The Journal of Supercomputing (2022) 78:4261–4289
https://doi.org/10.1007/s11227-021-04016-8

An energy-aware scheduling of dynamic workflows using big data similarity statistical analysis in cloud computing

Maziyar Grami[1]

Accepted: 5 August 2021 / Published online: 27 August 2021

Abstract
Cloud computing is a suitable platform for workflows that work with massive data and big data. Through virtualization, cloud computing converts physical infrastructures to virtual machines (VMs). Virtual machines can meet fluctuating and dynamic requests through simpler management. Workflow scheduling in cloud computing is important, concerning the fact that proper scheduling can enhance the efficiency of the cloud and good scheduling can cause energy consumption reduction. As energy efficiency is one of the most important issues in cloud computing, in this paper a new statistical analysis-based algorithm is suggested for defining similarities of input workflows. The proposed algorithm, which is called massive data similarity statistics analysis algorithm (MSSA), classifies virtual machines into virtual clusters and it executes scheduling by reforming the virtual clusters. Furthermore, MSSA investigates the similarities of message passing in two different periods; it decides for the next period, and finally, carries out the load balancing by a new method for transferring the machines in virtual clusters. The results of simulation with CloudSim show that the proposed algorithm is more energy efficient in comparison with traditional methods, like FIFO, and heuristic methods such as BlindPick, and relatively new method, named eOO as well as makespan. The main parameter for comparing is makespan and energy consumption. The results showed that the proposed method is more energy efficient compared with similar algorithms and it reduced the makespan significantly.

Keywords Big data · Cloud computing · Scheduling

✉ Maziyar Grami
maziyar.grami@yahoo.com

1 Kermanshah Branch, Islamic Azad University, Kermanshah, Iran

1 Introduction

Nowadays, organizations, companies, and social networks store and produce a great deal of data [1]. Managing these data requires powerful computational and storage systems, one of which is cloud computing. Cloud computing is a growing paradigm that provides computational and dynamic resources based on a pay-per-use model for a wide range of usage [2]. Virtualization technology is one of the main bases of cloud computing [3]. Virtualization hides the heterogeneity of computational resources, and cloud users can meet their fluctuating requests easily [4]. Cloud computing not only is popular in business but also due to cost-effectiveness, reliability, fault-tolerant, high ability, and scalability has gained a reputation in academic communities. Users in cloud computing do not need to possess new infrastructures. These requirements can be best fulfilled based on the demands of users anywhere in the world [3, 5].

Workflows are a famed programming method in distributed computational infrastructures, like the cloud. Workflows, which are in working with massive data and big data, are, like scientific workflows, highly demanded to form various infrastructures. Infrastructure provision for massive data is challenging and the existing resource management and scheduling methods may not be responsive to these requirements [6]. Workflows fall into two categories including computation intensive and data intensive. The advantages of cloud computing in executing data-intensive workflows are a virtual execution environment, on-demand resource provision, and elasticity [7]. In this paper, both of the mentioned workflows are accepted as input.

Scheduling in the cloud, especially when this platform is used for massive data analysis and dynamic less predictable workload enter the clouds, has a significant role in cloud computing efficiency [8]. Massive data workflow scheduling in the existing resources pertains to NP-Hard problems [9]. Big data dynamic task scheduling is a big challenge since it requires a great deal of repetition. Therefore, this real-time scheduling is necessary for enhancing the efficiency of task execution. Reaching optimal scheduling contains a lot of difficulties. The issue of scheduling becomes more important and more complicated in dynamic scenarios for workflows that work with big data and massive data and when jobs are implemented on an elastic cloud, like Amazon EC2 and IBM RC2 [3]. On the other, energy consumption is the other important problem in cloud computing. Using a better scheduler causes less makespan and better use of resources; as a result, it can cause better energy consumption [5]. In these problems, a large space and high overhead for the creation of optimal scheduling is regarded as a great challenge. Finding an optimal solution is most often so costly that a sub-optimal one would suffice. Instead of searching in all the possible situations, sub-optimal solutions investigate a limited number of them.

So the other part of this paper is focused on the energy consumption of the proposed method and comparing it with other methods. Scientific workflows and big data need high-performance computing infrastructures. They need large-scale data centers, consuming a large amount of energy. Because of the growing request for using these large data centers, reducing the energy consumption of the hardware was not a responsive method. Moreover, there are worse problems that arise from high power consumption; for example, using strong cooling systems can overheat other resources and it can cause a reduction in other hardware's life-time, or it can cause increasing carbon dioxide emissions and intense greenhouse effect.

Likewise, the proposed algorithm firstly offers a sub-optimal solution and during job execution, regarding the conditions, attempts to improve the throughput measure. This algorithm, which is abbreviated as MSSA, makes physical machines and servers available in the form of identical virtual machines. Virtual machines (VMs) are mount in virtual clusters (VCs). This mounting is done according to the number of input workflow types. After the random initial scheduling, the scheduler decides on the continuation of the scheduling process at end of time at the same intervals based on input workflows similarity. If there is no need to change the strategy of scheduling due to similarity, scheduling will continue in the manner of the previous period. Otherwise, the schedule will change with clusters reforming considering the load of each VC. Throughout the change in the arrangement of clusters, it endeavors that the VMs, with regard to message passing among clusters and load balancing, migrate from clusters with a low load to ones with a higher load. Considering elasticity in cloud computing, VM migration is executed quite easily, especially in VCs. As it was mentioned, the scheduler adapts itself with the conditions and if there is a sudden change in the type of input workflows, the selfsame scheduler can handle it. Scheduling and execution phases are carried out in a parallel way. In other words, during the execution of the workflows in the t_i period, necessary analyses for decision making in the t_{i+1} period can be done in a separate machine. This helps an excessive overhead not be imposed on the system. On the other hand, if there is no change in the input workflows, the scheduling will not change at all so the MSSA overhead dramatically decreases.

The rest of this paper is organized as follows: in Sect. 2, the most recent related work in the field of scheduling in cloud computing and its types are surveyed. In Sect. 3, the problem and its related parameters are further explained. In Sect. 4, dedicated to a discussion of the proposed algorithm. In Sect. 5, the results of algorithm evaluation and comparison with other algorithms are put forth. And in Sect. 6, the conclusion and future work are mentioned.

2 Related work

Workflows scheduling in cloud computing has changed into an interesting issue for researchers. This is probably due to the significance of proper scheduling in cloud efficiency and improvement in its function. Various methods with various purposes for resource scheduling in cloud computing have been proposed by researchers. Madni et al. [10] have investigated the challenges and resource scheduling opportunities for IaaS (infrastructure as a service) in cloud computing. They classified resource scheduling algorithm as follows: cost-aware, efficiency-aware, energy-aware, load balancing, QoS, and utilization-aware resource scheduling. Each of these types, in turn, has their parameters. A scheduling parameter can be mono- or multi-purpose. In other words, it selects one or more parameters for improvement. The present article is in the field of QoS and efficiency-aware including throughput and makespan parameters. Smanchat and Viriyapant classified workflow scheduling problems in the cloud. Their concentration was on the scientific workflow scheduling in cloud computing environments. They categorized the scheduling criteria in time, cost, reliability, energy, and security and stated that a workflow is comprised of several tasks with a priority in their execution which is shown by DAG [11]. Alkhanak et al. suggested another classification for workflow scheduling approaches in a cloud computing environment. This categorization was based on cost-aware challenges. They divided cost-aware workflow scheduling approaches into two general groups. The first one is related to approaches aiming at users benefits, and the second class is approached focusing on cloud providers [12]. In this regard, Mansouri et al. [13] have studied and analyzed some proposed schedulers in cloud computing. They presented algorithms in different studies based on the type of scheduling and compared their purposes, features, and limitations.

Zhang et al. proposed a method called multi-objective scheduling (MOS). Their MOS, which aimed to properly organize VCs to execute big data sets, was based on the ordinal optimization (OO) method, which was used to design the optimization of dynamic and complex systems. They developed OO in line with cloud platform needs. In OO algorithm, the problem is searched in the whole existing space to an optimal space to be found. In this algorithm, the whole space is searched just once and at the beginning. Their algorithm was such a one that would make the algorithm more flexible with respect to VCs on VMs and the creation of periods for workflows execution for the cloud environment. In order to measure the desirability of the produced scheduling at the beginning of each period, they used a two-objective evaluation function to reduce the costs and the time of workflows execution. At the end of periods, in case of lack of desirability, the algorithm searches for better VCs in a smaller space in comparison with the whole problem. They used their results for stimulation with LIGO. Due to the high overhead of their algorithm, real implementation was not possible. However, in the tests, the efficiency of their algorithm and also their selected scheduling for the previous algorithms, i.e., Monte Carlo and BlindPick, were compared and some desirable results were achieved [14].

Zhang et al. [9] presented another method for workload scheduling in a cloud computing environment. Their method, which is called iterative ordinal optimization (IOO), uses ordinal optimization in each iteration to reach sub-optimal scheduling. Their method was based on elastic cloud computing which supported clustering existing VMs in data centers virtually. They divided total time into some periods. At the end of each period, the scheduling desirability metric was evaluated thanks to a multi-objective function which included minimization of the used memory and the time of workflow execution. If workflow execution is not desirable regarding the amount of the used memory and the time of workflow execution at the end of each period, the arrangement of virtual clusters in an iteration changes, to reach a sub-optimal solution and schedules the following periods according to it. They carried out their tests on the IBM RC2 cloud using real workflow. Results showed that the proposed method in comparison with the previous methods, like Monte Carlo and BlindPick, presented a suitable efficiency in terms of memory usage.

The authors of [9] in another paper [7] presented the improved method of eOO based on IOO. This method also does the scheduling with the clustering of VMs. They offered the concept of similarity to decide on making new scheduling or not. Their tests, like the previous paper, showed an increase in the efficiency of algorithm [7].

Hanani et al. the authors did some improvement on eOO. In this paper, considering more than one parameter, a proper scheduling was created for each period. This scheduler was an organization for the number of virtual machines for each virtual cluster, but if there was a desirable similarity between workloads of two consequence periods, this procedure would was ignored. In this way, the time consumption for calculations is reduced. The results showed that a more optimized solution is obtained in comparison with the rated methods, such as BlindPick, OO, Monte Carlo, and eOO in a reasonable time. The suggested method was flexible and it was possible to change the weight ratio of the proposed criteria in different environments to be consistent with different environmental conditions. The results showed that proposed method achieved up to 28% performance improvement in comparison with eOO [15].

