MODE: A Multi-Objective Strategy for Dynamic Task Scheduling through Elastic Cloud Resources

MinaYazdanbakhsh[1], Reihaneh Khorsand Motlagh Isfahani[2*], Mohammadreza Ramezanpour [3]
1- Department of Computer Engineering, Dolatabad Branch, Islamic Azad University, Isfahan, Iran.
Email: st_yazdanbakhsh@yahoo.com
2- Department of Computer Engineering, Dolatabad Branch, Islamic Azad University, Isfahan, Iran.
Email: Reihaneh_khm@yahoo.com (Corresponding author)
3- Department of Computer Engineering, Mobarakeh Branch, Islamic Azad University, Isfahan, Iran.
Email: ramezanpour@mau.ac.ir

Received: July 2019 Revised: October 2019 Accepted: December 2019

ABSTRACT:
Cloud computing is introduced as a high-performance computing environment that manages a variety of virtualized resources. One of the major aspects of cloud computing is its dynamic scheduling of great number of task requests that are submitted by users. Cloud data centers in addition to implementing these tasks, should meet the conflicting multiple requirements of different users. Minimizing makespan and deadline violation on a great number of tasks are difficult while costs are reduced. Therefore, in this paper, a multi-objective strategy for dynamic task scheduling through elastic cloud resources (MODE) is proposed, where an algorithm is proposed to construct individual non-dominated sets of new received tasks. These non-dominated sets are sorted in different levels through a new crowding distance of the individuals. In addition, an elastic resource provisioning based on the maximum available VMs' load is proposed to provide resources in a dynamic manner. The total cost, makespan, and the deadline violations are reduced by 85.84%, 58.03%, and 47.77%, respectively, and the utilization of virtual machines is increased up to 53.2% through this strategy when compared to its counterparts.

KEYWORDS: Cloud Computing, Dynamic Task Scheduling, Multi-Objective Scheduling, Elasticity, Quality of Service.

1. INTRODUCTION

Cloud computing is a computing model based on dynamic provisioning of hardware, software, or services applicable in a pay-as-you-go method [1]. The concept of Virtual Machines (VMs) which act as the computational units is applied in this model. Depending on the computational needs of the tasks, the new VMs can be leased and released in a dynamic manner [2]. Many end-users can request services from cloud at any time. In these situations, the number of tasks, and the available resources, can rise on demand. Calculating and selecting all possible task-resource mappings in cloud computing is almost impossible because the complexity here would grow in an exponential manner. To reduce the complexity of the search space, a heuristic algorithm assures an acceptable runtime of the scheduling algorithm. Many heuristic optimization algorithms have been and are being applied in many works to optimize task scheduling, mostly in minimizing the task execution time in the cloud environment [3]. However, most of the existing scheduling strategies have more than one conflict objectives while executing the applications in the Cloud data center. These problems are referred to as Multi-Objective Optimization Problems (MOPs). There might be a state of conflict within these objectives, where no single solution exists. In this context, a good *trade-off* solution can be devised which would represent the best possible compromises among scheduling objectives [4-6]. Multi-objective scheduling problems can be divided into two groups of: priority-based methods with a weight assigned to each objective (classical methods) and methods including a set of non-dominated solutions [7]. Classical methods have the following three basic drawbacks compared to non-dominated solution: first, these methods are not able to search for all the permissible space related to the problem, second, are not considered as an intelligent method and third, the objective functions of these methods need to be normalized for summation. To solve these drawbacks, development of a dynamic task scheduling strategy where a set of non-dominated solutions is of concern for multiple and conflicting objectives, is a critical part of task scheduling optimization problems. Among the available studies, authors of [8-10] mainly emphasize on the optimization of the multi objectives in their scheduling models, which are not in conflict with each other. In addition, there are several studies [11-19] that mainly emphasize the minimization of job makespan and task execution cost in their multi-objective optimization models by applying evolutionary algorithms. However, these studies fail to consider the need to improve VM utilization by the cloud infrastructure. Meanwhile, there is a considerable amount of research work that fails to consider the dynamic resource provisioning by the cloud infrastructure. Also, to the best of our knowledge, applying a Pareto-based method able to make trade-offs within sets, instead of considering every performance metrics' full range has not been investigated in the proposed task scheduling optimization models. To improve previous research work and address these shortcomings in the existing literature, in this paper, a multi-objective approach for dynamic task scheduling through elastic resources in cloud environments is proposed. The efficiency of the proposed model is evaluated through different scenarios. Simulation results show that the proposed model significantly meets the multi-objective QoS requirements for optimization, in both cloud users' and providers' context by minimizing makespan and deadline violations for user, and minimizing total cost and improving VM utilization for providers. In fact, the proposed model is able to determine trade-off solutions that offer the best possible compromises among the optimization objectives. It has also been found that MODE is a faster and more accurate evolutionary algorithm than its counterparts for solving such problems. The main contribution of this study is briefed as follows:

- Designing a multi-objective task-scheduling framework where point of views of both users and provider in minimizing are of concern makespan, deadline violation, total cost and maximizing the resource utilization that are in conflict with one another.
- Applying a Pareto-based method able to make trade-offs within sets, instead of considering every performance metrics' full range.
- Proposing a task scheduling algorithm named MODE to construct the individual non-dominated sets of the newly received tasks where the non-dominated sorting approach is applied to generate Pareto fronts.
- Applying a new crowding distance operator to generate a uniformity in the distribution of the individuals in the Pareto optimal frontier.
- Applying the dynamic lease and release of resources, which can reduce the additional costs to maintain unusable VMs.
- Applying a series of experiments to evaluate this proposed approach's performance subject in different experiment conditions.

