the queue-based scheduling [27]. Navimpour et al. suggested a method to schedule the jobs on human resources in the expert cloud based on genetic algorithm. In this method, chromosome or candidate solution was represented by a vector; fitness function was calculated based on response time; and one-point crossover and swap mutation were also used. The results indicate that the proposed method could schedule the received jobs in appropriate time with high accuracy in comparison with common methods. Also, the proposed method had better performance in terms of total execution time, service plus wait time, failure rate and human resource utilization rate in comparison with common methods [28]. Li et al. proposed an online optimization for scheduling preemptive tasks on IaaS cloud systems. They recommended two online dynamic resource allocation algorithms for the IaaS cloud system with preemptive tasks. Their algorithms adjust the resource allocation dynamically based on the updated information of actual task executions. Their results showed that their algorithms can significantly improve the performance in the situation where resource contention is fierce [29]. Mezmaz et al. proposed a meta-heuristic for energy-aware scheduling for cloud computing systems. They proposed a new parallel biobjective hybrid genetic algorithm that takes into account makespan and energy consumption. They focused on the island parallel and the multi-start parallel models. Their algorithm was based on dynamic voltage scaling (DVS) to optimize energy consumption. In terms of energy consumption, the

results showed that their approach outperforms the previous scheduling. In terms of makespan, the results were significantly better [30]. In [31], Omara et al. proposed a genetic algorithm-based scheduling for task scheduling problem. They added some heuristics to their genetic algorithm-based scheduling and they obtained better results compared with traditional genetic algorithm-based scheduling.

Abouelela et al. proposed a top-down hierarchical framework for scheduling big data in optical grids. Their method ensures that each domain executes intra-domain scheduling algorithm to schedule its own computing and networking resources. For this framework, they introduced iterative scheduling algorithm and k-shortest paths algorithm. They have carried out some evaluations, and their results showed that their framework with these two algorithms is proper for data-intensive applications [32]. Lin et al. proposed a scheduling algorithm for big data workflows in multicloud environment. Their algorithm, aiming execution cost of workloads minimization, incorporates the concept of partial critical paths, considering quality of service. Their method, considering characteristics of multiclouds, showed better performance than many other comparative algorithms in all different cases. It also showed that the pretreatment procedure has a significant effect on the performance and runtime of proposed algorithm [33]. Somasundaram et al. proposed a swarm intelligence-based scheduling for big data applications. They have used particle swarm optimization-based profiling algorithm to profile applications and, using these profile templates, prefer proper resource list for each submitted big data application. According to these profiles, proper cloud resources from available resources allocate to users' applications. Their evaluations showed that the proposed algorithm maximizes application success ratio, scheduling success rate, utilization of cloud resources and user satisfaction [34].

3 Suggested algorithm

In this section, we first introduce different aspects of the scheduling problem, and then, we will elaborate the proposed algorithm.

3.1 Problem definition

The model used in this paper to describe and solve the scheduling problem is based on the proposed model by Zhang et al. [10] with small changes to make it more compatible with our purposes. Using their predefined parameters, we are able to understand the problem more easily and perform better comparisons. Table 1 shows the most used notation and parameters. Some of the notations used are taken from [10] and others are generated by authors.

In the beginning, we assume that a job (or a workload) is composed of several related tasks, is capable of parallelization and can be scheduled in a virtual cluster consisting of multiple virtual machines. It should be noted that putting interrelated tasks and the tasks with the same type is also feasible. Let C be the number of existing jobs (we assume that job and workload are the same) at the scheduling moment, then $c \epsilon [1.C]$ shows the number of available jobs.

Table 1 Notations and parameters

Notations	Definitions
T_i	Throughput at ith scheduling period
$\delta_c(t)$	The total workload for VC_c at time t
$\underline{\delta}_c(t)$	Remaining workload for VC_c at time t
S_i	The ith scheduling period
c or C	Index of a VC or the number of VCs
VC_c	cth virtual cluster
VM_k	kth virtual machine
i or I	Index of time or the last scheduling point
$\theta_c(t_i)$	Number of VMs in VC_c allocated at time t_i for S_i stage
θ	The number of available VMs
R_{δ_c}	The workload required version of δ_c
LFN_i	Local file name for the ith replica
TC_{δ_c}	The time cost for sending R_{δ_c} to VC_c
$M_{i,j}$	The number of messages between the clusters VC_i, VC_j
B_{ci}	The bandwidth of sending LFN_i to cluster c

Virtual machines (VMs) are computational units that are placed on top of the physical layer. The number of VMs during the scheduling is supposed to be fixed, and the employed scheduling algorithm categorizes VMs in all of C groups. Each group called virtual cluster (VC) serves a job. All the tasks of the cth job are assigned to cth VC. As a result, we should divide VMs to C VCs [10].

According to Fig. 1, the workload dispatcher sends tasks to VCs to run using the information of the similarity calculation and the scheduling improver sectors.

The number of related VMs for each VCs is different in any time interval. The purpose of the new algorithm is to find the optimal number of VMs related to each virtual cluster in order to improve the throughput and makespan to an acceptable level. Let $\theta(t_i)$ be the scheduling in i time interval, then its value is a vector by C element that each element indicates the number of clusters corresponding to that number of VMs as shown in Eq. (1)

$$\theta(t_i) = [\theta_1(t_i), \theta_2(t_i), \ldots, \theta_c(t_i), \ldots, \theta_C(t_i)] \tag{1}$$

where $\theta_c(t_i)$ is the number of used VMs in the cth VC, θ is the total number of available VMs, and $\theta(t_i)$ is the order which shows the number of virtual machines per cluster. The goal is to find the best order for any time interval where: $\sum_{c=1}^{C} \theta_c(t_i) = \theta$.

If at the time interval t_i, $\theta(t_i)$ scheduling is employed, this strategy will be valid until t_{i+1}. At the next time point (t_{i+1}) another scheduling vector will be used. The new scheduling vector, $\theta(t_{i+1})$, is generated during the simulation phase from t_i to t_{i+1}. In the proposed algorithm, we try to use the $\theta(t_i)$ feedback from t_i to t_{i+1} for generating $\theta(t_{i+1})$. In addition, job similarities are used for optimizing the scheduling algorithm. In other words, if the similarity between workloads from t_i to t_{i+1} is acceptable, then the same approach will be used and $\theta(t_i) = \theta(t_{i+1})$. The same strategy will help to

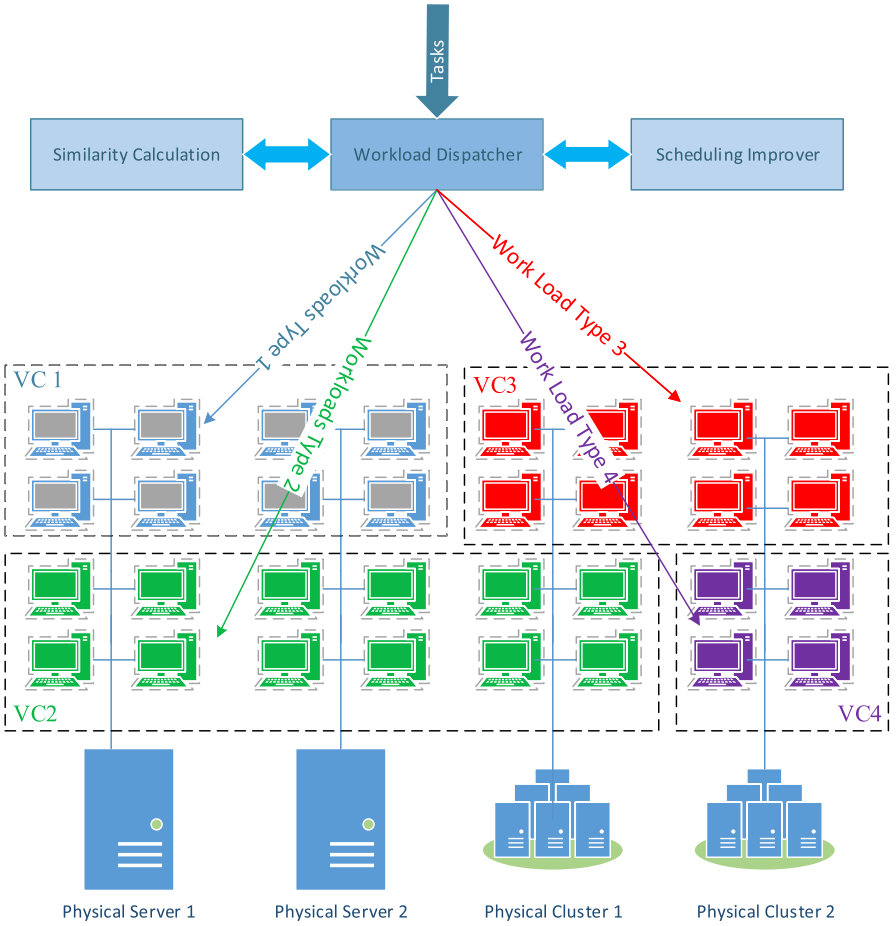


Fig. 1 The proposed cloud platform is made up of two physical servers and two physical clusters each of which is divided into eight virtual machines organized in virtual clusters. For example, in VC_1 there are eight virtual machines: Four VMs are located at physical server 1 and other four VMs are placed on physical server 2. The order will be changed during the scheduling

have less scheduling overhead between two time intervals. Otherwise, the scheduling vector should be renewed. A new scheduling vector $\theta(t_{i+1})$ is generated with n iteration manner as it was explained before.