In [16], Jafari proposed a new method for task scheduling in a cloud environment with the help of bee colonies. In this method, the place of food shows the possible solution of scheduling, and the makespan of a solution was a measure of quality or fitness of a solution. Experiments were done by MATLAB and they showed that the makespan of their proposed method was better in comparison with the other methods.

In [17], authors presented a bandwidth-aware Hadoop scheduling. They used Hadoop to apply the programming feature of MapReduce to process data in a parallel way. Their innovation was considering bandwidth as a bottleneck for big data transferring. In this scheduling, tasks are firstly dedicated and it is guaranteed that each finish time will be optimal. Then, those tasks which have local replication will be executed. This is done to prioritize those tasks which are more executable without bandwidth usage. Their tests were carried out on Hadoop and their results showed that both of their algorithms were more efficient in comparison with the default scheduling of Hadoop. In the very same year, Mashayekhy et al. [18] presented energy-aware scheduling for MapReduce jobs. In that paper, they proposed a framework for energy consumption improvement in MapReduce. This, in turn, supported the service level agreements. They tested their scheduling algorithm on a big Hadoop cluster with 224 processors. Results showed that their proposed algorithm on average has 40% less energy consumption in comparison with ordinary scheduling. Bodik et al. [19] presented a deadline-aware algorithm for big data jobs processing. Their main purpose was maximizing the accomplished jobs, in which the value of each job depended on its finish time.

Abouelela et al. presented a scheduling algorithm for big data using a reserving framework in optical grids. In this framework, a multi-domain hierarchically scheduling is suggested; because the hierarchical method guarantees that the scheduling algorithm would only use the dedicated resources. Their iterative scheduling algorithm showed better efficiency in experiments and results proved that this algorithm will be better when more data is needed and the problem is inclined toward big data [20]. Li et al. [21] presented a job management system architecture and a scheduling algorithm for big data analysis. Their job management system was a scientific big data-oriented project. Their model had four layers including application, service, management, and infrastructure. They generated random jobs and carried out simulations using Java. Their proposed algorithm showed 10% more efficiency in comparison with other algorithms. Gautam et al. [22] worked on scheduling algorithms in the field of big data processing. In effect, their framework was studying different scheduling algorithms that were designed and implemented for Hadoop. Moreover, they classified the existing algorithms and investigated their advantages and disadvantages in this study. Wang and Raicu [23] presented scheduling for big data jobs in the cloud. Their proposed method, MATRIX, was a distributed scheduler for big data jobs in the cloud. The main purpose of this scheduling was to reach load balancing. This was content-aware scheduling containing distributed queues for waiting, ready, running, and terminated jobs. However, they offered their work without any testing and it was only as a theory. Bardhan and Menascé presented an evaluation method for scheduling big data in computer clusters [24]. They suggested a trace-driven analytic model (TDAM) for evaluating different schematic effects on the makespan. In TDAM, not only servers are implemented, but also there is an analytical queuing containing a queue for each machine. Results showed that their algorithm was better in evaluating scheduling algorithms. Zhao et al. [25] presented an SLA-based resource scheduling for big data analysis as a service in computational environments. For this purpose, they reach three aims, including maximizing the use of resources, reducing the cost of resource usage by conducting queries on VMs with lower costs, and reducing the execution time of VMs aiming cost reduction.

Dashti and Rahmani presented a method for dynamic VMs replacement with the purpose of energy efficiency. They suggested a hierarchical architecture to meet the requirements of both provider and consumer. To guarantee QoS and reduction of energy consumption, their method used PSO. Results were investigated by CloudSim [26].

We have witnessed improving dynamic voltage scaling algorithms [27]. The main focus of this research has been PACE, an approach to reducing the energy consumption of the DVFS algorithm without affecting its performance. Their results showed that PACE can substantially reduce CPU energy consumption without affecting performance. Their algorithm reduced the CPU energy consumption of previously published algorithms by 1.4–49.5% with an average of 20.6%.

Lee and Zomaya [28] focused on the energy consumption of high-density computer systems. They focused on the problem of scheduling precedence-constrained parallel applications in multiprocessor computer systems and suggested two energy-aware scheduling algorithms using dynamic voltage scaling (DVS). Their results proved that the algorithm is very compelling in terms of both application completion time and energy consumption.

Topcuouglu et al. [29] studied various algorithms for heterogeneous processors. They focused on good quality schedules and they present two novel scheduling algorithms for a bounded number of heterogeneous processors to achieve high performance and fast scheduling time simultaneously, which is called the heterogeneous earliest-finish-time (HEFT) algorithm and the critical-path-on-a-processor (CPOP) algorithm. Wang et al. [30] also researched energy consumption and used the DVFS technique (dynamic voltage frequency scaling) to present application experience for reducing power consumption of parallel tasks in a cluster.

Kimura et al. [31] researched on DVFS mechanism and proposed a new algorithm that reduced the energy consumption by reclaiming slack time in a parallel program executed on a power-scalable cluster using DVFS that can represent DAG (directed acyclic task graph). They designed PowerWatch, a toolkit to measure the power consumption of the entire cluster in real-time. The result showed that the algorithm reduces energy consumption by 25% with only 1% loss performance.

Zhuo Tang et al. [32] focused on reducing energy dissipation, to achieve a DVFS-enabled energy-efficient workflow task scheduling algorithm, DEWTS. It was an algorithm that at first calculated initial scheduling for the entire tasks and obtained the makespan and deadline based on HEFT [29]. The experimental results showed that DEWTS can reduce the total power consumption up to 46.5% for various parallel applications as well as balance the scheduling performance.

Zhong and Xu [33] studied on conserve energy consumption for battery-powered systems. They present an analytical model of general tasks for DVS. It models the voltage scaling process as a transfer function-based filtering system, which facilitates the design of two efficient scaling algorithms. The model facilitates the derivation of two DVS algorithms, time-invariant and time-variant. The time-invariant algorithm is proved to be a generalization of several existing approaches for different task models respecting energy savings. The time-variant algorithm is essentially a water-filling process. It is more efficient and easily integrated into existing schedulers.

Bini et al. [34] researched about minimizing energy consumption and they present a method for that in periodic/sporadic task system executing in processors with a discrete number of operating modes, each characterized by speed, power consumption, and transition delay. They proposed a general framework for analyzing and designing embedded systems with energy and timing requirements.

Quan and Hu [35] studied dynamic voltage scaling (DVS), an efficient technique in reducing dynamic energy consumption. They present a scheduling technique that can effectively reduce the overall energy consumption for hard real-time systems, scheduled according to a fixed priority (FP) scheme. Experimental results demonstrated that a processor using this strategy consumes less than 15% of the idle energy of a processor employing the conventional strategy.

Zhuo and Chakrabarti [36] considered the problem of developing dynamic task scheduling algorithms that minimize the system-level energy consumption (sum of CPU and device energy). They used system-level energy consideration to derive the “optimal” scaling factor by which a task should be scaled if there are no deadline constraints. Next, they developed dynamic task scheduling algorithms that make use of dynamic processor utilization and optimal scaling factor to determine the speed setting of a task. Then, they presented the *duEDF* algorithm which reduced the CPU energy consumption, and the *duSYS* algorithm and its reduced preemption version, *duSYS PC*, which reduced the system-level energy. *duSYS* and *duSYS PC* achieved large energy savings (up to 25%) compared to the CPU energy-efficient *duEDF* algorithm and up to 15% energy saving over the non-DVS scheduling algorithm.

Juarez et al. [37] proposed a real-time dynamic scheduling system to execute efficiently task-based applications on distributed computing platforms to minimize energy consumption. Scheduling tasks on multiprocessors is a well-known NP-hard problem and the optimal solution of these problems is not feasible; they present a polynomial-time algorithm that combines a set of heuristic rules and a resource allocation technique to get good solutions on an affordable time scale. The proposed algorithm minimizes a multi-objective function that combines the energy consumption and execution time according to the energy-performance importance factor provided by the resource provider or user, also taking into account sequence-dependent setup times between tasks, setup times and downtimes for virtual machines (VM), and energy profiles for different architectures. They have also evaluated the introduced overhead by measuring the time for getting the scheduling solutions for a different number of tasks, kinds of DAG, and resources, concluding that this method is suitable for run-time scheduling.

Duan et al. [38] reviewed reducing energy consumption as well as maintained high computation capacity challenge, and proposed a new scheduling approach named *PreAntPolicy* that consists of a prediction model based on fractal mathematics and a scheduler based on an improved ant colony algorithm.

Wen et al. [39] proposed a multi-objective privacy-aware workflow scheduling algorithm, named MOPA. Their model can provide cloud customers with a set of Pareto trade-off solutions. The problem-specific encoding and population initialization are proposed in this algorithm. Their work is as follows: Firstly, they demonstrate the importance of ensuring privacy protection constraints when scheduling workflows in the context of cloud computing environments. Second, they formalize the problem of scheduling workflows with privacy protection constraints and propose a corresponding algorithm to minimize both monetary cost and execution makespan.

Elhoseny et al. [40] proposed a new model to optimize the performance of the healthcare systems by reducing the stakeholders' request execution time. The proposed model consists of four main components: stakeholders' devices, stakeholders' requests, cloud broker, and network administrator. To optimize virtual machines selection (VMs) in cloud-IoT health services applications to efficiently manage a big amount of data which come from different sources such as sensor data, without human intervention, they had used three different well-known optimizers: genetic algorithm (GA), particle swarm optimizer (PSO), and parallel particle swarm optimization (PPSO). To calculate the execution time of stakeholders' requests, the proposed function is a composition of three important criteria which are CPU utilization, turn-around time, and waiting time.

Alboaneen et al. [41] have proposed a new metaheuristic method to optimize joint task scheduling and VM placement in the cloud data center. Their method has two parts, task scheduling and VM placement. Their method aims to schedule task into the VM which has the least execution cost within deadline constraint and then to place the selected VM on the most utilized physical host. They want to increase the performance in terms of execution cost, makespan, degree of imbalance, and resource utilization.