The structure of this paper is as follows: Section 2 provides an overview of related works. Section 3 describes the proposed MODE strategy acting in a multi-objective and dynamic manner. Section 4 provides an experimental design and discusses the experimental results. Section 5 concludes the paper and presents future works.

2. LITERATURE REVIEW

The contribution of task-scheduling problem is important in optimizing cloud utilization [11-13]. The scheduling algorithms can be classified based on the improvement parameters in the cloud computing environments like load balancing, cost, priority, makespan, and resource utilization.

To improve the load balancing in the cloud computing, Babu and Samuel [14] proposed an improved honey colony algorithm where the bee's behavior is applied to balance load through VMs. Patel et al. [15] modified the Min-Min load balancing algorithm, which is based on the effect of the Min-Min load balancing algorithm on the grid computing. The tasks with a minimum makespan are selected and allocated to the appropriate resources for producing the makespan time and the use of resources effectively through this algorithm. A Group Scheduling Algorithm (GTS) is proposed by Gamal et al. [16] to schedule tasks in a cloud computing with respect to quality of service requirements. This algorithm is evaluated through different performance metrics like execution time, load balancing, and average latency. The drawback of this algorithm is that it does not consider the elasticity of the cloud.

To improve the cost in the cloud computing, Lakra and Yadav [17] proposed a multi-objective task scheduling algorithm for mapping tasks to VMs which improves the throughput of the data center and reduces costs without Service Level Agreement (SLA) violation for the software. It is necessary for this algorithm to lease and release the new VM which are assessed here. A scaling approach equipped with a super professional executor named Suprex with a cost-aware approach is proposed by Aslanpour et al. [18]. Their evaluation results indicate that Suprex can reduce resource rental costs for the application providers, together with a decrease in both the response time and SLA violation. Gabi et al. [19] proposed a QoS task scheduling algorithm with *Taguchi* optimization approach for cloud computing. Their proposed algorithm indicates better cost and time values than that of standard CSO, MOPSO, EPCSO, and OTB-CSO algorithms. The drawbacks of their approach is in applying static resource provisioning, moreover, non-consideration of the important performance metrics like the utilization of VMs, makespan, and deadline violations. To improve the priority-based task scheduling, Baofang et al. [20] designed a Parallel Adaptive Genetic Algorithm (PAGA) based on the priority mechanism in the cloud when dealing with the problem of assigning priority to the tasks and sub-tasks, while their algorithm is of high complexity. Based on Analytic Hierarchy Process (AHP) theory, Ghanbari and Othman [21] proposed a job scheduling algorithm applying a multi-criteria decision model. To improve the makespan in the cloud computing, Bhoi and Ramanuj [22] chose the expected execution instead of the completion time as the base with the objective to schedule several jobs on several machines in a dynamic manner to decrease makespan and increase efficiency. Goyal and Agrawal [23] proposed a scheduling model based on the principles of improved genetic algorithm. Panda et al. [24] proposed three allocation-aware task-scheduling algorithms for multi-cloud environments. The traditional Min-Min and Max-Min algorithms are the source of this algorithms developed for multi-cloud environment. In these algorithms matching, allocating, and scheduling are the common stages to adapt multi-cloud environment.

To improve the resource utilization in the cloud computing, Islam et al. [25] revealed the elasticity metrics based on penalties for over and under-utilization of resources. Shawky and Ali [26] presented a method to measure the cloud elasticity in reference to elasticity in physics. Belteran et al. [27] introduced a new metric of elasticity capable of considering the scalability, accuracy, time and cost independent main components where an approach is provided to assess behavior of the service elasticity. An autonomic resource provisioning method is proposed by Ghobaei-Arani et al. [28] according to the MAPE-k (monitoring-analysis-planning-execution control loop with a shared Knowledge) control loop. This approach should adapt to uncertainties and workload spikes in a dynamic manner, and should manage undesirable states of over and under-provisioning. The detailed of the task scheduling methods are tabulated in Table 1.

Table 1. Summary of cloud based task scheduling algorithms.

Literature	Algorithm	Types of Classification	Performance metrics	Elasticity
[14]	Enhanced Bee Colony Algorithm for Efficient Load Balancing	Load balancing	Response time, Load balancing	-
[15]	Enhanced Load Balanced Min-Min task scheduling algorithm (ELBMM)	Load balancing	Makespan, VM Utilization	-
[16]	Grouped Tasks Scheduling Algorithm (GTS)	Load balancing	Average latency to expected urgent priority of tasks with urgent users	-
[17]	Multi-objective task scheduling algorithm with the goal of reducing costs	Cost	Throughput, execution time, bandwidth	-
[18]	An executor for the cost-aware auto-scaling mechanism (Suprex)	Cost	SLA violation, resource rental costs, response time	-
[19]	A QoS task scheduling algorithm with Taguchi optimization approach	Hybrid	average execution time, cost	-
[20]	An improved Adaptive Genetic Algorithm (PAGA)	Priority	Iteration times, Bandwidth	-
[21]	A new Priority based Job Scheduling algorithm (PJSC)	Priority	Consistency, makespan,	-
[22]	Enhanced Max-Min task scheduling Algorithm	makespan	Makespan,	-
[23]	Genetic algorithm coupled with suffrage heuristic	makespan	Response time	-
[24]	Allocation-aware Min-Min Max-Min batch algorithm (AMinMaxB)	makespan	Makespan, Average utilization, Throughput	-
[25]	Improved ways to quantify the elasticity concept, using data available to the consumer	Elasticity	Elasticity	✓
[26]	Defining a Measure of Elasticity	Elasticity	Elasticity	✓
[27]	A new approach to analyse elasticity enablers of cloud services	Elasticity	Scalability, Accuracy, Time, Cost	✓
[28]	An autonomic approach for resource provisioning of cloud services	Resource utilization	Total cost, resource utilization, SLA violation	-
The proposed approach	A Multi-Objective Strategy for Dynamic Task Scheduling using Elastic Cloud Resources	Hybrid	Total cost , makespan, deadline violations, mean utilization of VMs	✓