If $\delta_c(t)$ is the workload of related cth virtual cluster at time t, the workload is executed at the end of the time interval or before its completion; otherwise, the remaining amount will be transferred to the next time interval. The remaining amount of workload is shown with $\underline{\delta}_c(t)$. Time of completion of each cluster in each time interval will be calculated separately (if a workload will not be complete at the end of a time period, t_i, t_i is considered as its completion time).

The "throughput" criterion is used for different scheduling measurements in each time interval. Throughput is the number of tasks carried out in the time unit. A through-

put is calculated for each time interval. Throughput for the time interval i is shown as T_i. This throughput is calculated for all clusters in a time interval.

Considering $\theta(t_i) = [\theta_1(t_i), \theta_2(t_i), \ldots, \theta_c(t_i), \ldots, \theta_C(t_i)]$ scheduling, task throughput is the total number of completed tasks divided by the total time devoted to running tasks.

3.2 The introduction of the proposed technique

According to Fig. 2, the proposed algorithm is divided into the following general components:

1. Primary scheduling (primary calculation of the number of virtual machines per cluster) phase.
2. Similarity calculation phase.
3. Scheduling improvement phase.

After the primary scheduling, which repeats n times (n is much smaller than the entire condition space), based on the best throughput, one of the n random modes is selected as the scheduling strategy. Similarities of newly entered jobs are compared with completed jobs during and after each time interval (time interval duration will be defined based on the usage). If the similarity is more than a threshold, continuing the scheduling in accordance with the previous interval is feasible; otherwise, by involving the information of scheduling improvement level, another time scheduling should be employed. Creating the new scheduling vector starts by n time random iterations. It chooses the best created θ among the n random generated vectors, based on the throughput. Also, the probable cluster weight changes according to the feedback from the previous time interval and the importance of features, such as load balancing and replication. The clusters with more loads absorb more VMs in the next step and have more calculation power for new courses. In the following, we delineate each stage of the above-mentioned algorithm.

3.2.1 Primary scheduling phase

In this phase, the number of VCs is determined based on the number of initial workloads. Then, the virtual machines are equally distributed among the existing clusters to achieve $\theta(t_0) = [\theta_1(t_0), \theta_2(t_0), \ldots, \theta_c(t_0), \ldots, \theta_C(t_0)]$ vector; random choosing is based on a random number which at first is constant for each cluster and equal to $\frac{1}{C}$. But it may change during the simulation. If the estimated amount of a cluster is higher, the possibility of assigning more VMs to it is stronger. Then, we calculate throughput for this scheduling and repeat it n times, and the scheduling with the best throughput will be selected as the primary scheduling and workloads will be assigned to their corresponding VCs.

3.2.2 Similarity calculation phase

This phase has three parts and similar workloads are calculated from three different perspectives. If the similarity is lower than a threshold, the scheduling vector should

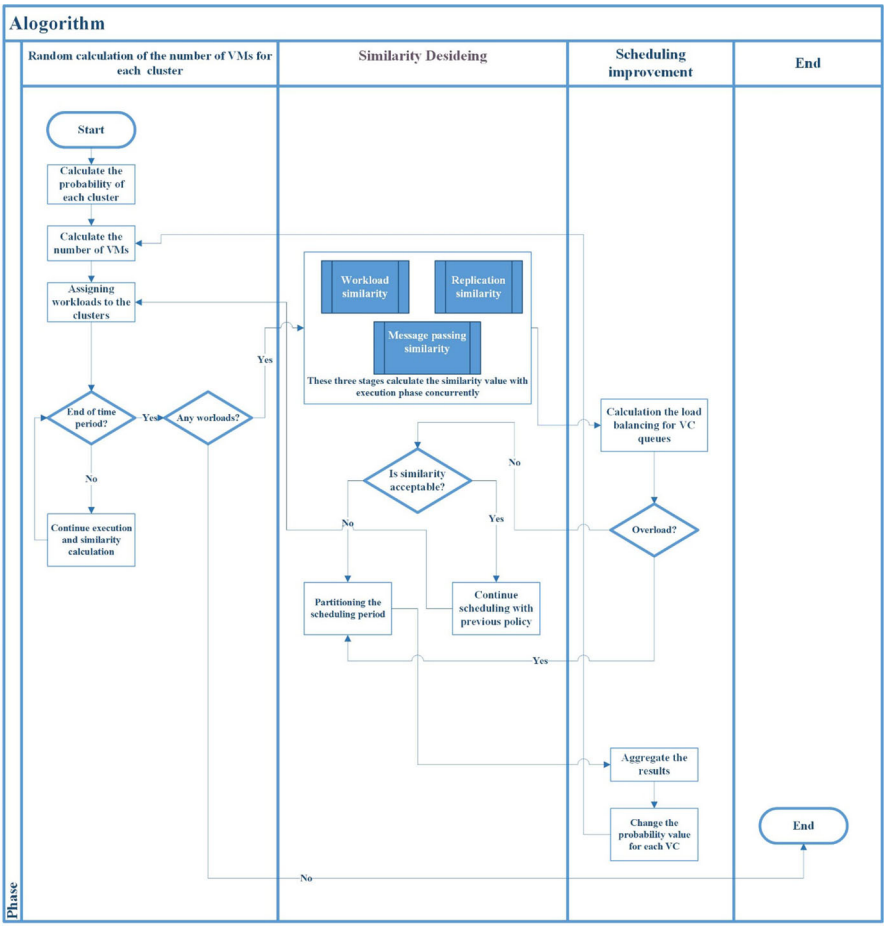


Fig. 2 The flowchart of the proposed algorithm phases

be calculated again; otherwise, we can continue the previous algorithm. To calculate resemblance to the workloads more accurately, the following three criteria are employed:

- Similarity in size.
- Similarity in replication.
- Similarity in message passing.

Using a good similarity calculation, we can achieve better comparison between two time periods. As creating new scheduling has some overhead, it is desirable to decrease these procedures. On the other hand, if similarity standard is more accurate, deciding about creating a new scheduling would be more precise. In the other words, if workloads of two-consequence time periods are similar to a more precise standard, changing scheduler would take place in right place, and in addition to time reduction of running workloads with a proper scheduler, the overhead of scheduling would be lesser

due to the reduction of changing scheduler. In the following, each part is explained in detail.

Similarity in the size of workloads

Similarity in the size is shown by α parameter, which is extracted from [10], where it is the only criterion for calculation of workloads similarities. In the current paper, α is one of the parameters that aim at determining the similarity of workloads in different time intervals. This parameter is calculated through Eq. (2)

$$\alpha(t_1, t_2) = \frac{\sum_{c=1}^{C} (\delta_c(t_1) \times \delta_c(t_2))}{\sqrt{\sum_{c=1}^{C} \delta_c(t_1)^2 \times \sum_{c=1}^{C} \delta_c(t_2)^2}} \tag{2}$$

where $\delta_c(t)$ is the volume of workloads in cluster c at time t.

Similarity in required replicas to do one workload in two time periods t_1 and t_2

If the settled workload in the cluster C needs to run the data in other clusters, the transfer cost of these files will be an influential factor to cluster c. This cost can be calculated in two time intervals and considered as a measure of similarity. It means that if the cost of the replica in time t_1 is not discriminated from the cost of time t_2, it can be concluded that there was not a large fluctuation in workloads and scheduling algorithm did not change. These factors will be used to improve the scheduling as well.