3 Problem definition

To reach a higher throughput in the cloud system scheduling which works on virtualization, the proposed algorithm reforms the cluster in the periods. In Fig. 1, the overall schema of the cloud system for solving this problem is shown.

This system is adapted from the proposal of Zhang et al. [7]. In problem definition, the elasticity of the cloud has been considered and it is assumed that VCs can transfer VMs among themselves. VMs are computing units that are located above the physical layer. It is supposed that the VMs are equally defined. VMs are mounted in C groups. Each group is called a VC. If there are C types of jobs, therefore there will be C virtual clusters. If recognizing job classes is impossible, they are classified based on the volume of the file on which workflow processes. A VM with a suitable processing capacity, which is not mounted into any cluster, is responsible for executing the scheduler. Workflows enter the cloud system and it is assumed that each workflow is comprised of several interconnected tasks that have the capability of parallelism. The number of VCs is defined according to the workflow types. In Fig. 1, it is assumed that there are four workflow types and, therefore, four VCs. The $\theta(t_i)$ vector shows the assigning of VMs to each VC. Each element of this vector shows the number of VMs assigned to each cluster t_i.

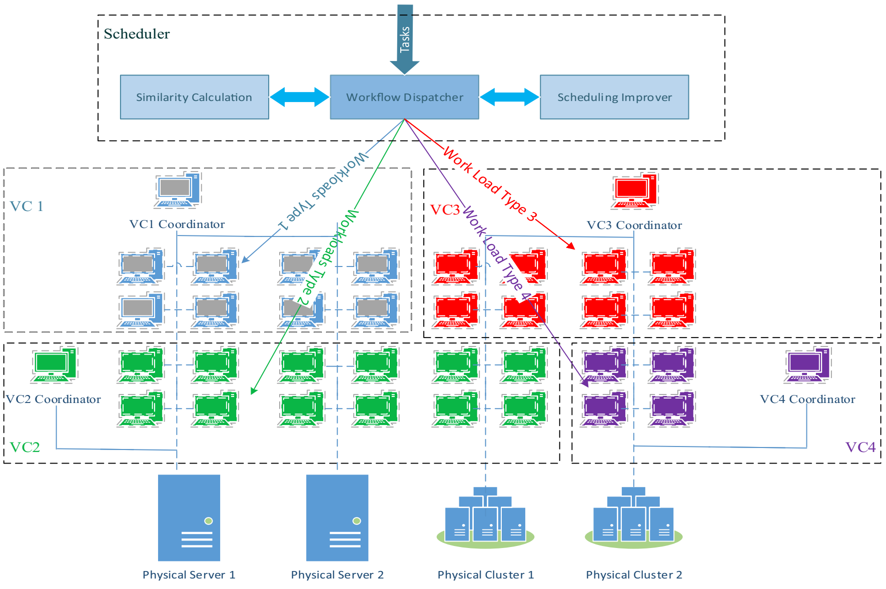


Fig. 1 The proposed cloud model based on [7], in which virtual cluster allocation is customized for the proposed algorithm

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

In Eq. (1), $\theta_c(t_i)$ means the number of VMs in the cth cluster in t_i and i is the period number. The ID of each cluster is presented as c which $c \in [1, C]$. The aim is fining $\theta(t_i)$ which has the highest throughput in each period. Let θ be the number of all usable VMs, then:

$$\sum_{c=1}^{C} \theta_c(t_i) = \theta \tag{2}$$

At the beginning of scheduling $\left(t_0\right)$, the number of VMs in each cluster is randomly defined. This procedure is used as the initial scheduling. Jobs will be randomly executed for n times to find the best scheduling in terms of throughput at the beginning of the intervals, where n is selected as lesser than all possible cases. Then, the proposed algorithm decides to change the organization of the clusters based on the existing workflows or to continue execution without reforming the clusters. In Fig. 2, the arrangement of workflows in the queue and they are entering the system can be seen.

In the input queue, in each period, existing jobs in that period are analyzed. At the end of the period, the workflow dispatcher decides for t_{i+1}th period. If it is supposed to be no changes, there will be no change in the organization of the clusters and the schedule will continue as the previous period. Otherwise, the organization of clusters will change as is explained in the following.

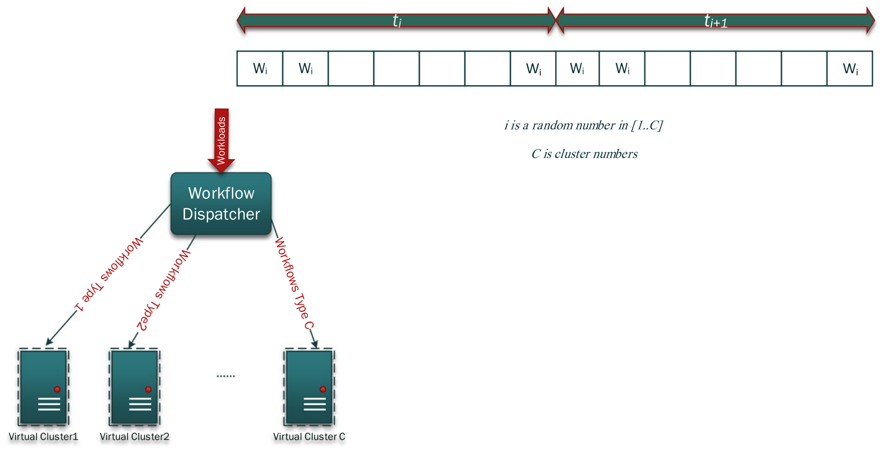


Fig. 2 Arrangement of workflows in the queue and they are entering the system

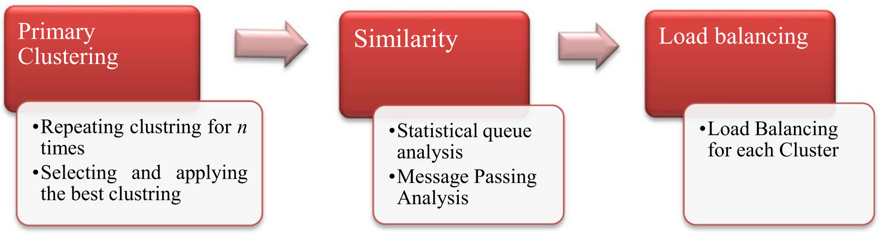

Primary Clustering
•Repeating clustring for *n* times
•Selecting and applying the best clustring
Similarity
•Statistical queue analysis
•Message Passing Analysis
Load balancing
•Load Balancing for each Cluster

Fig. 3 Phases of the proposed algorithm

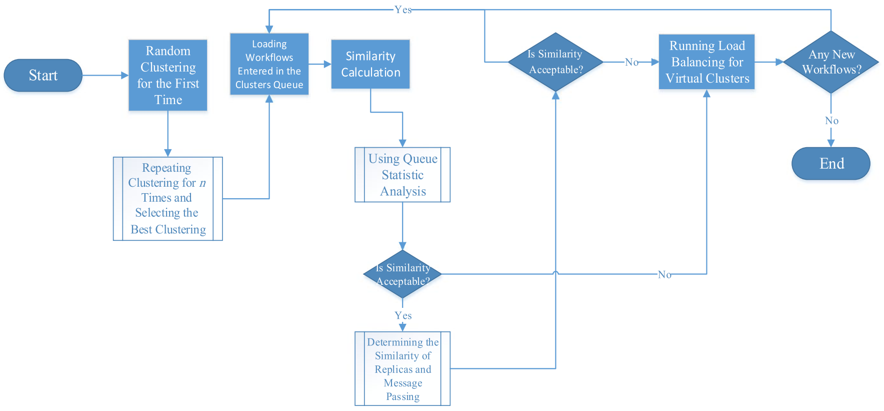

Start
Random Clustering for the First Time
Repeating Clustering for *n* Times and Selecting the Best Clustering
Loading Workflows Entered in the Clusters Queue
Similarity Calculation
Using Queue Statistic Analysis
Is Similarity Acceptable?
Yes
Determining the Similarity of Replicas and Message Passing
No
Is Similarity Acceptable?
No
Yes
Running Load Balancing for Virtual Clusters
Any New Workflows?
No
End

Fig. 4 MSSA in the form of a flowchart

4 Proposed algorithm

The proposed algorithm (MSSA) is more useful in cloud systems that work to analyze big data. As a result, a better decision regarding workflows has a considerable effect on throughput and performance. Workflow analysis in the proposed algorithm is done in a parallel way with job execution and does not interfere with the makespan. It means there is a VM that is responsible for cluster management and task transfer to the corresponding clusters. The proposed algorithm has generally three phases, which are shown in Fig. 3. These phases include:

– Primary clustering.
– Similarity calculations.
– Load balancing.

4.1 Algorithm phases discussion

The algorithm phases are shown in detail in a flowchart in Fig. 4. In the following, each section is totally explained.

4.1.1 Primary clustering phase

At the beginning of scheduling, existing VMs are placed in VCs based on the number of workflow types present in the cloud system queue. If there are C types of workflows, there will be C clusters. The VMs are randomly placed in the clusters; then, the throughput is calculated based on this random scheduling. After *n* times calculation, the best distribution, regarding throughput, will be chosen and the workflows will be sent to clusters (which *n* is much smaller than all possible conditions).

Each workflow contains several tasks that can be executed in a parallel way on VMs. In Fig. 5, it is assumed that there are 20 VMs. If there are 4 clusters, there will be two hypothetical conditions for distributing VMs in virtual clusters.

4.1.2 Similarity calculation phase

This section is related to similarity calculation. Similarity calculation, which was first introduced by Zhang et al. [7], is applied more completely. In this section, the similarity of workflows that are supposed to enter the system in the next period, with the workflows that have already been executed in the previous period is calculated; besides, the similarity of message passing of VCs between current and previous period is calculated. If there is an acceptable similarity level, the organization of the VMs among the VCs will not change. However, if the similarity is lower than a certain level, clustering should be performed. This change in the VMs distribution is optimally done to adapt the scheduler with conditions and to have better efficiency.