Some studies apply task scheduling based on single criteria [13-14]. In the cloud computing environment, single-objective scheduling algorithms encounter some problems, like, high-priority tasks with high chance to be run, and tasks with low priority which have to wait for a long time. Sometimes tasks with low priority receive an opportunity to be run while high-priority tasks always are run before the low priority task, which lead to an increase in execution time and a decrease in system throughput. It only satisfies one user's requirements at execution time, and the user does not achieve other objectives [15], [29]. Some of the previous algorithms [17, 29] may reduce the cost but violate the deadline, which will not allow the overall costs to be reduced. Another notable challenge is that most prior research works [15, 30-32] apply a fixed number of VMs at execution time, which are not flexible.

Table 2. Definitions of variables.

Symbol	Definition
n	The number of arrival tasks
T	The set of arrival tasks={t_1, t_2, …, t_n}
VM_j	Virtual machine j, j={1, 2, …, k}
Task_Load	The task load
Total_Task_Load	The total load of users' requests
Maximum_VM_Mips	The maximum VM Mips (VM Mips × its Cores)
System_Total_Rate	The system total rate
VM_Normal_Rate	The normal rate of VM
Rental_Vm_Counts	The leased VM counts
m	Total number of VMs including the number of initial VMs plus leased VMs.
Cost_of_VM	The VM Cost
$Task\_Execution\_Time_k$	The execution time of the tasks allocated to the VM_k
r	The number of service QoS attributes
$q_k(s)$	The *k*-th QoS attribute value of service *s*
P_k	The priority of the user
$L_{i,j}$	Load of VM
$Q_{j,k}^{max}$	Maximum value of k-th QoS attribute of all candidate services belonging to service class S respectively.
$Q_{j,k}^{min}$	Minimum value of k-th QoS attribute of all candidate services belonging to service class S respectively.
$Task\_Size_k$	The size of k-th task

3. PROPOSED APPROACH (MODE)

Multi-objective optimization problem devises a set of points known as the Pareto optimal set. The dynamic task scheduling is a multi-objective optimization problem with the most important objectives to minimize 1) the overall execution time or makespan, 2) deadline violation of a set of tasks, and 3) the total cost in both the customer and provider context. Due to independent and conflicting nature of these objectives, reducing one objective would lead to compromising the other. The details of the basic notation and their definitions used in the proposed approach are tabulated in Table 2.

In task scheduling approach, there exist n tasks { t1 , t2,…, tn } with variable task receiving rates assignable to k VMs { VM1, VM2,…, VMk } for execution. In this paper, a multi-objective strategy named MODE is proposed for dynamic task scheduling through elastic cloud resources according to the improved non-dominated sorting algorithm. The framework of this proposed approach for multi-objective task scheduling problem is shown in Fig. 1.

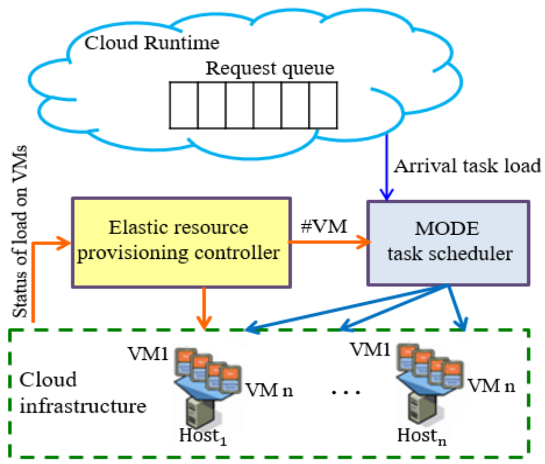

Fig. 1. The perspective of MODE scheduling framework in cloud environments .

Cloud runtime receives tasks from users at first. The submitted tasks from the users are inserted in the Request queue and the MODE task scheduler is called. The objective of the MODE scheduler is to generate the individual non-dominated sets of the newly received tasks, which are sorted by the crowding distance in different levels. Individuals at higher non-dominated sets level and greater crowding distance have higher priority to the next level. If the points with the same rank for the problem of task scheduling are selected randomly, the distribution of the solution in the optimal solution set would not be met. To make this distribution consistent in the Pareto optimal front, a new crowding distance operator is applied in this paper. Therefore, the MODE scheduler performs the final assignment of tasks to resources. In step 2, during runtime, Elastic resource provisioning controller checks resources status in λ time intervals and if there exists any overload in the VM, it will create a new VM for scheduling in next step.

The pseudo-code of the proposed MODE (algorithm1) is described as follows:

Step 1: The initial number of VMs is determined based on the received task load and status of load on VMs is determined according to Eq. (1), (Line 1)

Step 2: For each VM, the QoS utility function is calculated through the Eq. (3), and then the VMs are sorted in a descending order based on these values (Line 2).

Step 3: The set of received tasks are sorted by the non-dominated sorting method in a descending order, as shown in Eqs. (4 and 5) based on the size and cost of the task execution performance metrics.