If the workload requires a version of δ_c which is shown as R_{δ_c} and LFN_i refers to the local file name for the ith replica, then:

$$R_{\delta_c} = \{\text{LFN}_1, \text{LFN}_2, \ldots, \text{LFN}_n\} \tag{3}$$

If the LFN_i exists in cluster c, there will be no cost; otherwise, the cost is calculated by Eq. (4)

$$\text{TC}_{\delta_c} = \frac{1}{\text{par}_c} \times \sum_{i=1}^{n} \frac{|\text{LFN}_i|}{B_{ci}} \tag{4}$$

par_c is a fixed ratio for showing the amount of parallelism degree to download the replica in cluster c. B_{ci} is the bandwidth of sending LFN_i to cluster c. The parameter β is defined as a similarity in replica as shown in Eq. (5)

$$\beta(t_1, t_2) = \frac{\sum_{c=1}^{C} \left(\text{TC}_{\delta_c}(t_1) \times \text{TC}_{\delta_c}(t_2)\right)}{\sqrt{\sum_{c=1}^{C} \text{TC}_{\delta_c}(t_1)^2 \times \sum_{c=1}^{C} \text{TC}_{\delta_c}(t_2)^2}}. \tag{5}$$

Message passing among clusters

The number of sent messages between two clusters is a criterion for the similarity in two time intervals. Figure 3 shows the hypothetical example of communication graph with four clusters. Nodes are clusters and edges indicate the number of messages that are exchanged between two clusters.

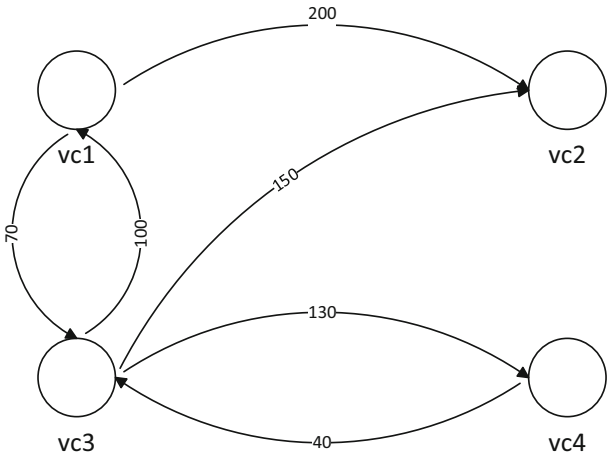


Fig. 3 An example for message passing among VCs. *Asterisk* Weights of edges show the number of commuted messages

Let $M_{i,j}$ be the summation of the number of messages between the clusters $\mathrm{VC}_i, \mathrm{VC}_j$, then $M_{i,j} = M_{j,i}$ and there is no need to calculate $M_{j,i}$ separately. It seems calculation of similarity in t_1 and t_2 intervals in terms of exchanging messages between clusters is sufficient from the viewpoint of accuracy. In this case, let C be the total number of clusters corresponding to graph matrix element, and C^2 be exchanged messages; with the proposed algorithm, calculating the similarity in messaging has the complexity of $O\left(\frac{C(C-1)}{2}\right)$ instead of $O\left(C^2\right)$.

γ is used as the degree of similarity of the time intervals t_1 and t_2, which is calculated by Eq. (6)

$$\gamma(t_1, t_2) = \left(\sum_{i=1}^{C} \sum_{j=1}^{i} \frac{\left| M_{ij}(t_1) - M_{ij}(t_2) \right|}{M_{ij}(t_1) + M_{ij}(t_2)} \right) / (C(C+1)/2) \tag{6}$$

$M_{ij}(t)$ is the total number of messages from cluster i to cluster j and vice versa at time t.

When $M_{ij}(t_1) = M_{ij}(t_2) = 0$, there were no exchanged messages between the two clusters and the amount of fraction is considered one in that repetition.

Finally, the similarity is defined as shown in Eq. (7)

$$\begin{aligned} \mathrm{Sim}\ (t_1, t_2) &= w_1 \times \alpha(t_1, t_2) + w_2 \times \beta(t_1, t_2) + w_3 \times \gamma(t_1, t_2) \\ &\text{where } \textstyle\sum_{i=1}^{3} w_i = 1. \end{aligned} \tag{7}$$

According to the calculated similarity, the key decision could be made on changing the scheduling policy in the next time interval or continuing the last policy with no change. If the $\mathrm{Sim}(t_1, t_2)$ is less than a specific threshold, then the scheduling order should be changed. Due to this change, θ should be calculated with n time repetition

and selecting the best θ based on the throughput. When the scheduler decides to change the scheduling order, it must divide the scheduling interval adaptively into smaller intervals for separating the scheduling. On the other hand, if the similarity is in an acceptable range of values, it is not necessary to change the scheduling policy and two scheduling intervals could be merged to continue scheduling with the least order.

3.2.3 Improving scheduling using load balancing phase

At the moment of time interval completion or when the interval is broken into two parts, the random assignment of machines into clusters is done several times, and the scheduling with the best throughput is chosen. It is obvious that such random operations which have been conducted up to now do not involve the existing feedback in the next time interval of scheduling.

In this section, we try to perform the scheduling with respect to the existing status to make improvements in the next scheduling tasks. This is done while running workloads. It means that in order to enhance the running speed and not to stop running works for the reason of simulation actions, simulation stage will be done parallel with execution stage. In the following, a criterion is defined for considering the load of each cluster.

As it was mentioned, selecting the number of machines per cluster at the beginning stage is a random process and most probably equal to $\frac{1}{C}$. At each step, we attempt to distribute the number of virtual machines based on improvement of stages targeted. That is, if workload of one cluster queue is more than the average of workload queues, it probably will be enhanced in the way that in the next scheduling interval there exists more chance for assigning VMs to that VC. As illustrated in Fig. 4, each cluster has a queue. The entered workloads for each cluster are sent to a specific queue.

To balance the workload of each cluster, deviation amount of waiting works in the queue is calculated and the probability ratio will be changed for each cluster according to the deviation

- $q_\delta(t)$ total volume of workloads in all queues at time t
- $\bar{q}_\delta(t)$ average volume of workloads in all queues
- $q_{\delta_c}(t)$ volume of workload in the cluster queue c at time t.

Standard deviation is calculated for the volume of workload in queue. If the deviation is less than a certain value, it does not need to take other steps and the algorithm will go to the last step (similarity calculation); otherwise, the scheduling policy should be changed to achieve more optimized load balancing. Therefore, the following steps are executed:

First, the standard deviation should be calculated through Eq. (8)

$$\sigma = \sqrt{\frac{1}{C}\sum_{c=1}^{C}\left(q_{\delta_c}(t) - \bar{q}_\delta(t)\right)^2}. \tag{8}$$

Second, if σ is more than a certain threshold, the scheduling policy should be changed. Finding suitable threshold is done by trial and error.

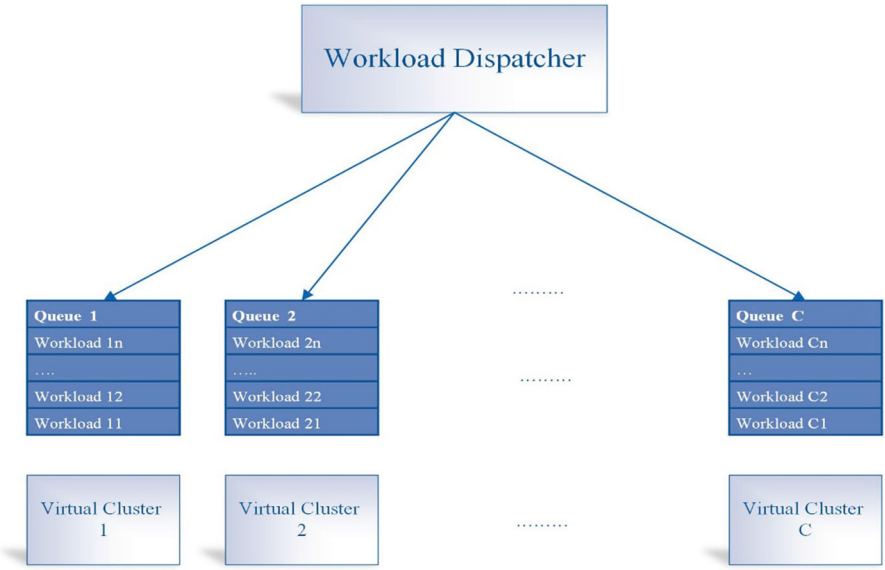


Fig. 4 Queues of VCs, which contain workloads of similar types

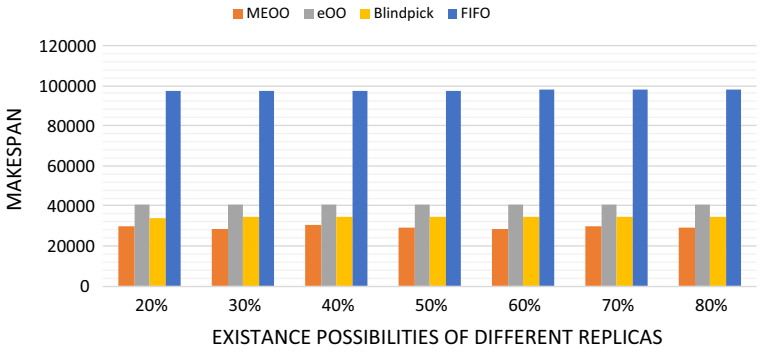
Table 2 Details of workloads

Jobs	Number of tasks	Total length (no. of instructions)	Size of needed file (if exist) (MB)
Job 1	6	24,000	300
Job 2	6	82,000	300
Job 3	3	360,000	300
Job 4	4	80,000	300
Job 5	2	70,000	300
Job 6	10	38,000	300
Job 7	8	400,000	300