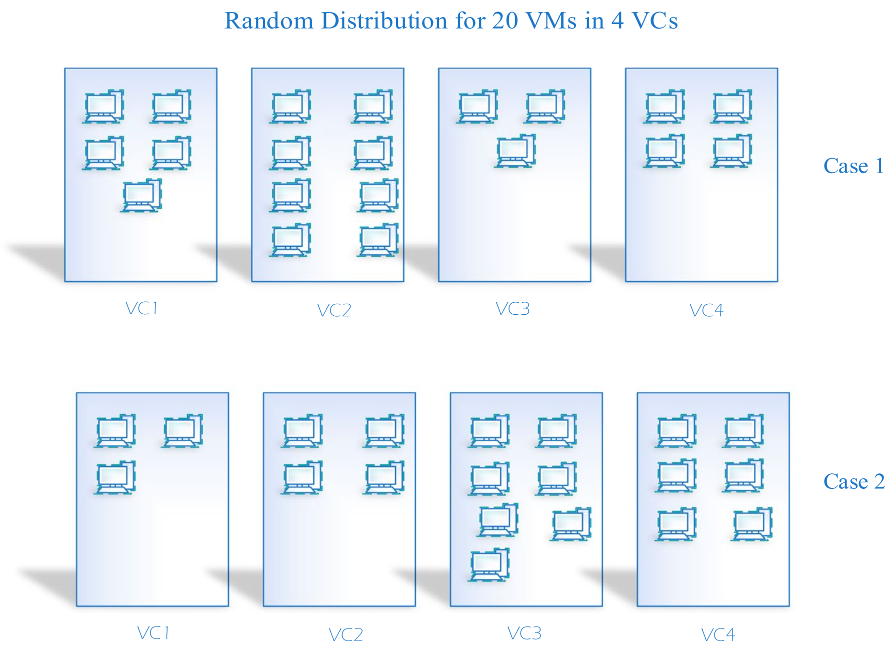


Fig. 5 Distributing VMs in VCs

4.1.2.1 Similarity using statistic analysis In this phase, the similarity of the existing workloads in the queue in the current period is compared with workflows in the previous periods. This is done by a Chi-squared statistical test. If the scheduler is at the beginning of t_{i+1}th period, the existing workflow in the queue in the current period should be compared with the workflow in t_ith period. In Fig. 6, the hypothetical form of the queue in t_ith period is shown.

According to the number of repetitions of jobs in each period, the possibilities are calculated in Table 1. To calculate the possibilities of each workflow, the frequency is divided by the number of all workflows [Eq. (3)].

$$p_i = \frac{\text{Number of Workflows Type } i}{\text{Number of All Workflows}} \tag{3}$$

In Table 1, the calculated possibilities for each cluster are shown.

In the t_{i+1}th period, m jobs entered Fig. 7. In this phase, it is tested whether m new workflows are similar to previous workflows. First, the expected frequency and observed frequency are calculated according to Table 2 and are placed in the Chi-square equation.

The Chi-square equation, which is used for testing the data similarity, is calculated using Eq. (4).

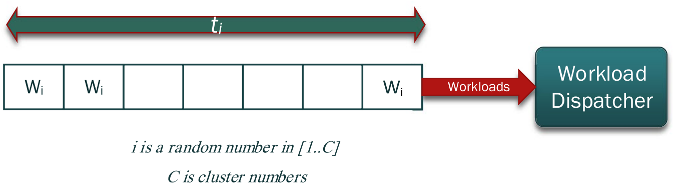


Fig. 6 Hypothetical schema of the queue

Table 1 possibilities of each cluster in t_{ith} period

Cluster numbers [1,…,C]	1	2	…	c	…	C
Possibility	p_1	p_2	…	p_c	…	p_C

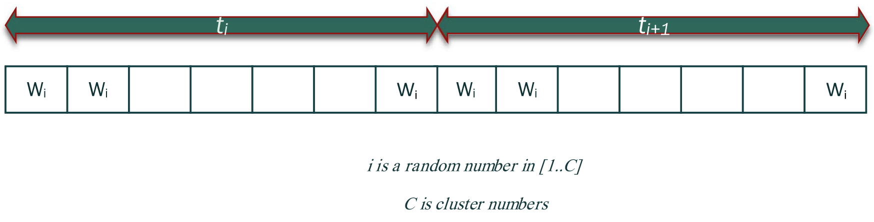


Fig. 7 Distribution of new workflows in the queue

Table 2 Expected frequency calculation for each cluster

Expected frequency	Observed frequency	Type
$F_{o1} = \mathrm{m}\cdot p_1$	f_1	For the first type
$F_{o2} = m\cdot p_2$	f_2	For the second type
…	…	…
…	…	…
$F_{oC} = m\cdot p_C$	f_C	For cth type

$$\chi^2 = \sum_{c=1}^{C} \frac{(f_c - F_{oc})^2}{F_{oc}} \tag{4}$$

where f_c is the observed frequency for cth cluster in t_ith period, and f_{oc} is the expected frequency for the cth cluster in t_{i+1}th period. Then, the χ^2 calculated is compared with $\chi^2_{\alpha,c-1}$ from Chi-square probabilities table. α is the error rate that is assumed 0.05 in this research. If $\chi^2 > {\chi^2}_{\alpha,c-1}$, there is a significant difference between the types of workflows which is entered in these two consequence times; so, H_0 and H_1 hypotheses are defined as follows:

H_0: entering workflows in the queue is similar to the previous period.
H_1: new input is not similar to the previous period.

If $\chi^2 > {\chi^2}_{\alpha,c-1}$, H_0 passes. Thus:

If H_0 is confirmed, the similarity is acceptable in this phase and the similarity of message passing should also be investigated for the final decision. Otherwise, the similarity is not acceptable and load balancing should be executed.

4.1.2.2 Similarity in message passing using Markov chain If queue analysis similarity is at an acceptable level, the similarity in message passing among clusters will calculate and the final decision about the similarity in the two following periods will be made. Figure 8a shows, in a hypothetical condition, the existence of 4 clusters for message passing among VCs in two periods. It is assumed that there is a definite message passing between every two VCs. This assumption is based on reality; because it is probable that clusters have message passing between each other due to request of replicas which are placed in the VMs of other clusters. This request for replica may be because of reforming the clusters in the previous periods. Even if there may be no replica requests, cluster heads should have messages passing among themselves for knowing the condition of the clusters. Each cluster head has the existing information of VMs in its VC. The similarity of message passing for t_ith period is calculated through a comparison of the message passing graph in the current period (t_i) and the previous period (t_{i-1}).

For this purpose, firstly vector $p\left(t_{i-1}\right)$ is formed based on Eq. (5). M_{ij} is the volume of sent messages from VC_i to VC_j in megabytes.

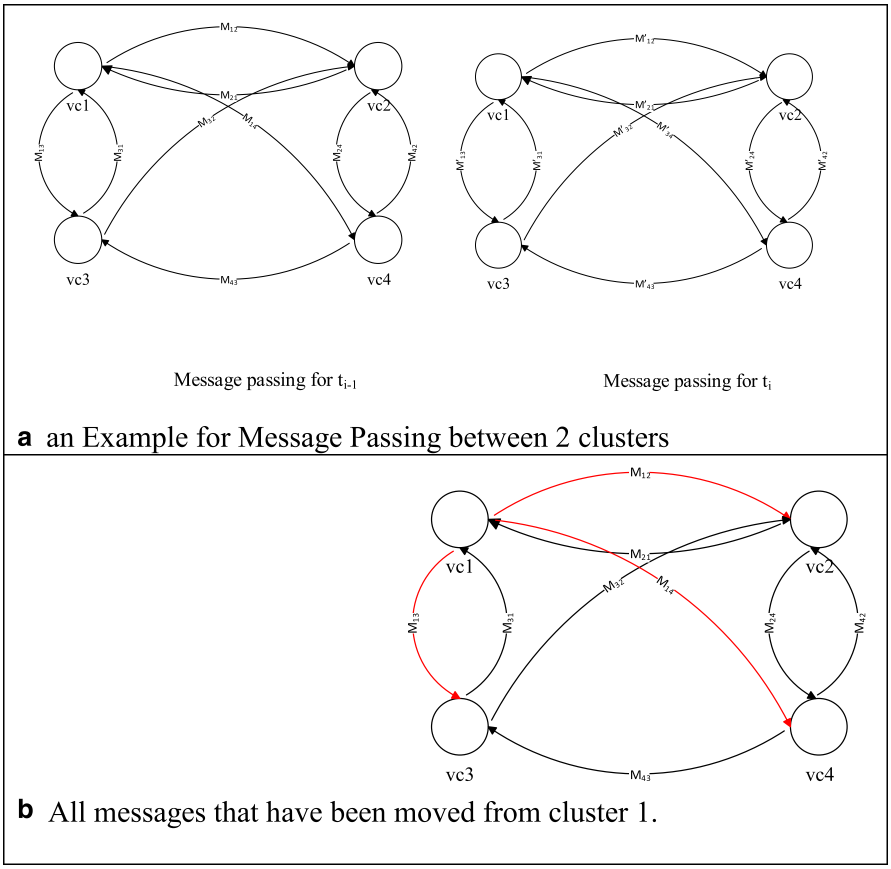


Fig. 8 Message passing procedure

$$p(t_{i-1}) = \left[\frac{\sum M_{1j}}{\sum Mij} \ \frac{\sum M_{2j}}{\sum M_{ij}} \ \cdots \ \frac{\sum M_{Cj}}{\sum M_{ij}} \right] \quad i,j = 1,\ldots,C \tag{5}$$

$\sum M_{1j}$ illustrates the total of messages for VC_1 in t_{i-1}th period. In Fig. 8b, messages for VC_1 are shown. In the next step, the transfer matrix is obtained for the message passing graph related to t_ith period using Eq. (6).

$$p_{\text{Tran}}(t_{i-1}) = \begin{bmatrix} 0 & \frac{M_{12}}{\sum M_{1j}} & \cdots & \frac{M_{1c}}{\sum M_{1j}} \\ \frac{M_{21}}{\sum M_{2j}} & 0 & \cdots & \frac{M_{2c}}{\sum M_{2j}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{M_{c1}}{\sum M_{cj}} & \frac{M_{c2}}{\sum M_{cj}} & \cdots & 0 \end{bmatrix} \quad j = 1,\ldots,c \tag{6}$$