In addition, the cost of executing the task is obtained through Eq. (6) in a sense that the cost of each VM obtained from Eq. (7) is multiplied by task execution time obtained from Eq. (8), (Line 3)

Step 4: After sorting tasks and VMs, some tasks will be placed on the execution list according to their priorities. Then, dispatcher component allocates the first task from the execution list to the first VM at the beginning of the sorted VM list, while the normal rate of each VM is calculated through Eq. (9) and is compared with the threshold rate until the normal rate becomes greater than the threshold. If its available workload on VM plus the next task load exceeds the threshold, this task will be allocated to the next VM. This algorithm re-calculates VM load to find the appropriate VM. This process is repeated until all tasks from the list are assigned to the proper VMs (Lines 4-12).

Step 5: When the second series of tasks are received, the tasks of the list are resorted with respect to the new tasks. Then, according to Eq. (9), the total rate of each VM is calculated, and compared with the threshold rate. If the total rate system is lower than that of the threshold rate, the new set of the received task are executable on VMs that were devised at the beginning. Further, the maximum workload of existing VMs applied in Eq. (10) is calculated through Eq. (11), otherwise, the system is overloaded, and the new VM number must be leased by applying the calculated value of Eq. (12). At this point, the new VMs are sorted and tasks are allocated to VMs (Lines 13-18)

Step 6: In case of a deadline violation, the provider adds a specified value based on Eq. (13), for each delay in response time to the service being served in a time unit (Lines 19-21)

Step 7: In every step, if the load on the leased VMs reaches to zero, they are considered to be excessive and eliminated (Lines 22-24)

Algorithm 1: Proposed MODE algorithm.

```
Input: A set of different types of tasks with variable entry rates, deadline and cost of tasks parameters.
Output: A set of scheduling and allocation results for tasks.

1.  Determine the number of VMs according to the arrival tasks load that are calculated by Eq.1.
2.  Calculate of the QoS_Utility(s) function for VMs by Eq.3, and sort VMs descending order.
3.  Sort tasks by non-dominated sorting based on task size and task execution cost, that are calculated by Eqs.4 and 6
4.  for i ← 1 to Size of VM's list do
5.          for j ← 1 to Size of task's list do
6.              Calculate VM_Normal_Rate by Eq.9.
7.              if $VM_i$_Normal_Rate < Threshold_rate then
8.                      Resource allocation $task_j$ to $VM_i$.
9.              end if
10.         end for
11. end for
12. a new set of task arrived
13. Calculate System_Total_Rate by Eq.10.
14.       if System_Total_Rate < Threshold_rate then
15.             New arrived tasks are executable on available VMs.
16.             else system is overloaded.
17.                 rent a VM according to the calculated number by Eq.12.
18.       end if
19. if deadline missed then
20.       Penalty cost= deadline missed seconds * penalty cost per second (calculated by Eq.13).
21. end if
22. if new VM load == 0 then
23.       Delete VM.
24. end if
25. Return a set of task scheduling solutions, VM counts
```

3.1. Elastic Resource Provisioning Controller

3.1.1. Determining the number of VMs

To compromise cost reduction and a decrease of deadline violations, it is essential to prevent over-loaded and under-loaded VMs. In this approach, the number of VMs is determined according to the received task load. The initial number of required VMs is calculated by dividing the sum of received task load to current status of load on VMs (the maximum MIPS value of VMs) through Eq. (1).

$$Vm\_Counts = \frac{Total\_Task\_Load}{Maximum\_VM\_Mips} \qquad (1)$$

Where, Eq. (2) is applied in calculating the total received task load as follows:

$$Total\_Task\_Load = \sum_{k=1}^{n} Task\_Load \qquad (2)$$

Where, n is the task count.

3.1.2. Sorting VMs

After determining the required VMs count, sorting VMs become must take place. QoS is applied in describing user's requirements. Different users require different cloud computing services. In this paper, the service execution time, cost, and bandwidth are applied to describe QoS on resource services. Here, the QoS_Utility(s) function (Li , 2014) are obtained through Eq. (3), to map QoS attributes vector Qs={ q1(s), q2(s), ..., qr(s)} of each candidate service to one real value.

$$QoS\_Utility(s) = \sum_{k=1}^{r} \frac{Q_{j,k}^{max} - q_k(s)}{Q_{j,k}^{max} - Q_{j,k}^{min}} \times P_k \qquad (3)$$

Where, r is the count of QoS attributes of VM, qk(s) is the kth qualitative attribute value on the service s. Pk is the priority of the user. $Q_{j,k}^{max}$ and $Q_{j,k}^{min}$ are the maximum and minimum values of the kth attribute of the QoS of all the candidate VMs. After calculating the QoS utility function for each VM, according to the values obtained for each VM, they are sorted in a descending order. According to this proposed approach, tasks with higher cost and size are sent to VMs with high processing power.

3.2. MODE Task Scheduler

3.2.1. Sorting tasks by non-dominated sorting method

When a multi-objective algorithm is applied to solve a problem, at least two objective functions are of concern, where it is not easy to give a definitive opinion about some of the responses. In most cases, there exist points where none has priority on other, and they cannot be compared with the domination concept. Therefore, in order to obtain the best responses, they should be sorted according to a certain standard. In this paper, for handling multi-objective scheduling, a Pareto-based method is applied for the non-dominated solutions to be selected and to direct the study towards the true Pareto-optimal front. In this process, a rank is allocated to each response, based on the number of their domination compared to other points. The points with the best rank, 1, and the lowest domination are chosen as the response set or points of the Pareto fronts. Allocating goods among individuals where none can improve his/her situation without worsening another's is defined as Pareto efficiency; the individuals of such set generate a Pareto frontier curve, the Pareto front, which is particularly applicable where designers would make trade-offs within sets, instead of considering full range of every parameter's. After constructing the individual non-dominated sets of the new received tasks, the non-dominated sets are sorted through the crowding distance of the individuals in different levels.