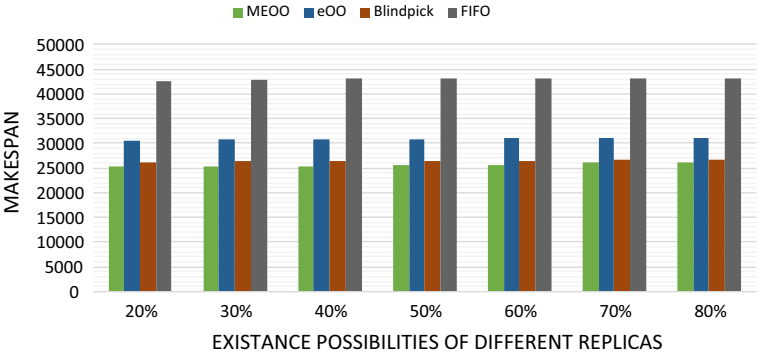
Third, the probability of clusters should be calculated again. For each c, if $p_c(t)$ is the eventual ratio for each virtual machine assignment for cluster c at time t, it can change over the time as shown in Eq. (9)

$$\begin{aligned} &\forall c \in [1, C], (q_{\delta_c}(t) - \bar{q}_\delta(t)) = \varphi_c \\ &\varphi_c \begin{cases} >0 \rightarrow p_c(t) = p_c(t) + \frac{q_{\delta_c}(t)}{q_\delta(t)} \\ <0 \rightarrow p_c(t) = p_c(t) - \frac{q_{\delta_c}(t)}{q_\delta(t)} \\ =0 \rightarrow p_c(t) = p_c(t) \end{cases} . \end{aligned} \tag{9}$$

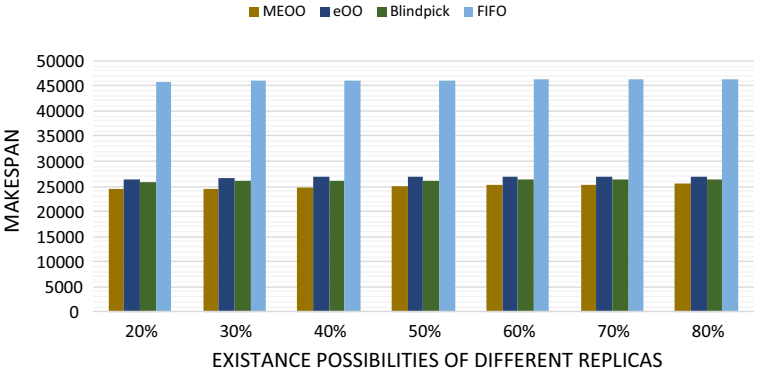
However, if $p_c(t)$ is more than one or less than zero, this step will fail. By changing clusters probability in the next scheduling interval, the number of machines will change according to the targeted probability. In fact, we aim at raising the possibility of the



(a)



(b)



(c)

Fig. 5 Makespan of running simulation of BIG scenario for existence possibilities of different replicas, **a** 32 virtual machines, **b** 64 virtual machines, **c** 128 virtual machines

busier clusters to privilege higher chances for absorbing more VMs, and also we decrease the possibility of clusters the queues of which are less congested. In this way, we reach load balancing for VCs over time.

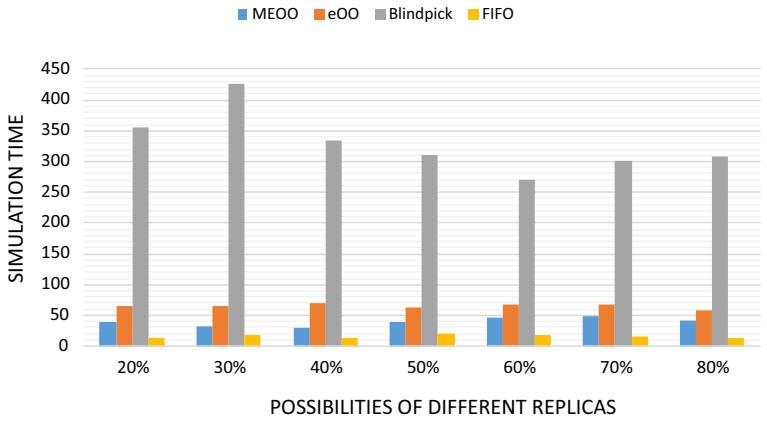


Fig. 6 Simulation time for 64 virtual machines for possibilities of different replicas

4 Results and discussion

In this section, we present detailed results for the aforementioned multitasking workload scheduling.

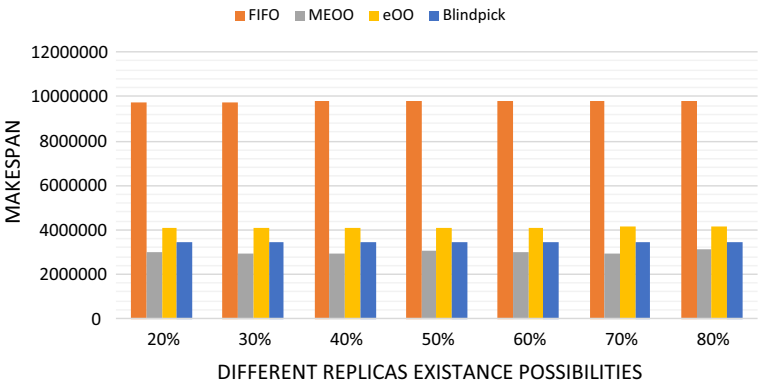
The simulations are carried out on a PC with the Intel Dual Core CPU with 2 GHz frequency, 3 Gigabytes of Ram and a 32-bit windows 7 operating system. This simulation is performed by CloudSim version 3.0 and Java Development Kit 1.7.0 in NetBeans IDE 8.0.2.

We utilized seven clusters for distributing the nodes. In three scenarios, we used 32, 64 and 128 nodes. Also, a data center was utilized, which comprised of four servers each of which had eight CPUs (2.0 GHz), 8 GB of Ram and a 32-bit Linux system. Each worker node had one CPU (1.0 GHz), 512-MB Ram and a 32-bit Linux system. Workers were interconnected by a connection with a 1000-Mbit bandwidth.

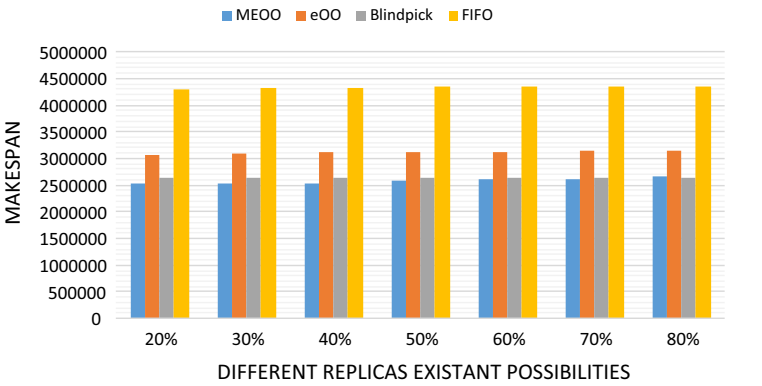
Since we used the simulation method for evaluation, we evaluated our scheduler using jobs generated by ourselves. It was aimed to mime the benchmarking applications that were used in [10]. Since there were seven clusters, there were seven types of jobs as well. In our simulations, we had one thousand jobs that were randomly generated. But, for each simulation, the seed of random generator was identical to others in order to have the same order of jobs to provide a fair simulation. Details of jobs are shown in Table 2.