In the transfer matrix $p_{\mathrm{Tran}}(t_{i-1})$, the summation of each row equals 1. Through multiplying the possible matrix of t_{i-1} period, which is shown by $p(t_{i-1})$, by the transfer matrix $p_{\mathrm{Tran}}(t_{i-1})$, the expected matrix of probability in t_ith period is obtained. The expected probability matrix is shown as $p_{\exp}(t_i)$. The calculation method of $p_{\exp}(t_i)$ is shown in Eq. (7):

$$p(t_{i-1}) \times p_{\mathrm{Tran}}(t_{i-1}) = p_{\exp}(t_i) \tag{7}$$

If the expected probable vector or $p_{\exp}\left(t_i\right)$ compares with $p\left(t_i\right)$ probable vector, which can be calculated through the message passing graph in t_{i-1}th period and using Eq. (5), it can be seen that whether the message passing in the last two periods is similar or not. If similarity reaches a certain level, the organization of clusters is allowed not to change in t_{i+1}th period, considering similar message passing in the previous periods, and if there not any noticeable change in similarity, the arrangement of VMs in the clusters can be changed in the following periods. Equation (8) shows how to compare $p_{\exp}\left(t_i\right)$ with $p\left(t_i\right)$. If it is smaller than a threshold, the similarity is acceptable and there is no need to execute the load-balancing phase and new workflows can be assigned to VCs. In other terms, if so, the current scheduling is good and the scheduler continues with the current scheduling. Otherwise, the similarity level is not acceptable and the load-balancing phase should be executed.

$$p_{\exp}(t_i) - p(t_i) = \alpha \tag{8}$$

The threshold is obtained through trial and error and by repeating the test several times for different measures, which is investigated in the further results section.

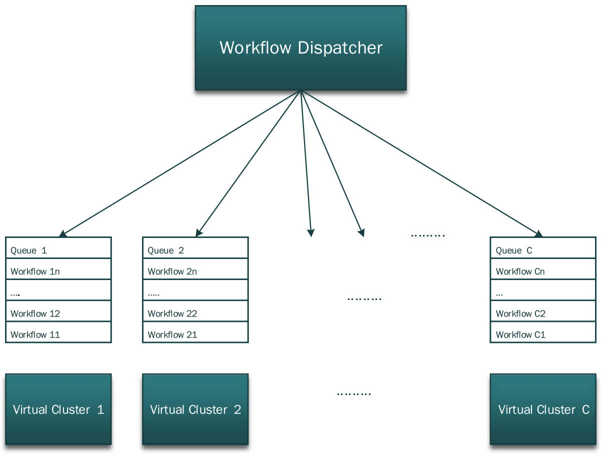


Fig. 9 Workflows arrangement in the VC queue

Table 3 Signs and equations used in the load-balancing phase

Symbol	Explanation
$q_c(t)$	Total volume of workflows in *c*th queue cluster in *t*th period in MB
$\theta_c(t_i)$	The number of VMs in $t_{i\text{th}}$ period for *c*th cluster
$\theta(t_i) = \sum_{c=1}^{C} \theta_c(t_i)$	$\theta(t_i)$ is the total active VMs in $t_{i\text{th}}$ period
$\widehat{t}_i$	The final moment of $t_{i\text{th}}$ period
$f_c(t_i)$	The volume of new *c*-type workflows in $t_{i\text{th}}$ period calculated in the similarity phase
$q_c(\widehat{t}_i) + f_c(t_{i+1})$	The load volume of cluster *c* in $t_{i+1\text{th}}$ period The above amount defines the workload for each cluster in the next period
$\varepsilon = \frac{\sum_{c=1}^{C}(q_c(\widehat{t}_i)+f_c(t_{i+1}))}{\theta(t_i)}$	The number of workflows for each machine (The existing workflow and that one which is supposed to enter the clusters queue in the next period is divided by the total number of machines. The workload which is supposed to enter is calculated in the similarity determination phase)
$\theta_c^{\exp}(t_{i+1}) = \left\lceil \frac{(q_c(\widehat{t}_i)+f_c(t_{i+1}))}{\varepsilon} \right\rceil$	The number of expected machines for cluster *c* in t_{i+1} period. This amount shows the number of machines, which is assigned to each cluster, expected for load balancing considering the present load and that one that is supposed to be added to the cluster

4.1.3 Load-balancing phase

This phase will have done when the similarity calculation phase proves that the similarity level is not acceptable. Initially, it should be noted that each VC has one queue, and the workflows assigned to that cluster are placed in the selfsame queue (Fig. 9).

In Table 3, the used signs are explained.

Now, based on $\theta_c^{\exp}(t_{i+1})$ and number of VMs in the *c*th cluster in $t_{i\text{th}}$ period ($\theta_c(t_i)$), shortage or excess of VMs can be calculated for each cluster [Eq. (7)].

$$\forall c \in C \Rightarrow \theta_c^{\exp}(t_{i+1}) - \theta_c(t_i) = \theta_c^{extra}(t_{i+1}) \tag{9}$$

$\theta_c^{extra}(t_{i+1})$ shows the number of VMs which are expected to be assigned to the *c*th virtual cluster in t_{i+1}th period. This parameter can be positive, negative, or even zero. If this is positive, it means that the *c*th cluster has received more VMs than expected. Otherwise, it has received less, and if zero, the cluster has received as many VMs as it was expected. The proposed algorithm transfers the VMs from VCs which have received more machines than expected to those clusters that received fewer VMs. This is done according to Algorithm 1. In this algorithm, Q^+ is the queue in which VCs with a positive amount $\theta_c^{extra}(t_{i+1})$ is placed and Q^- is the queue in which clusters with a positive $\theta_c^{extra}(t_{i+1})$ are placed.

```
for   $c = 1$   to   $C$   {
    $\theta_c^{extra}(t_{i+1}) = \theta_c^{\exp}(t_{i+1}) - \theta_c(t_i)$
    if   $\theta_c^{extra}(t_{i+1}) > 0 \Rightarrow$ insert   $<\theta_c^{extra}(t_{i+1}), c>$   in   $Q^+$
    if   $\theta_c^{extra}(t_{i+1}) < 0 \Rightarrow$ insert   $<\theta_c^{extra}(t_{i+1}), c>$   in   $Q^-$
}
Order   $Q^+$   by   $\theta_c^{extra}(t_{i+1})$   descendly
Order   $Q^-$   by   $\theta_c^{extra}(t_{i+1})$   ascendly
Calculate   $\Delta p$
for   $i = 1$   to   $Q^+$.lenght
for   $j = 1$   to   $Q^-$.lenght
   if   $(\Delta p[i][j] > 0$   and   $Q^+[i].\theta_c^{extra}(t_{i+1}) > 0)$
      while   $Q^+[i].\theta_c^{extra}(t_{i+1}) > 0$
         move   VMs   from   VC   $Q^+[i].c$   to   $Q^-[j].c$
```

In the above algorithm, each object in the queues is an object in which $\theta_c^{extra}(t_{i+1})$ and c are placed. Then, Q^+ and Q^- are arranged in descending and ascending orders, respectively. According to the Δ_p matrix, the VMs in virtual clusters, which are placed in Q^+, are transferred to those which are placed in Q^-. Δ_p matrix is calculated based on the message passing graph and using Eq. (10). The message passing graph is shown in Fig. 8a.

$$p_{i-1} = \begin{bmatrix} 0 & \frac{M_{12}}{\sum M_{ij}} & \cdots & \frac{M_{1c}}{\sum M_{ij}} \\ \frac{M_{21}}{\sum M_{ij}} & 0 & \cdots & \frac{M_{2c}}{\sum M_{ij}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{M_{c1}}{\sum M_{ij}} & \cdots & \cdots & 0 \end{bmatrix} p_i = \begin{bmatrix} 0 & \frac{M'_{12}}{\sum M'_{ij}} & \cdots & \frac{M'_{1c}}{\sum M'_{ij}} \\ \frac{M'_{21}}{\sum M'_{ij}} & 0 & \cdots & \frac{M'_{2c}}{\sum M'_{ij}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{M'_{c1}}{\sum M'_{ij}} & \cdots & \cdots & 0 \end{bmatrix}$$

$$\Delta p = p_i - p_{i-1}$$

M_{ij} is the number of sent messages from VC_i to VC_j in MB. The difference between matrixes is that each element M_{ij} is divided by all the sent and received messages in the system. In effect, the proportion of the volume of messages in each cluster is obtained in relation to the total messages. Δ_p shows the number of changes in the proportion of message passing in the present period in relation to the previous one. Based on these changes, a decision will be made for the next period.

4.1.4 Energy consumption model

Today's servers consume up to half of their peak power in idle time. The power consumption of each server includes static power consumption and dynamic power consumption which are called p_{static} and $p_{dynamic}$, respectively. $p_{dynamic}$ can be formulated as Eq. (10) [38].

$$p_{dynamic}(t) = \begin{cases} U_t^j \cdot \alpha_j \\ O_j \cdot \alpha_j + \left(U_t^j - O_j\right)^2 \cdot \beta_j \end{cases} \tag{10}$$

where O_j is the optimal usage level in terms of performance per watt which is about 70% server utilization below the optimal usage level, there would be a linear relationship between utilization ratio and dynamic power consumption; otherwise, the extra utilization will cause the exponential rise of dynamic energy consumption [42]. α_j and β_j are the coefficient of the linear part and the exponential part for the jth server. U_t^j is calculated using Eq. (11).

$$U_t^j = \frac{\sum_{x=1}^{m} n_{VM_x}^j \cdot VC_{cpu}^x}{SC_{cpu}^j} \times 100 \tag{11}$$

where $n_{VM_x}^j$ shows the number of VMs allocated on the jth server and VC_{cpu}^x shows the CPU capacity of the jth server.

5 Results

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

Simulations are carried out on a PC with Intel Dual Core CPU with 2 GHz frequency, 3 Gigabyte Ram, and 32-bit Windows 7 operating system. This simulation was performed by CloudSim version 3.0 and Java Development Kit 1.8.0 in Netbeans IDE 8.1.