3.2.2. Construct a Non-dominant Set

In sorting tasks through the non-dominated sorting method, the objective here is to select a set of tasks with the minimum task size, which affects the overall execution time or makespan and the execution cost from the customer perspective. Therefore, in this proposed approach the two objective functions are obtained through Eqs. (4 and 5).

$$Min\ f(Task_{Size_k}) = Task_{Size_k} | \forall\, j\ \exists\, i,\quad f(Task\_Size_i) \le f(Task\_Size_j) \qquad (4)$$

$$Min\ g(Task_{execution\ cost_k}) = Task_{execution\ cost_k} \mid \forall\, j\ \exists\, i,\ f(Task\_execution\_cost_i) \le f(Task\_execution\_cost_j) \qquad (5)$$

Where, $Task_{Size}$ is the task size, $Task_{execution\ cost}$ is the cost of executing the task, T is a set of tasks = {t1, t2, …, tn} and n is the count of tasks. To implement a multi-objective task-scheduling algorithm through these two functions, the non-dominated sorting is applied.

The cost of executing tasks, which is one of the intended objectives in non-dominated sorting, is calculated through Eq. (6):

$$Task\_execution\_cost = \sum_{k \in SPp} Cost\_of\_VM_k \times Task\_Execution\_Time_k \qquad (6)$$

Where, SPp is the set of VMs of the pth provider (

SPp={ k | VMk ∈ P-th cloud provider, P∈ {1,2,…,cp} }), and cp is the count of cloud providers. The $Cost\_of\_VM_k$ is the cost of a CPU unit for the kth provider, which can be calculated by Eq. (7), and $Task\_Execution\_Time_k$ is the execution time of the tasks allocated to the VM_k, calculated through Eq. (8):

$$Cost\_of\_VM = Cost\_Per\_Second() / Mips\_Of\_One\_Pe() \quad (7)$$

$$Task\_Execution\_Time_k = \sum_{i=1}^{n} \frac{Length \times PE + OutPutSize}{MIPS \times PE} \quad (8)$$

To shade more light on the concept of domination, Fig. 2 is drawn. A number of points are specified in the space of all possible solutions of the problem where each has two objective functions $f(Task\_Size_k)$ and $f(Task\_execution\_cost_k)$.

The status of point 2 Vs. other points in the page is checked, indicating that point 2 dominates all members in range A. The value of the objective functions $f(Task\_Size_k)$ and $f(Task\_execution\_cost_k)$ for this point in relation to the value of the objective functions $f(Task\ Size_k)$ and $f(Task\_execution\_cost_k)$ for all points on page A is low and it is dominated by all points in space C. In this process, sometimes the candid opinions cannot be provided about the superiority of points. Consequently, the points in spaces B and D cannot be directly judged compared to point 2 because the points on page B, on the function $f(Task\_execution\_cost_k)$, are better than 2 and worse than 2 for the function $f(Task\_Size_k)$. Through this direct comparison, one cannot claim which point dominates the other, and in such cases, presence of other members of the population becomes contributive in judgment.

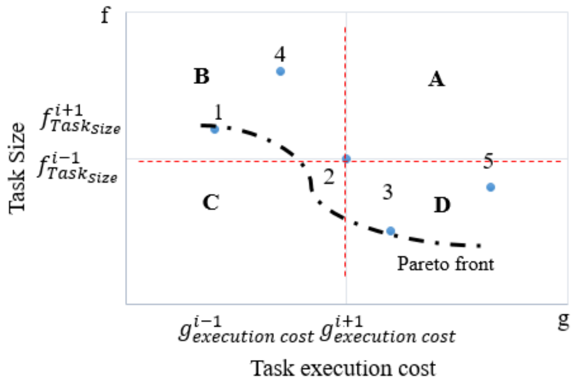

Fig. 2. Domination concept for sorting tasks.

In the same context, as observed because there is no point in space C, a comparison is made between points 2 and 4 in space B. In this situation, it should be checked whether there exists another point which is better than these points in terms of both the objective functions: If point 1 is better than point 4, point 1 dominates point 4, while none of these points are better than point 2, indicating that point 4 is dominated by other members of the population once, while point 2 in this condition is never dominated. Consequently, point 2 has a better chance of being chosen between either of points 1 and 4. This fact holds true for points 2 and 5. As to the points 1 and 3, it is not possible to comment on 2, because these points are not dominated by any point, and each has a superiority and non-superiority over the other. Since, points 1, 2 and 3, which have never been dominated and ranked 1, constitute the Pareto front points. Here, the points with the same ranking are selected by the crowding distance of the individuals.

3.2.3. Improving the crowding distance calculation

Different dimensions of optimization objectives like the task execution cost and execution time (task size) generate the individuals of the optimal Pareto front that show a big gap on these sub-objectives while the crowding distance is calculated. The traditional multi-objective optimization algorithms obtain the crowding distance of an individual through calculating the sum of distance difference between the individual and two individuals next to it in each sub-objective and it does not consider the influence brought by the different sub-objective. According to the traditional crowding distance operator, a normalizing technique is applied on the sub-objectives fitness where the individual is located, weakening the influence of the crowding distance calculation because of the difference of the dimension, in a sense that make the distribution of individuals in optimal Pareto frontier better. To obtain a uniform measurement between the task execution cost and time, the Simple Additive Weighting (SAW) technique [33] is applied to normalize these two scheduling objectives. The fitness value of the individual will increase by reducing the task execution cost and time. Thus, when calculating the crowding distance, the sub-objective fitness is normalized where the individual is located, it can further weaken the situation where the individual distribution is not ideal in the optimal Pareto front due to the difference between the sub-objectives. Therefore, we improve the crowding distance of the individual as follows:

$$\text{Crowding distance} = \left|f^{i+1}_{-TaskSize} - f^{i-1}_{-TaskSize}\right| + \left|f^{i+1}_{-ExecutionCost} - f^{i-1}_{-ExecutionCost}\right| \quad (9)$$

Where, $f^{i-1}$and $f^{i+1}$are previous and next individuals to the current individual in each sub-objective, respectively.