According to Table 2, there are seven jobs (workloads), each of which has a special number of tasks (cloudlets), length and file size. For example, job 1 has six tasks with a total length of 24,000 million instructions, and if it needs a file for replication, the file size will be 300 MB. It must be noted that CloudSim is not a tool for big data, we mime the behavior of the big data by extending the total length of jobs. Actually, we created jobs with more instructions by this means and, as a result, jobs take more time for the operation.

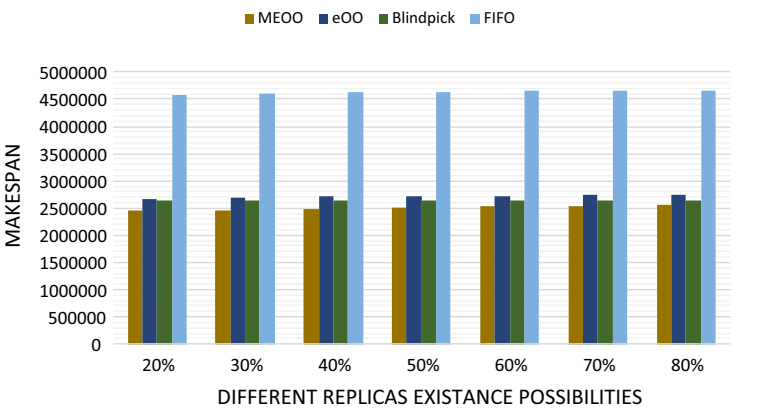
We have designed some scenarios for a proper validation and comparison according to [10]. As the current paper is focused on big data, we have designed three comprehensive scenarios that in each of them file sizes are different, to test methods in the



(a)



(b)



(c)

Fig. 7 Makespan of running simulation for VBIG scenario for existence possibilities of different replicas, **a** 32 virtual machines, **b** 64 virtual machines, **c** 128 virtual machines

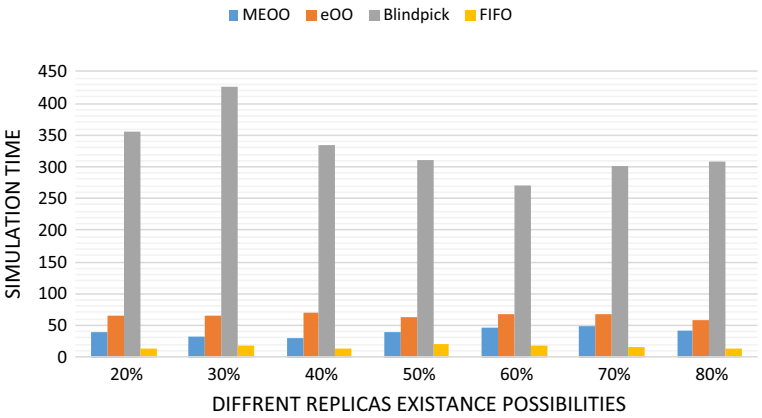


Fig. 8 Simulation time for 64 virtual machines for different possibilities of replicas

situations that workloads are big, very big and huge. On the other hand, as replication and message passing is one of the other important parameters, we have tested all methods in all scenarios with a different probability of message passing existence among virtual clusters.

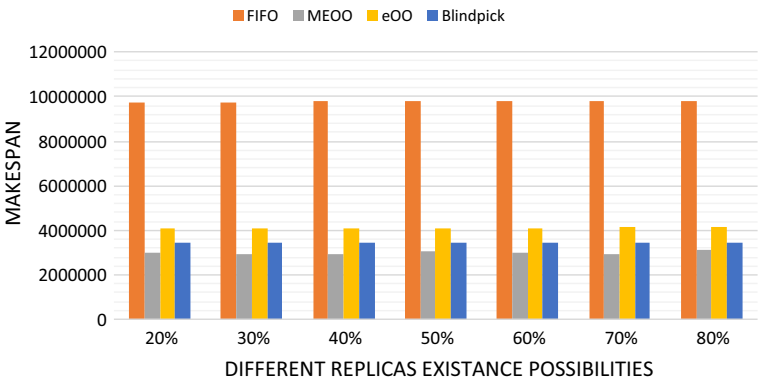
Simulation is performed in three scenarios; Scenario 1: We used the numbers that are presented in Table 2 as the boundaries of our experiment and called it "Big workflow generation" or "BIG." Scenario 2: We multiplied the total length of each task and file size by 10 and named it "Workflow generation" or "VBIG." And Scenario 3: We multiplied the total length of each task and file size by 100 and called it "Huge workflow generation" or "HUGE."

The results of each scenario were compared for 32, 64 and 128 nodes. For each of them, the simulation was run for existence possibility of different replicas. In effect, the proposed algorithm was tested in situations where possibilities of needing replications were 0.2, 0.3, 0.4, 0.5, 0.6, 0.7 and 0.8. We have called proposed method as *multi-parameter evolutionary ordinal optimization (MEOO)* and it is compared with eOO, BlindPick and FIFO methods.

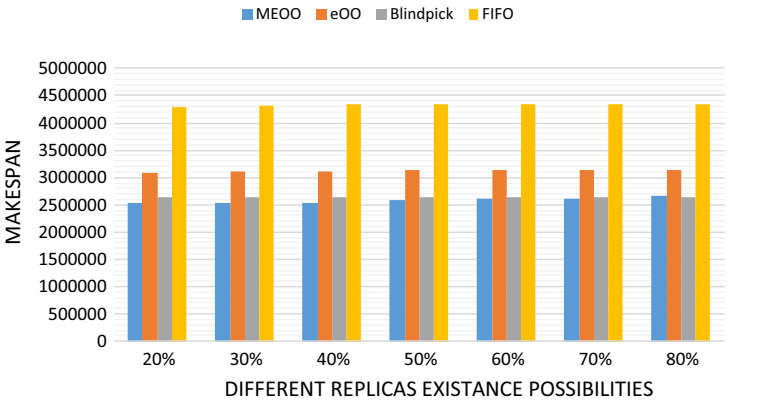
Scenario 1: Big workflow generation (BIG)

Figure 5 indicates the comparison of Scenario 1 for different replicas possibility for 32 (5.a), 64 (5.b) and 128 (5.c) virtual machines.

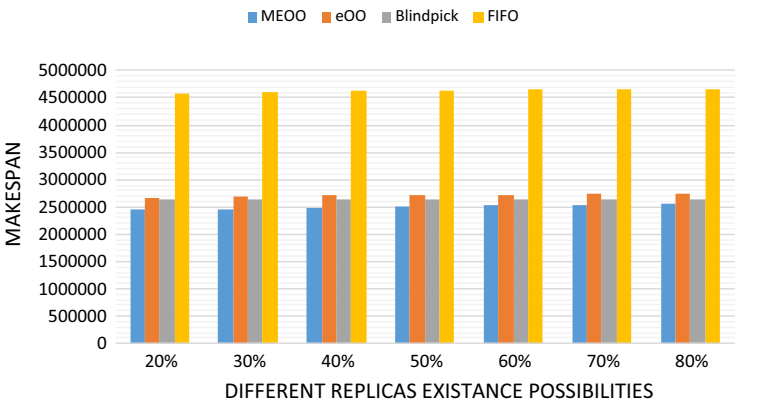
It was observed that the variation of existence possibility of replications has a very limited impact on all of the tested methods, because the size of replications is very smaller than the actual size of running workloads. As the replication rate raises, we can see better improvement in proposed method. In Fig. 5 we can see the improvement of our method in terms of makespan. After rising replica existence ratio from 0.2 to 0.8, in comparison with eOO, BlindPick and first in first out (FIFO), the improvement is raised from 0.25, 0.11 and 0.69 to 0.29, 015 and 0.70, respectively. FIFO always had the worst and the proposed algorithm (MEOO) always had the best makespan among the tested algorithms. By increasing the numbers of VMs, performance was improved in all of them. In Fig. 5 we can see that in terms of makespan, our method



(a)



(b)



(c)

Fig. 9 Makespan of running simulation for HUGE scenario for each existence possibility of different replicas, **a** 32 virtual machines, **b** 64 virtual machines, **c** 128 virtual machines