We utilized seven clusters and seven jobs type. We tested our method in three scenarios and with 32, 64, and 128 virtual machines. Also, a data center was utilized, which comprised of 4 servers that each of which had 8 CPUs (2.0 GHz), 8 GB Ram, and 32-bit Linux system. Each virtual machine had one CPU (1.0 GHz), 512 MB Ram, and 32-bit Linux system. These VMs were interconnected by a connection with 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 workflows that were used in [14]. 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 the random generator was identical to others in order to have the same order of jobs to provide a fair simulation. Details of jobs are showed in Table 4.

According to Table 4, there are 7 types of jobs (workloads), each of which has a special number of tasks (cloudlets), length, and file size. For example, job 2 has 6 tasks with a total length of 82,000 million instructions and if it needs a file for replication, the file size will be 300 MB. It must be noted that since CloudSim is not a tool for big data, we mime the behavior of the big data by extending the total length of jobs. We created jobs with more instructions and, as a result, jobs take more time for executing.

We evaluated our method and similar methods under different scenarios; scenario 1: we used the numbers that are presented in Table 3 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 "Very Big Workflow Generation" or "VERY BIG." 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 simulations were run for the existence possibility of different replicas. In effect, the proposed algorithm was tested in situations where possibilities of needing replications were 0.2, 0.4, and 0.8; but in order to abbreviate, we kept the message passing ratio fixed. In all scenarios, the message passing ratio is 0.7. We have compared our method with eOO, BlindPick, FIFO, and MEOO. BlindPick and eOO are introduced in [9]. MEOO method is introduced in our previous paper that is submitted before publishing this paper [15]. MEOO is multi-parameter scheduling for scheduling big data workflows.

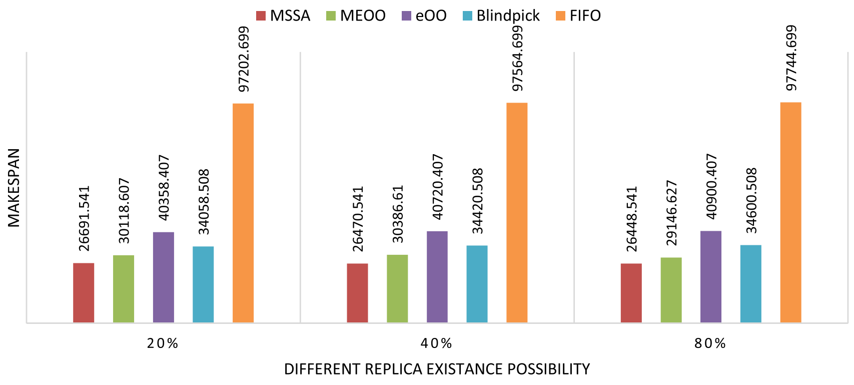
Scenario 1 (BIG):

Figure 10 indicates the comparison of scenario 1 for 32, 64, and 128 virtual machines.

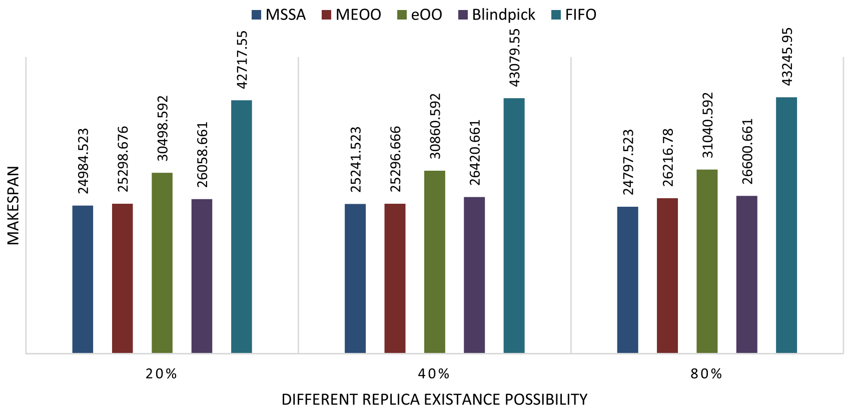
It was observed that the different replication ratio causes a small impact on all of the tested methods. However, the time of processing replications is very smaller processing time than these big workloads. As the replication rate raises, we can see better improvement in the proposed method. MSSA is not very good in scenario 1, especially when the replication ratio is low, but when the replica ratio raises, our method conquers other methods, but it is not significant. In Fig. 10, we can compare the improvement of our method in terms of makespan.

Table 4 Details of workloads

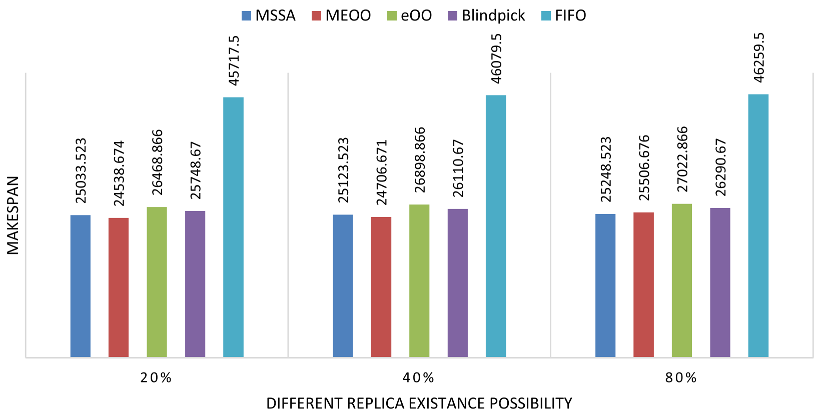
Jobs	Number of tasks	Total length (no. of instructions)	Size of needed file (if exist) (MB)
Job 1	6	24,000	300
Job 2	6	82,000	300
Job 3	3	360,000	300
Job 4	4	80,000	300
Job 5	2	70,000	300
Job 6	10	38,000	300
Job 7	8	400,000	300



(a) 32 Virtual Machines



(b) 64 Virtual Machines



(c) 128 Virtual Machines

Fig. 10 the makespan in the 1st scenario

When the number of VMs is 32, we can see a significant improvement, but when the number of VMs is 64 and 128, our method is not worthy. FIFO always had the worst and the proposed algorithm (MSSA) always had the best makespan among the other algorithms. By increasing the number of VMs, the performance was improved in all of them. In Fig. 10, we can see that in terms of makespan, our method 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.

For the sake of brevity, energy consumption for the 1st scenario only for 32 virtual machines is shown in Fig. 11. As we can see, the proposed method uses about 3.45 kWh energy for this scenario. There is a significant improvement in comparison with other methods. This energy consumption is about 4.2, 6.5, 7.7, and 15.6 in comparison with MEOO, BlindPick, eOO, and FIFO.

Scenario 2 (VBWG):

Figure 12 shows the comparison of scenario 2 for 32, 64, and 128 virtual machines.

The difference and efficiency will increase as the data grows. In the second scenario, there is a significant difference between the proposed method and the other methods.

Like scenario 1, the FIFO algorithm always had the worst and the proposed algorithm (MSSA) always had the best makespan among the tested ones. By increasing the number of VMs, all algorithms worked better. As we can see, our method has a very good improvement in this scenario. We can conclude that increasing the length of workloads causes our method to show its power better.

Like scenario 2, for the sake of brevity, energy consumption for 32 virtual machines is shown in Fig. 13. As we can see, the proposed method uses about 26.71 kWh energy for this scenario. There is a significant improvement in

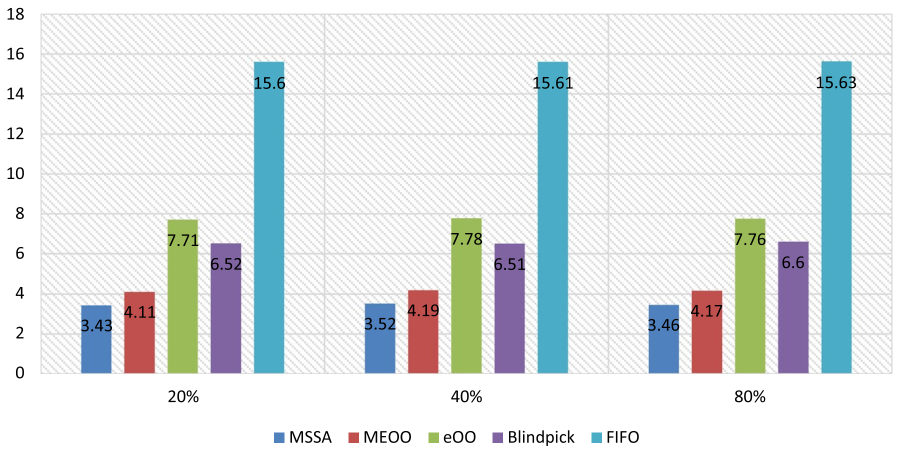
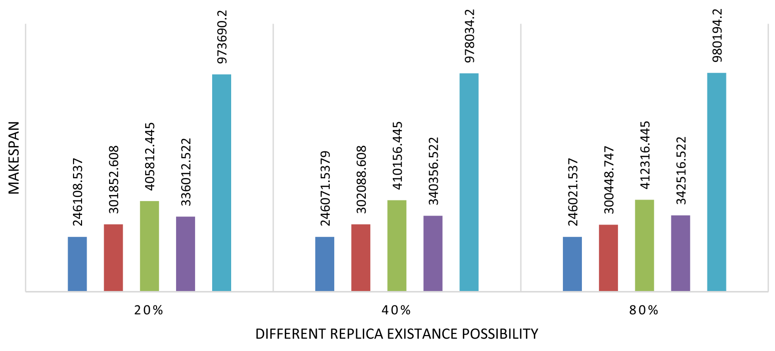
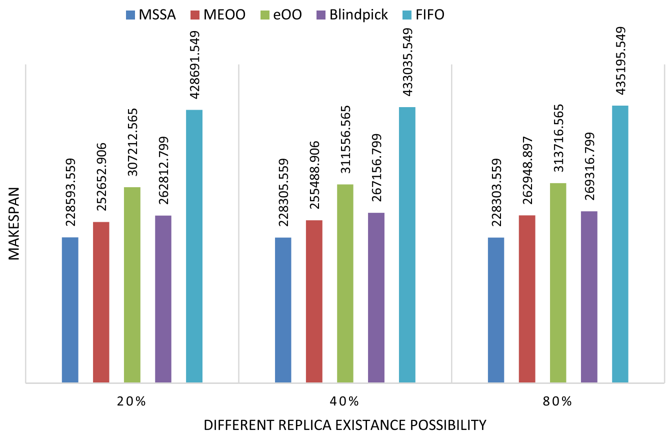