In the proposed MODE, the non-dominated sorting and crowding distance calculation operations in each execution, and the time complexity for these two operations is $O(M(2N)^2+M*2N*\log(2N))$ where M and N are the number of objectives and the population size, respectively.

3.2.4. Mapping of tasks to VMs

After sorting the tasks and VMs, at first, a number of tasks are placed on the execution list according to their priority, next, the first task from the execution list is allocated to the first VM at the beginning of the sorted VM list. The normal rate of each VM is calculated according to Eq. (10) and is compared to the threshold rate. If the normal rate is lower than the threshold rate, the next task becomes executable on VMs generated first. In a similar sense, the next task is allocated to VMs from the list until tasks are entirely allocated to the existing VMs, otherwise, if the normal rate is greater than the threshold rate, the system becomes overloaded, and a new VM must be leased. To avoid/minimize the deadline violations caused service response delay, this configuration is chosen.

$$VM\_Normal\_Rate = \frac{Current\_Workload\_of\_a\_VM}{Maximum\_VM\_Mips} \quad (10)$$

3.2.5. Lease and release of resources

National Institute of Standards and Technology (NIST) defines elasticity in cloud computing as: "capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand" [34]. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time. In this paper, a resource-provisioning approach is proposed to reduce the total cost and deadline violation. In addition, deadline violations are due to the elapse in the expected deadline, and where a monetary penalty is imposed, consequently, cost minimization is expressed as follows:

$$\min(TotalCost) = \text{VM cost} + Penalty\ Cost \quad (11)$$

Where the VM cost is the total cost for all VMs expressed through Eq. (12):

$$VM\ cost = \sum_{i=1}^{I} Cost\_of\_VM_i \quad (12)$$

Where, I is the count of initiated VMs, and i is the VM id. Many of the available algorithms may reduce the cost, where violating the deadline, in a sense that no cost reduction may be observed. Therefore, to calculate the cost in case of deadline violation by imposing penalty on the cost of leasing VM, one can confidently claim that the lower cost is a sign of success. A deadline violation occurs when the execution time of a task exceeds its deadline value. Penalty cost consists of total penalty costs incurred in all task requests, expressed through Eq. (13):

$$Penalty\ Cost = \sum_{i=1}^{C} Penalty\_Cost_i \quad (13)$$

Where, the penalty cost for every task requests is calculated though Eq. (14).

$$Penalty\_Cost_i = Deadline_{Missed_{Seconds}} \times Penalty\_rate \quad (14)$$

Where, penalty-rate is the monetary cost per unit of time delay, $Deadline_{Missed_{Seconds}}$ is delay time. The product of which yields the $Penalty_$cost per \$.

In this paper, elasticity feature is applied to avoid unnecessary additional costs in maintaining not needed VMs. After allocating a set of tasks and increasing workload at execution time, additional VMs can be required at any time. Therefore, first, some VMs are leased and next, released to reduce network load and prevent resource wastes. For this purpose, first, system total rate is calculated to obtain the overload information through Eq. (16). The maximum workload of the existed VMs applied in Eq. (15) is calculated through Eq. (16). If the system is overloaded, new VMs must be leased indicated through Eq. (18). Following this, the new VMs are sorted and, the tasks are allocated to VMs, which in turn, reduce the count of VMs to decrease total cost. The deadline violations of existing task requests are avoided by not allocating new task request to the initiated VM.

$$System\_Total\_Rate = \frac{Total\_Load}{Maximum\_Workload\_of\_existed\_VMs} \quad (15)$$

$$Maximum\_Workload\_of\_existed\_VMs = \sum_{k=1}^{n} Maximum\_VM\_Mips \quad (16)$$

$$Rental_{Vm_{Counts}} = \frac{System_{Total_{Rate}} - Maximum\_Workload\text{Of}existedVM}{Maximum\_VM\_Mips} \quad (17)$$

If the load on a VM is zero, it will be removed and released. If the next series of tasks are to be met and are needed again, they will be reused. The time complexity of lease and release algorithm of VMs is O (ITC+I), where, I is the total VMs count, T is VM types and C is the total count of requests.

4. EVALUATIONS

4.1. Experimental Setup

In this paper, the simulation is run by applying the CloudSim 3.0.3 toolkit [17, 35-36] to accomplish the proposed MODE. The objective of the proposed approach is to meet the multi-objective QoS requirements, in both cloud users' and providers' context by minimizing makespan and deadline violations for user, and minimizing total cost and improving VM utilization for providers. This total cost includes both the execution and penalty cost of both the customer and provider. From the provider view, because there exist no public data on the SaaS provider's spending on VMs, the price schema of Amazon EC2 [36] is applied here to estimate the per hour cost of a hosted VM. The MIPS ratings are applied to simulate the effect of using different VM types. Resource price and capabilities applying in modeling VMs are tabulated in Table 3 similar to that of by Wu et al. [38].

From the customer view, the received request rate varies in evaluating their impact on implementing of this proposed algorithm. The received request rate is subjected to Poisson distribution. Due to lack of available workload as to specify these parameters, standard deviation = (1/2) × mean is applied as the normal distribution in modeling all parameters. The received request rate for 100, 200, 400, 600, 800, 1000 users per second are applied similar to [39] and the tasks are allocated to each VM at the threshold rate of 0.9 similar to [40] in the simulation.