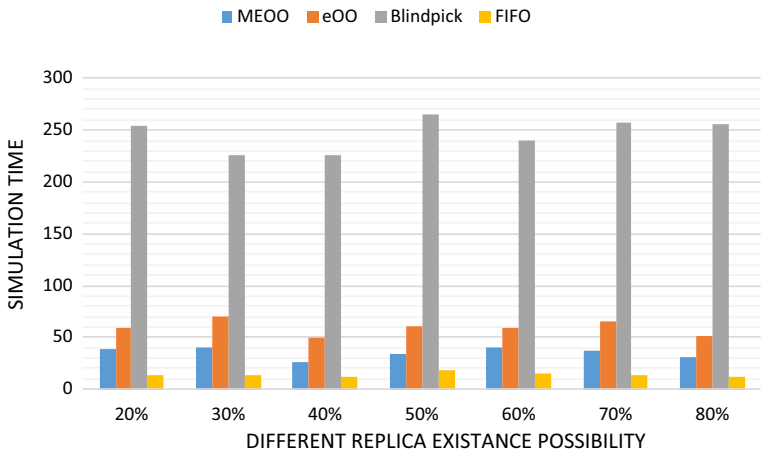


Fig. 10 Simulation time for 64 virtual machines for different possibilities of replicas

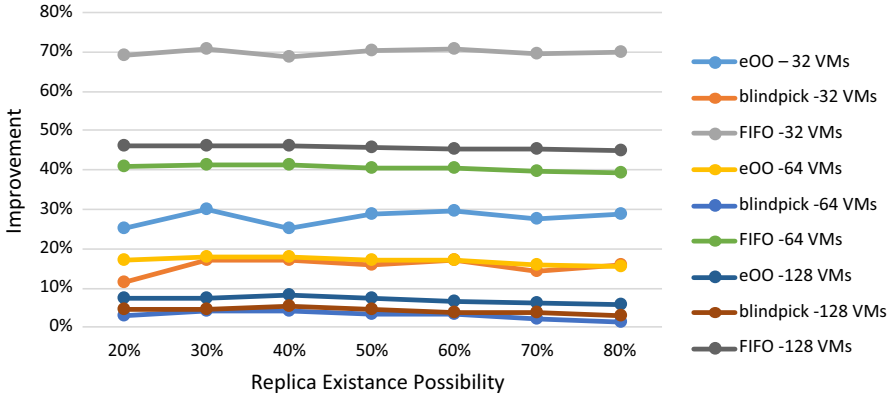


Fig. 11 Performance improvement of the proposed algorithm compared to the others in BIG scenarios

shows better results in comparison with other methods. In some cases, the proposed algorithm and BlindPick had almost similar performance, but as it is shown in Fig. 6, the proposed algorithm had much less simulation time and consequently a much less overhead. According to Fig. 6, BlindPick had a very bad overhead, although it had nearly the same makespan as the proposed algorithm in some situations. For the sake of brevity, Fig. 6 just depicts the results for 64 VMs; other results for 32 and 128 VMs are similar to 64.

Scenario 2: Very big workflow generation (VBIG)

Figure 7 indicates the comparison of Scenario 2 for different replicas possibility for 32 (7.a), 64 (7.b) and 128 (7.c) virtual machines.

Like Scenario 1, the FIFO algorithm always had the worst and the proposed algorithm (MEOO) always had the best makespan among the tested ones. By increasing the numbers of VMs, all algorithms worked better. In some situations, the performance

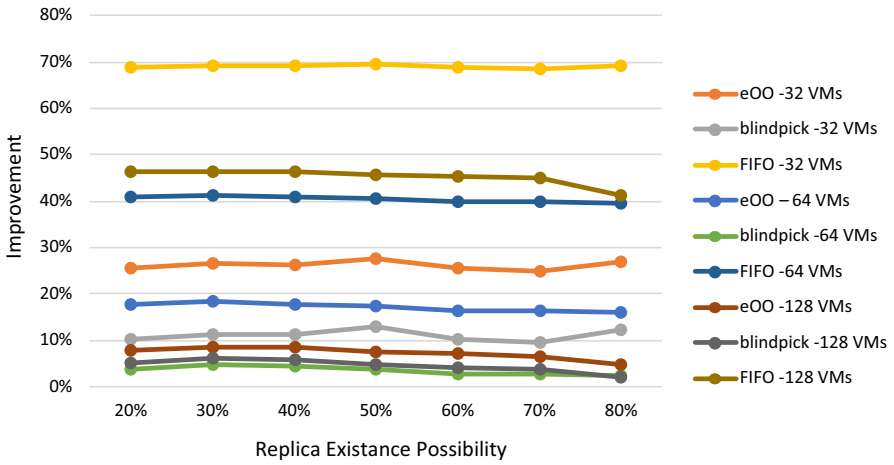


Fig. 12 Performance improvement of the proposed algorithm compared to the others in VERY BIG scenarios

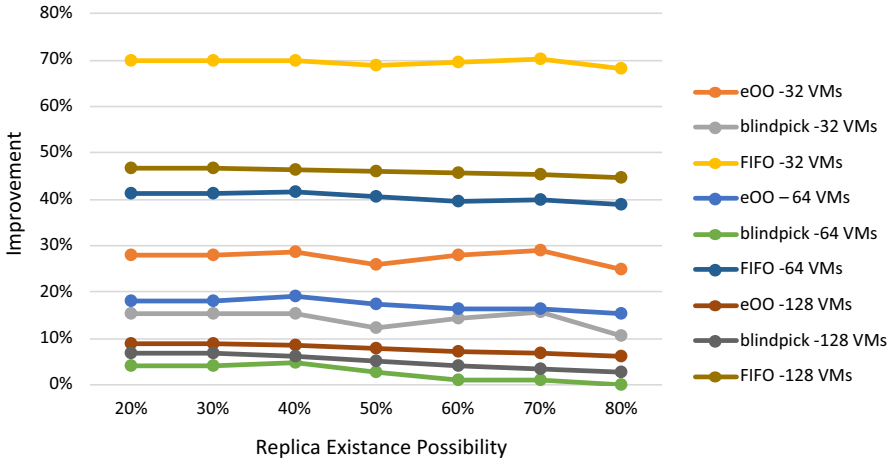


Fig. 13 Performance improvement of the proposed algorithm compared to the others in HUGE scenarios

of the proposed algorithm and BlindPick was analogous, but according to Fig. 8 the proposed algorithm had much less simulation time and, as a result, it had a much less overhead. For the sake of brevity, Fig. 8 only illustrates the results for 64 VMs; other results for 32 and 128 VMs are similar to 64.

Scenario 3: Huge workflow generation (HUGE)

Figure 9 indicates the comparison of Scenario 3 for different replicas possibility for 32 (9.a), 64 (9.b) and 128 (9.c) virtual machines.

Likewise, the FIFO algorithm was always the worst and the proposed one (MEOO) outperformed the others. Also, increasing the numbers of VMs caused all algorithms to

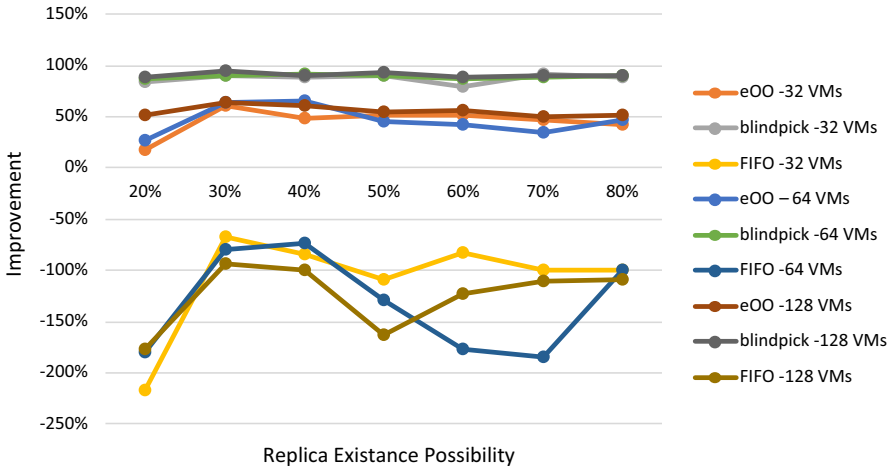


Fig. 14 Overhead reduction of the proposed algorithm compared to the others in BIG scenarios

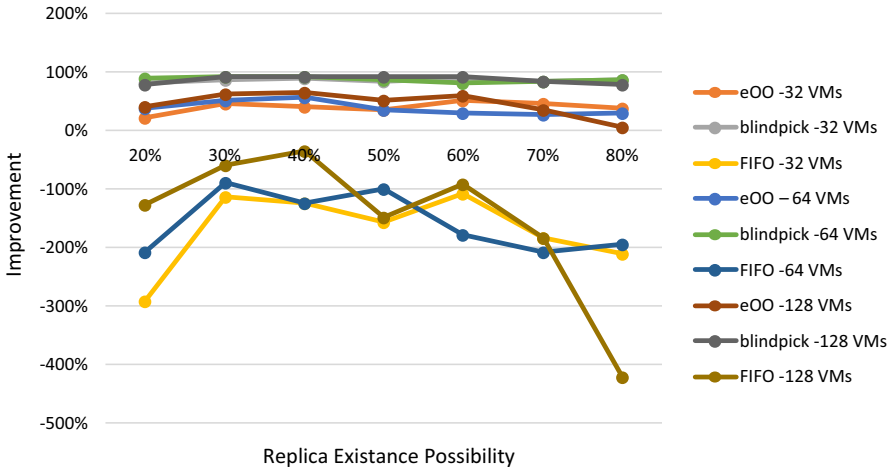


Fig. 15 Overhead reduction of the proposed algorithm compared to the others in VBIG scenarios

have better performance. As shown in Fig. 10, the overhead of the proposed algorithm is much less than BlindPick in similar situations.