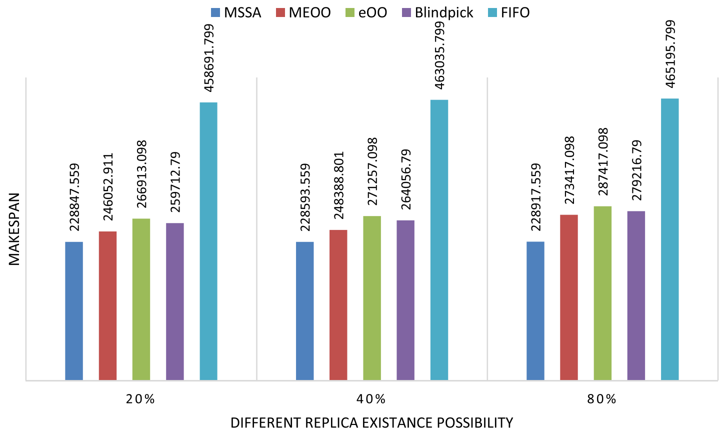
Fig. 11 Energy consumption for the 1st scenario and with 32 virtual machines (kWh)



(a) 32 Virtual Machines

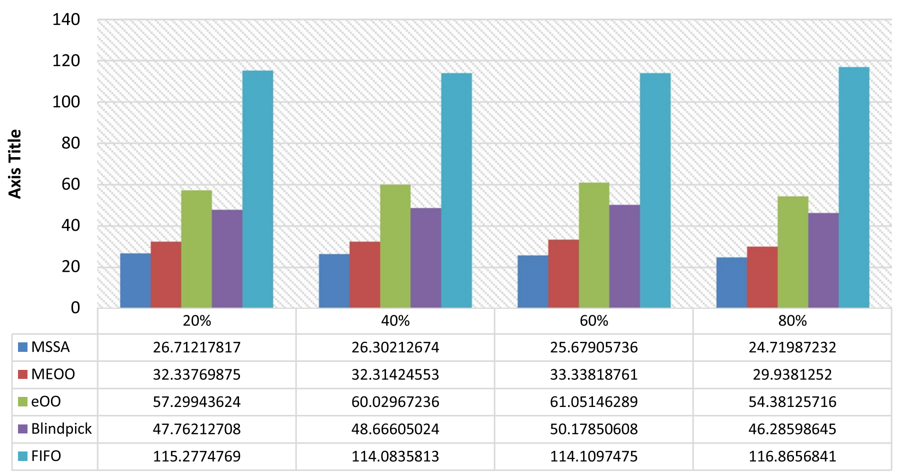


(b) 64 Virtual Machines



(c) 128 Virtual Machines

Fig. 12 The makespan in the 2nd scenario



	20%	40%	60%	80%
🟦 MSSA	26.71217817	26.30212674	25.67905736	24.71987232
🟥 MEOO	32.33769875	32.31424553	33.33818761	29.9381252
🟩 eOO	57.29943624	60.02967236	61.05146289	54.38125716
🟪 Blindpick	47.76212708	48.66605024	50.17850608	46.28598645
🟦 FIFO	115.2774769	114.0835813	114.1097475	116.8656841

Fig. 13 Energy consumption for the 2nd scenario and with 32 virtual machines (kWh)

comparison with other methods. This energy consumption is about 32.33, 47.76, 57.29, and 115.2 in comparison with MEOO, BlindPick, eOO, and FIFO.

Scenario 3 (HUGE):

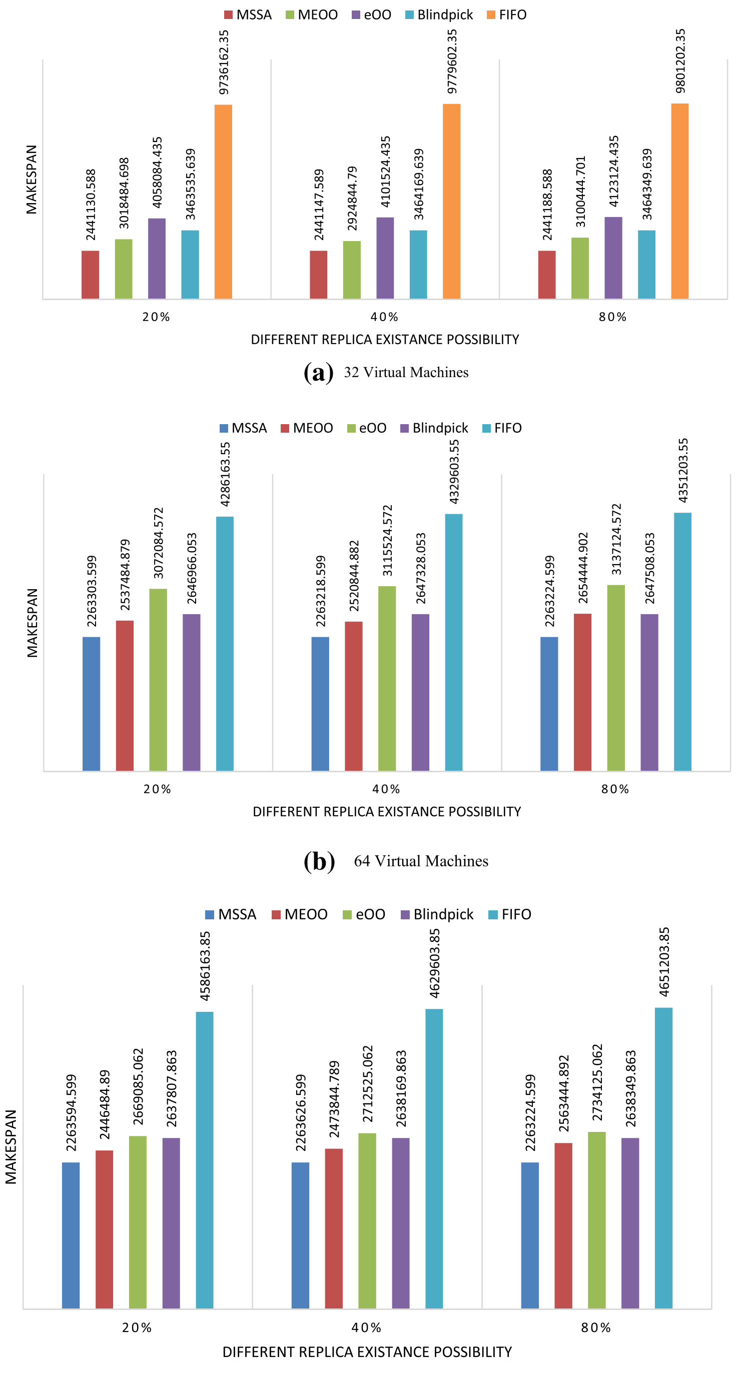
In the final scenario, with increasing the length of workflows, our method shows more improvement in comparison with the others. The execution time and the method overhead are considerably less than other methods. In other words, it showed about 15, 20, 25, and 30% improvement in comparison with MEOO, eOO, BlindPick, and FIFO algorithms. For the sake of brevity, an example of execution time comparison is shown in Fig. 14 and their improvements showed in Table 5.

Table 5 shows the improvement of our method in comparison with other methods. According to Table 5, our method has less simulation time and overhead compared with other methods. According to this table, our method has 20–60% in comparison with eOO. Also, it reaches 90% in comparison to BlindPick. It is 7–20% for MEOO and as expected, the FIFO algorithm has the least overhead in comparison with other methods in comparison with other methods for different replica possibility (Fig. 15).

Like previous scenarios, for the sake of brevity, energy consumption for 32 virtual machines is shown in Fig. 16. As we can see, the proposed method uses about 30 kWh of energy for this scenario. There is a significant improvement in comparison with other methods. This energy consumption is about 35, 63, 52, and 120 in comparison with MEOO, BlindPick, eOO, and FIFO.

6 Conclusion

In this paper, we have extended the ordinal optimization (OO) method and evolutionary ordinal optimization (eOO) method from stochastic virtual machine clustering to the intelligent load-balancing method for scheduling many-task workflow applications. Also, we added replication and message passing parameters to



(a) 32 Virtual Machines

(b) 64 Virtual Machines

(c) 128 Virtual Machines

Fig. 14 The makespan in the 3rd scenario

Table 5 Improvement comparison in scenario 3 (in %)

		20%	40%	80%
32 VMs	MEOO	0.191	0.165	0.212
	eOO	0.398	0.286	0.248
	BlindPick	0.295	0.295	0.295
	FIFO	0.749	0.75	0.75
64 VMs	MEOO	0.108	0.102	0.147
	eOO	0.263	0.19	0.153
	BlindPick	0.144	0.145	0.145
	FIFO	0.074	0.084	0.11
128 VMs	MEOO	0.151	0.087	0.062
	eOO	0.141	0.141	0.142
	BlindPick	0.506	0.511	0.513
	FIFO	0.074	0.084	0.117