Table 3. Type of VMs.

VM Type	VM Capacity	Pes	Ram	Size	Cost per hour
1	1 CPU unit	1	2 GB	160 G Disk	0.12 $
2	2 CPU unit	2	4 GB	850 G Disk	0.48 $
3	4 CPU unit	4	8 GB	1690 G Disk	0.96 $

4.2. Metrics for Evaluating the Proposed MODE

To compare the proposed approach with its counterparts, the following metrics are applied:

- Makespan: It is the completion time of the task, which includes the execution time and the latency of the cloud system response [30]
- Total cost: It is the sum of VM allocating and penalties of SLA violation costs calculated through Eqs. (11, 12, and 13)
- Mean utilization of the VM: It is the capacity of resources applied during execution time, known as resource utilization. This criterion is calculated through Eq. (18):

$$VM_{Utilization_{Mean}} = \frac{\sum_{i=1}^{m} \frac{\int_{s_i}^{f_i} L_{i,j}}{(Finish\_Time\_of\_each\_VM_i - Start\_Time\_of\_each\_VM_i)}}{m} \quad (18)$$

Where, L(i, j) is the load on VM and m is the total number of VMs including the number of initial VMs plus leased VMs.

- Deadline violation: If the completion time of each task is greater than the deadline, it leads a deadline violation. This performance metric has a direct relation with the total cost. Many specially devised algorithms may reduce costs while they violate the deadline, thus no cost reduction. For this purpose, the system receives the penalty for every second of deadline violation [41-42].

4.3. Evaluation Experiments

The proposed approach is conceptually different from the ones like, and although it is in the same category with a number of them. To evaluate the MODE approach and to have a better view of its advantages, the following algorithms are described which are then compared in the following subsections:

- FCFS: is an algorithm where the incoming tasks are scheduled based on First Come First Serve (FCFS) concept [17].
- Improved Min-Min algorithm: A single objective popular algorithm in existing task scheduling. The task with minimum completion time is selected and allocated to the corresponding VM through this algorithm [43].
- The cost-based priority algorithm: A single objective scheduling algorithm, with the drawback of: high priority tasks always get chance to be executed first, while the low priority task have to wait for a long time [44].
- MOTS: A multi-objective task-scheduling algorithm applied in allocating tasks to VMs to increase the throughput of the datacenter and decrease the execution time in cloud SaaS. This algorithm sorts tasks with non-dominated sorting method, and then allocates them to VMs sorted by MIPS in an ascending order and applies a fixed count of VMs during the experiments [17].
- SHARP: It concentrate prioritized jobs in a dynamic manner according to their and VM attributes. It allocates the jobs with an appropriate count of VMs in the cloud service providers subject to their requirements and system load for processing [45].

The newly proposed MODE approach is compared with the above mentioned approaches with respect to the characteristics tabulated in Table 4. In the simulation, values are obtained by running each algorithm for 20 times inspired by [46].

Table 4. Experiments configuration.

Experiment	VM count	Number of user tasks
1) Overall algorithms' makespan during variation in number of user tasks	Variable for MODE and SHARP approaches & Constant (20) for other approaches	Variable 100-1000
2) Overall algorithms' mean utilization during variation in number of user tasks	Variable for MODE and SHARP approaches & Constant (20) for other approaches	Variable 100-1000
3) Overall algorithms' deadline violation during variation in number of user tasks	Variable for MODE and SHARP approaches & Constant (20) for other approaches	Variable 100-1000
4) Overall algorithms' total cost during variation in number of user tasks	Variable for MODE and SHARP approaches & Constant (20) for other approaches	Variable 100-1000
5) Impact of variation in received request rate	Variable for MODE and SHARP approaches & Constant (20) for other approaches	Constant 1000
6) Impact of variation in penalty rate	Variable for MODE and SHARP approaches & Constant (20) for other approaches	Constant 1000
7) Comparison of VMs required in various approaches	Variable for MODE and SHARP approaches	Variable 100-1000

4.3.1. Makespan metric during user tasks count variation

To observe the makespan of the algorithm, the count of user tasks is within 100-1000, Fig. 3 and Table 5, where a direct relation is evident between the increase in makespan and the task count, due to an increase in task request per VM processing. When the user task count is high a significant difference is observed between different algorithm's makespant. For example, with 1000 user task, this MODE provides users with 57.31%, 55.43%, 30.32% lower makespan than FCFS, Cost-based priority and Improved Min-Min, respectively. MODE and SHARP outperform other algorithms, because they use a dynamic resource provisioning, while MODE outperforms SHARP, because MODE sends tasks with higher cost and size to the VMs with higher power.

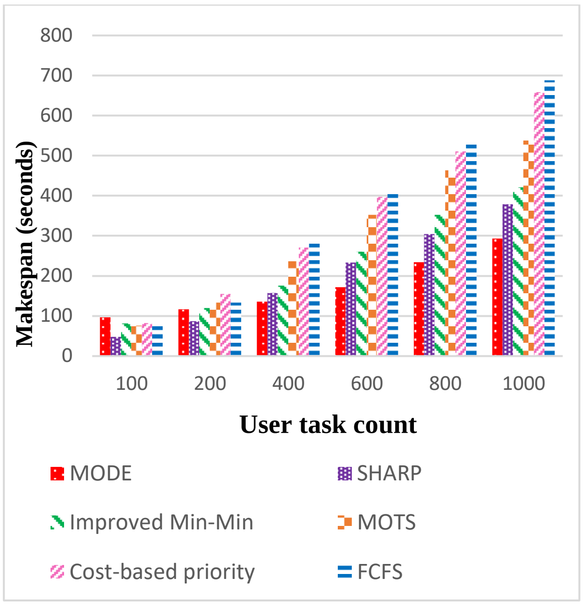

Fig. 3. Overall algorithms' makespan during variation in number of user tasks.