In all scenarios, the proposed algorithm outperformed the others and it had a better makespan. The improvements in Scenarios 1, 2 and 3 are shown in Figs. 11, 12 and 13, respectively. These charts are shown as three tables in "Appendix 1" section.

In all scenarios and in all situations with different replications, the proposed algorithm works better than the others. Overall, MEOO had 45% speedup in comparison with FIFO, 3–10% with BlindPick and 5–10% with eOO. Rising the amount of replication existence causes better performance. As these replications are small in comparison with these big data, its influence is about 1%.

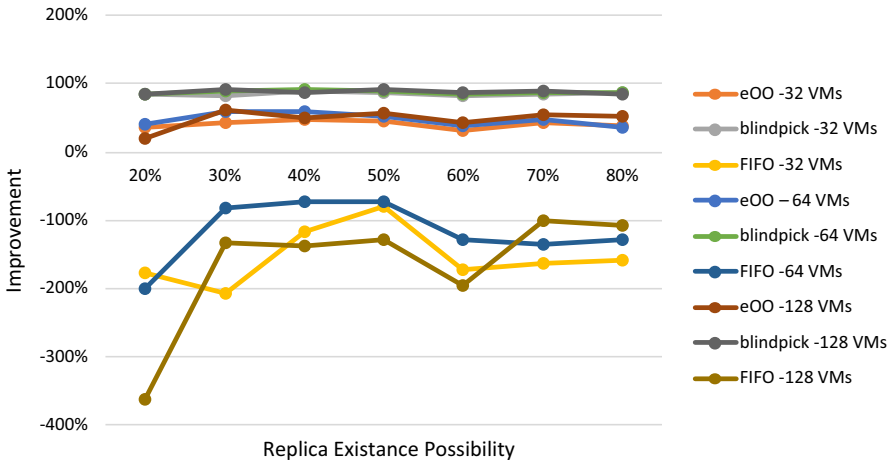


Fig. 16 Overhead reduction of the proposed algorithm compared to the others in HUGE scenarios

Simulation time can be supposed as overhead. In Figs. 14, 15 and 16, the overhead of the proposed algorithm is compared to the other tested ones. Totally, MEOO had 80–90 and 5% overhead time reduction compared to BlindPick and eOO, respectively. But FIFO had less overhead time because of its simplicity.

5 Conclusion

A minimal-overhead algorithm for dynamic multitasking workload scheduling is proposed in this work. This algorithm is based on allocating virtual cluster resources on demand. The most important advantage of MEOO algorithm is impressively decreasing the overhead in creating suboptimal schedules. On the other hand, the major technical contribution of this study is the acceptable performance of MEOO in all situations with different existence possibilities of replications, a different number of virtual machines and different data sizes. MEOO was tested with prominent extant methods through simulation. The results in different scenarios and situations showed that proposed method achieve less makespan and as a result more throughput. It should be mentioned that MEOO always has reached better makespan and performance in comparison with other methods, and its overhead is always much less than the other methods, except FIFO. The better throughput and makespan in cloud computing can cause many improvements in cloud indirectly; consumers' costs, efficient use of resources, increased outcome for service providers and energy saving are just some of these improvements. As a suggestion for further research, we recommend the use of statistical analysis for determining similarity. Furthermore, we strongly suggest using queue theory for managing queues in order to make a probable improvement.

Appendix 1: Performance improvement of proposed method

See Tables 3, 4 and 5.

Table 3 Performance improvement of the proposed algorithm compared to the others in BIG scenarios

Algorithm	20%	30%	40%	50%	60%	70%	80%
eOO-32 VMs	0.253	0.298	0.253	0.287	0.297	0.275	0.287
BlindPick-32 VMs	0.115	0.169	0.172	0.157	0.169	0.144	0.157
FIFO-32 VMs	0.690	0.707	0.688	0.702	0.706	0.696	0.701
eOO-64 VMs	0.170	0.179	0.180	0.172	0.170	0.160	0.155
BlindPick-64 VMs	0.029	0.041	0.042	0.033	0.032	0.020	0.014
FIFO-64 VMs	0.407	0.413	0.412	0.406	0.405	0.397	0.393
eOO-128 VMs	0.072	0.072	0.081	0.072	0.065	0.061	0.056
BlindPick-128 VMs	0.046	0.046	0.053	0.044	0.039	0.036	0.029
FIFO-128 VMs	0.463	0.463	0.463	0.458	0.454	0.452	0.448

Table 4 Performance improvement of the proposed algorithm compared to the others in VERY BIG scenarios

Algorithm	20%	30%	40%	50%	60%	70%	80%
eOO-32 VMs	0.256	0.265	0.263	0.276	0.255	0.249	0.271
BlindPick-32 VMs	0.101	0.113	0.112	0.129	0.103	0.096	0.122
FIFO-32 VMs	0.689	0.692	0.691	0.696	0.687	0.684	0.693
eOO-64 VMs	0.177	0.185	0.179	0.175	0.165	0.165	0.161
BlindPick-64 VMs	0.038	0.048	0.043	0.038	0.027	0.027	0.023
FIFO-64 VMs	0.410	0.414	0.410	0.406	0.398	0.398	0.395
eOO-128 VMs	0.078	0.085	0.084	0.074	0.071	0.064	0.048
BlindPick-128 VMs	0.052	0.060	0.059	0.049	0.040	0.039	0.020
FIFO-128 VMs	0.463	0.465	0.463	0.457	0.452	0.450	0.412

Table 5 Performance improvement of the proposed algorithm compared to the others in HUGE scenarios

Algorithm	20%	30%	40%	50%	60%	70%	80%
eOO-32 VMs	0.281	0.281	0.286	0.260	0.278	0.291	0.248
BlindPick-32 VMs	0.152	0.152	0.155	0.122	0.143	0.156	0.105
FIFO-32 VMs	0.699	0.699	0.700	0.689	0.696	0.701	0.683
eOO-64 VMs	0.181	0.181	0.190	0.174	0.164	0.165	0.153
BlindPick-64 VMs	0.042	0.042	0.047	0.026	0.011	0.010	0.002
FIFO-64 VMs	0.412	0.412	0.417	0.405	0.397	0.398	0.389
eOO-128 VMs	0.089	0.089	0.087	0.079	0.073	0.069	0.062
BlindPick-128 VMs	0.069	0.069	0.062	0.051	0.042	0.035	0.028
FIFO-128 VMs	0.467	0.467	0.465	0.460	0.455	0.452	0.448

Appendix 2: Overhead reduction of proposed algorithm

See Tables 6, 7 and 8.