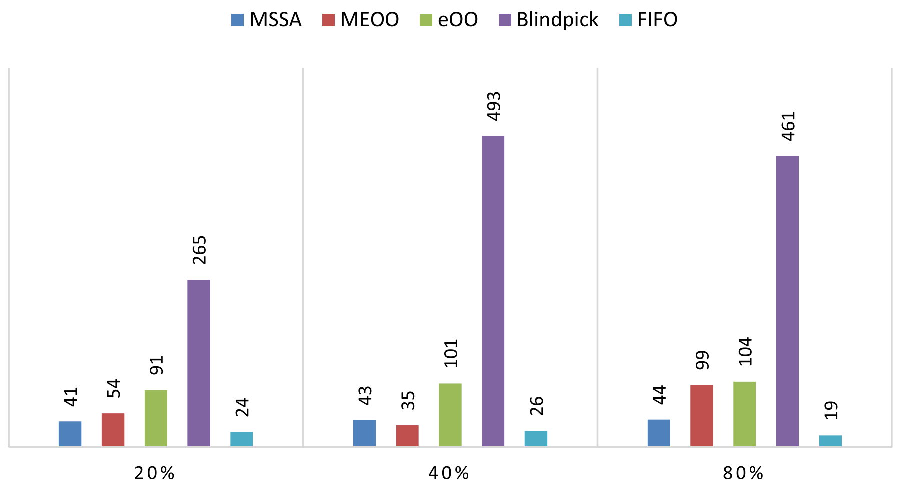
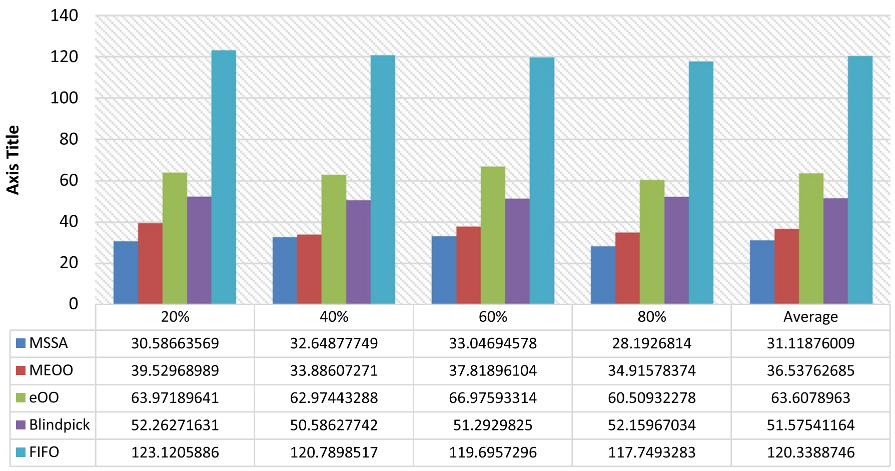


Fig. 15 Simulation time for 128 virtual machines for different replica possibility (in seconds)

the similarity measure to be closer to reality. Our main technical contributions are summarized below.

(1) We proposed a Chi-square-based method for evaluating the similarity between two consecutive periods. This method includes tasks' similarity, replications' similarity, and message passing similarity. So, it shows a good performance in evaluating periods' similarity.
(2) We achieved scalability on a virtualized platform. The MSSA method reduced the makespan significantly: also, MSSA showed less overhead in comparison with MEOO, BlindPick, FIFO, and eOO.



	20%	40%	60%	80%	Average
🟦 MSSA	30.58663569	32.64877749	33.04694578	28.1926814	31.11876009
🟥 MEOO	39.52968989	33.88607271	37.81896104	34.91578374	36.53762685
🟩 eOO	63.97189641	62.97443288	66.97593314	60.50932278	63.6078963
🟪 Blindpick	52.26271631	50.58627742	51.2929825	52.15967034	51.57541164
🟦 FIFO	123.1205886	120.7898517	119.6957296	117.7493283	120.3388746

Fig. 16 Energy consumption for the 2nd scenario and with 32 virtual machines (kWh)

(3) We achieved significant improvement in terms of energy consumption in comparison with eOO, MEOO, BlindPick, and FIFO.

For further research, we suggest extending the work for evaluating this method on a real platform with a real data set.

References

1. Schomm F, Stahl F, Vossen G (2013) Marketplaces for data: an initial survey. ACM SIGMOD Rec 42(1):15–26
2. Sedaghat M, Hern F, Elmroth E (2011) Unifying cloud management: towards overall governance of business level objectives. In: 2011 11th IEEE/ACM international symposium on cluster, cloud and grid computing. IEEE, pp 591–597
3. Panda SK, Jana PK (2015) Efficient task scheduling algorithms for heterogeneous multi-cloud environment. J Supercomput 71(4):1505–1533
4. Djebbar EI, Belalem G (2013) Optimization of tasks scheduling by an efficacy data placement and replication in cloud computing. In: International Conference on Algorithms and Architectures for Parallel Processing. Springer, pp 22–29
5. Duy TVT, Sato Y, Inoguchi Y (2010) Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In: 2010 IEEE international symposium on parallel and distributed processing, workshops and PhD forum (IPDPSW). IEEE, pp 1–8
6. Rani BK, Babu AV (2015) Scheduling of big data application workflows in cloud and inter-cloud environments. In: 2015 IEEE International Conference on Big Data (big data). IEEE, pp 2862–2864
7. Zhang F, Cao J, Hwang K, Li K, Khan SU (2014) Adaptive workflow scheduling on cloud computing platforms with iterativeordinal optimization. IEEE Trans Cloud Comput 3(2):156–168
8. Xiao P, Hu Z-G, Zhang Y-P (2013) An energy-aware heuristic scheduling for data-intensive workflows in virtualized datacenters. J Comput Sci Technol 28(6):948–961
9. 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
10. Madni SHH, AbdLatiff MS, Coulibaly Y (2016) Resource scheduling for infrastructure as a service (IAAS) in cloud computing: challenges and opportunities. J Netw Comput Appl 68:173–200

11. Smanchat S, Viriyapant K (2015) Taxonomies of workflow scheduling problem and techniques in the cloud. Futur Gener Comput Syst 52:1–12
12. Alkhanak EN, Lee SP, Khan SUR (2015) Cost-aware challenges for workflow scheduling approaches in cloud computing environments: taxonomy and opportunities. Futur Gener Comput Syst 50:3–21
13. 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
14. Zhang F, Cao J, Li K, Khan SU, Hwang K (2014) Multi-objective scheduling of many tasks in cloud platforms. Futur Gener Comput Syst 37:309–320
15. Hanani A, Rahmani AM, Sahafi A (2017) A multi-parameter scheduling method of dynamic workloads for big data calculation in cloud computing. J Supercomput 73(11):4796–4822
16. Navimipour NJ (2015) Task scheduling in the cloud environments based on an artificial bee colony algorithm. In: International Conference on Image Processing, pp 38–44
17. Qin P, Dai B, Huang B, Xu G (2015) Bandwidth-aware scheduling with SDN in Hadoop: a new trend for big data. IEEE Syst J 11(4):2337–2344
18. Mashayekhy L, Nejad MM, Grosu D, Zhang Q, Shi W (2014) Energy-aware scheduling of mapreduce jobs for big data applications. IEEE Trans Parallel Distrib Syst 26(10):2720–2733
19. Bodík P, Menache I, Naor J, Yaniv J (2014) Deadline-aware scheduling of big-data processing jobs. In: Proceedings of the 26th ACM symposium on parallelism in algorithms and architectures, pp 211–213
20. Abouelela M, El-Darieby M (2016) Scheduling big data applications within advance reservation framework in optical grids. Appl Soft Comput 38:1049–1059
21. Li X, Song J, Huang B (2016) A scientific workflow management system architecture and its scheduling based on cloud service platform for manufacturing big data analytics. Int J Adv Manuf Technol 84(1–4):119–131
22. Gautam JV, Prajapati HB, Dabhi VK, Chaudhary S (2015) A survey on job scheduling algorithms in big data processing. In: 2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT). IEEE, pp 1–11
23. Wang K, Raicu I (2014) Scheduling data-intensive many-task computing applications in the cloud. In: NSFCloud workshop
24. Bardhan S, Menascé DA (2014) A contention aware hybrid evaluator for schedulers of big data applications in computer clusters. In: 2014 IEEE International Conference on Big Data (big data). IEEE, pp 11–19
25. Zhao Y, Fei X, Raicu I, Lu S (2011) Opportunities and challenges in running scientific workflows on the cloud. In: 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. IEEE, pp 455–462
26. Dashti SE, Rahmani AM (2016) Dynamic VMs placement for energy efficiency by PSO in cloud computing. J Exp Theor Artif Intell 28(1–2):97–112
27. Lorch JR, Smith AJ (2001) Improving dynamic voltage scaling algorithms with PACE. ACM SIGMETRICS Perform Evaluat Rev 29(1):50–61
28. Lee YC, Zomaya AY (2010) Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans Parallel Distrib Syst 22(8):1374–1381
29. Topcuoglu H, Hariri S, Wu M-Y (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
30. Wang L, Von Laszewski G, Dayal J, Wang F (2010) Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. IEEE, pp 368–377
31. Kimura H, Sato M, Hotta Y, Boku T, Takahashi D (2006) Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster. In: 2006 IEEE International Conference on Cluster Computing. IEEE, pp 1–10
32. Tang Z, Qi L, Cheng Z, Li K, Khan SU, Li K (2016) An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment. J Grid Comput 14(1):55–74
33. Zhong X, Xu C-Z (2007) Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee. IEEE Trans Comput 56(3):358–372
34. Bini E, Buttazzo G, Lipari G (2009) Minimizing CPU energy in real-time systems with discrete speed management. ACM Trans Embed Comput Syst (TECS) 8(4):1–23
35. Quan G, Hu XS (2007) Energy efficient dvs schedule for fixed-priority real-time systems. ACM Trans Embed Comput Syst (TECS) 6(4):29

36. Zhuo J, Chakrabarti C (2008) Energy-efficient dynamic task scheduling algorithms for DVS systems. ACM Trans Embed Comput Syst (TECS) 7(2):1–25
37. Juarez F, Ejarque J, Badia RM (2018) Dynamic energy-aware scheduling for parallel task-based application in cloud computing. Futur Gener Comput Syst 78:257–271
38. Duan H, Chen C, Min G, Wu Y (2017) Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems. Futur Gener Comput Syst 74:142–150
39. Wen Y, Liu J, Dou W, Xu X, Cao B, Chen J (2020) Scheduling workflows with privacy protection constraints for big data applications on cloud. Futur Gener Comput Syst 108:1084–1091
40. Elhoseny M, Abdelaziz A, Salama AS, Riad AM, Muhammad K, Sangaiah AK (2018) A hybrid model of internet of things and cloud computing to manage big data in health services applications. Futur Gener Comput Syst 86:1383–1394
41. Alboaneen D, Tianfield H, Zhang Y, Pranggono B (2021) A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers. Futur Gener Comput Syst 115:201–212
42. Zhao Q, Xiong C, Yu C, Zhang C, Zhao X (2016) A new energy-aware task scheduling method for data-intensive applications in the cloud. J Netw Comput Appl 59:14–27