Table 6 Overhead reduction of the proposed algorithm compared to the others in BIG scenarios

Algorithm	20%	30%	40%	50%	60%	70%	80%
eOO-32 VMs	0.173	0.607	0.489	0.520	0.511	0.463	0.428
BlindPick-32 VMs	0.844	0.904	0.887	0.904	0.794	0.911	0.889
FIFO-32 VMs	−2.166	−0.666	−0.846	−1.090	−0.833	−1.000	−1.000
eOO-64 VMs	0.275	0.635	0.648	0.457	0.419	0.339	0.473
BlindPick-64 VMs	0.865	0.908	0.913	0.904	0.871	0.881	0.903
FIFO-64 VMs	−1.800	−0.800	−0.733	−1.285	−1.769	−1.846	−1.000
eOO-128 VMs	0.514	0.633	0.612	0.548	0.565	0.500	0.511
BlindPick-128 VMs	0.883	0.945	0.908	0.928	0.888	0.906	0.903
FIFO-128 VMs	−1.77	−0.941	−1.000	−1.625	−1.222	−1.100	−1.095

Table 7 Overhead reduction of the proposed algorithm compared to the others in VBIG scenarios

Algorithm	20%	30%	40%	50%	60%	70%	80%
eOO-32 VMs	0.232	0.475	0.408	0.368	0.509	0.465	0.381
BlindPick-32 VMs	0.811	0.864	0.884	0.854	0.889	0.853	0.864
FIFO-32 VMs	−2.909	−1.133	−1.230	−1.571	−1.076	−1.818	−2.090
eOO-64 VMs	0.393	0.507	0.585	0.354	0.308	0.268	0.305
BlindPick-64 VMs	0.887	0.924	0.913	0.871	0.826	0.836	0.867
FIFO-64 VMs	−2.076	−0.882	−1.230	−1.000	−1.764	−2.062	−1.928
eOO-128 VMs	0.406	0.623	0.653	0.530	0.611	0.360	0.048
BlindPick-128 VMs	0.796	0.933	0.929	0.916	0.912	0.846	0.785
FIFO-128 VMs	−1.250	−0.590	−0.346	−1.473	−0.904	−1.818	−4.210

Table 8 Overhead reduction of the proposed algorithm compared to the others in HUGE scenarios

Algorithm	20%	30%	40%	50%	60%	70%	80%
eOO-32 VMs	0.35	0.428	0.480	0.442	0.305	0.430	0.392
BlindPick-32 VMs	0.846	0.823	0.884	0.872	0.829	0.856	0.878
FIFO-32 VMs	−1.78	−2.076	−1.166	−0.789	−1.733	−1.642	−1.583
eOO-64 VMs	0.407	0.592	0.582	0.512	0.392	0.466	0.369
BlindPick-64 VMs	0.853	0.903	0.909	0.883	0.838	0.879	0.876
FIFO-64 VMs	−2	−0.823	−0.736	−0.727	−1.285	−1.352	−1.277
eOO-128 VMs	0.195	0.614	0.494	0.567	0.425	0.538	0.517
BlindPick-128 VMs	0.843	0.906	0.859	0.925	0.869	0.882	0.853
FIFO-128 VMs	−3.625	−1.333	−1.380	−1.285	−1.952	−1.000	−1.074

References

1. Schomm F, Stahl F, Vossen G (2013) Marketplaces for data: an initial survey. ACM SIGMOD Rec 42(1):15–26
2. Assunção MD et al (2015) Big data computing and clouds: trends and future directions. J Parallel Distrib Comput 79:3–15
3. Gartner I (2008) Gartner says contrasting views on cloud computing are creating confusion. http://www.gartner.com/newsroom/id/766215. Accessed on 9 July 2015
4. Kambatla K et al (2014) Trends in big data analytics. J Parallel Distrib Comput 74(7):2561–2573
5. Djebbar EI, Belalem G (2013) Optimization of tasks scheduling by an efficacy data placement and replication in cloud computing. In: Aversa R, Kolodziej J, Zhang J, Amato F, Fortino G (eds) Algorithms and Architectures for Parallel Processing. ICA3PP 2013. Lecture Notes in Computer Science, vol 8286. Springer, Cham, pp 22–29
6. Vecchiola C, Pandey S, Buyya R (2009) High-performance cloud computing: a view of scientific applications. In: 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN). IEEE
7. Ismail L, Barua R (2013) Implementation and performance evaluation of a distributed conjugate gradient method in a cloud computing environment. Softw Pract Exp 43(3):281–304
8. Piraghaj SF et al (2016) Virtual machine customization and task mapping architecture for efficient allocation of cloud data center resources. Comput J 59(2):208–224
9. Yang C et al (2017) Big data and cloud computing: innovation opportunities and challenges. Int J Dig Earth 10(1):13–53
10. Zhang F, Cao J, Tan W, Khan SU, Li K, Zomaya AY (2014) Evolutionary scheduling of dynamic multitasking workloads for big-data analytics in elastic cloud. IEEE Trans Emerg Top Comput 2(3):338–351
11. Ho Y-C, Zhao Q-C, Jia Q-S (2008) Ordinal optimization: soft optimization for hard problems. Springer Science & Business Media, Berlin
12. Hanani A, Nourossana S, Javadi H, Rahmani AM (2010) Solving the scheduling problem in multiprocessor systems with communication cost and precedence using bee colony system. In: 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), vol 5. IEEE, pp V5–V464
13. Moon S, Lee J-W (2016) Multi-residential demand response scheduling with multi-class appliances in smart grid. IEEE Trans Smart Grid. doi:10.1109/TSG.2016.2614546
14. Mansouri N, Dastghaibyfard GH, Mansouri E (2013) Combination of data replication and scheduling algorithm for improving data availability in data grids. J Netw Comput Appl 36(2):711–722
15. Rahmati B, Rahmani AM, Rezaei A (2017) Data replication-based scheduling in cloud computing environment. J Adv Comput Eng Technol
16. Wang K et al (2016) Load-balanced and locality-aware scheduling for data-intensive workloads at extreme scales. Concurr Comput Pract Exp 28(1):70–94

17. Liu C et al (2016) HKE-BC: hierarchical key exchange for secure scheduling and auditing of big data in cloud computing. Concurr Comput Pract Exp 28(1):646–660
18. Jiang C, Wang C, Liu X, Zhao Y (2007) Adaptive replication based security aware and fault tolerant job scheduling for grids. In: SNPD 2007. 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, vol 2. IEEE, pp 597–602
19. Gai K, Qiu M, Zhao H (2016) Security-aware efficient mass distributed storage approach for cloud systems in big data. In: 2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS). IEEE
20. Jiang J, Xu G, Wei X (2006) An enhanced data-aware scheduling algorithm for batch-mode data-intensive jobs on data grid. In: International Conference on Hybrid Information Technology, 2006. ICHIT'06, vol 1. IEEE
21. Mei J, Li K, Li K (2014) A resource-aware scheduling algorithm with reduced task duplication on heterogeneous computing systems. J Supercomput 68(3):1347–1377
22. Wang X, Perlman E, Burns R, Malik T, Budavári T, Meneveau C, Szalay A (2010) Jaws: job-aware workload scheduling for the exploration of turbulence simulations. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society, pp 1–11
23. Khanli LM, Far ME, Rahmani AM(2010) RFOH: a new fault tolerant job scheduler in grid computing. In: 2010 Second International Conference on Computer Engineering and Applications (ICCEA), vol 1. IEEE
24. Kazem AAP, Rahmani AM, Aghdam HH (2008) A modified simulated annealing algorithm for static task scheduling in grid computing. In: International Conference on Computer Science and Information Technology, 2008. ICCSIT'08. IEEE
25. Zhang F, Cao J, Hwang K, Li K, Khan S (2015) Adaptive workflow scheduling on cloud computing platforms with iterative ordinal optimization. IEEE Trans Cloud Comput 3(2):156–168
26. Zhang F, Cao J, Li K, Khan SU, Hwang K (2014) Multi-objective scheduling of many tasks in cloud platforms. Future Gener Comput Syst 37:309–320
27. Nanduri R, Maheshwari N, Reddyraja A, Varma V (2011) Job aware scheduling algorithm for mapreduce framework. In: 2011 IEEE 3rd International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, pp 724–729
28. Navimipour JN et al (2014) Job scheduling in the expert cloud based on genetic algorithms. Kybernetes 43(8):1262–1275
29. Li J et al (2012) Online optimization for scheduling preemptable tasks on IaaS cloud systems. J Parallel Distrib Comput 72(5):666–677
30. Mezmaz M et al (2011) A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. J Parallel Distrib Comput 71(11):1497–1508
31. Omara FA, Arafa MM (2010) Genetic algorithms for task scheduling problem. J Parallel Distrib Comput 70(1):13–22
32. Abouelela M, El-Darieby M (2016) Scheduling big data applications within advance reservation framework in optical grids. Appl Soft Comput 38:1049–1059
33. Lin B et al (2016) A pretreatment workflow scheduling approach for big data applications in multicloud environments. IEEE Trans Netw Serv Manag 13(3):581–594
34. Somasundaram TS, Govindarajan K, Kumar VS (2016) Swarm intelligence (SI) based profiling and scheduling of big data applications. In: 2016 IEEE International Conference on Big Data (Big Data). IEEE