Table 5. Improvement percentages of MODE according to the makespan metric.

Algorithm	Number of Task					
	100	200	400	600	800	1000
SHARP	-	-	13/24	26/33	23/29	22/5
Improved Min-Min	-	3/21	22/83	34/09	33/86	30/32
MOTS	-	12/52	42/52	51/23	49/63	45/4
Cost-based priority	-	24/94	49/74	56/76	54/36	55/43
FCFS	-	12/56	51/88	58/03	56/17	57/31

4.3.2. Mean utilization metric during user tasks count variation

The mean utilization metric subject to different user tasks count, compared with the other five algorithms is bar charted in Fig. 4. The objective of SHARP is to meet user requirements, to stabilize the VMs count, and to promote resource utilization with the assistance of resource provisioning, with it no concern on the multi objective scheduling. In MOTS, the resource provisioning is not considered, because the objective is to satisfy the cloud user's objectives. As observed in Fig.4 and Table 6, this MODE in all cases has better utilization than the other algorithms. This MODE seeks to increase the revenue of the service provider by optimally VM utilizing and concentrates on data center utilization. Data center utilization is determined through the VMs count applied in processing the given task requests. This reason for improving the mean utilization in this proposed algorithm is due to resource provisioning and lease and release of resources required according to the cloud's elasticity in a manner that the surplus leased VMs are released when the load reaches zero, and the existence of additional VMs does not lead to a lower overall utilization.

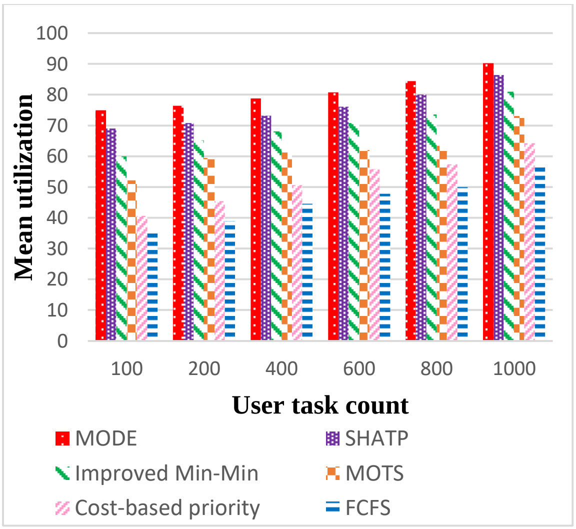

Fig. 4. Overall algorithms' mean utilization during variation in number of user tasks .

Table 6. Improvement percentages for MODE according to the mean utilization metric.

Algorithm	Number of Task					
	100	200	400	600	800	1000
SHARP	8	7/32	7/1	5/81	5/21	4/21
Improved Min-Min	20	14/65	13/57	12/37	12/79	10/19
MOTS	30/4	22/25	22/33	23/26	24/88	18/95
Cost-based priority	45/86	40/57	35/78	30/94	31/99	28/82
FCFS	53/2	48/5	43/4	40/84	40/75	36/69

4.3.3. The deadline violation metric during user tasks count variation

The results of deadline violation metric in relation to variation in task count are bar charted in Fig. 5. One of the reasons for evaluating this metric is to assure cost reduction. There exists a direct relation between cost and deadline violation. As observed in Fig. 5, the reason of this considerable difference between the MODE and SHARP group and other algorithms is due to their initiating new VMs upon receiving more user task. The count of tasks vary in relation to time. Unlike the available algorithms, which do not apply system load to allocate the resource, the MODE and SHARP do so. Here, it can be deduced that MODE outperforms SHARP, because it sends tasks with higher cost and size on the VMs with higher power. The improvements for deadline violation metric are tabulated in Table 7.

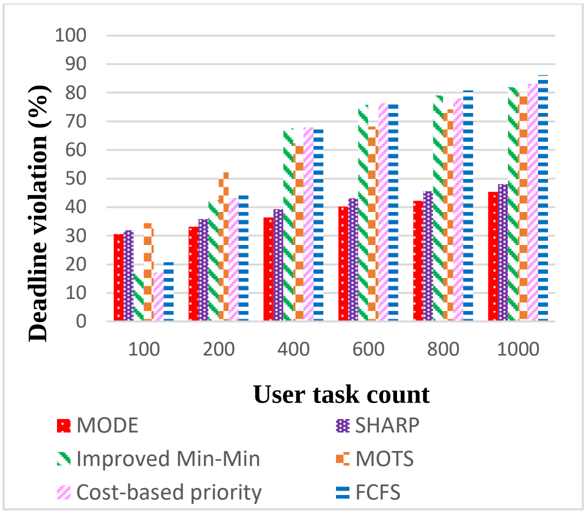

Fig. 5. Overall algorithms' deadline violation during variation in number of user tasks.

Table 7. Improvement percentages for MODE according to the deadline violation metric.

Algorithm	Number of Task					
	100	200	400	600	800	1000
SHARP	4/07	7/26	7/37	6/72	7/45	5/61
Improved Min-Min	-	21/69	46/15	46/96	46/58	44/56
MOTS	8/13	36/39	41/66	41/05	43/12	43/25
Cost-based priority	-	23/14	46/47	47/38	45/89	45/43
FCFS	-	26/87	46/31	47/51	47/77	47/33

4.3.4. The total cost during user tasks count variation

Comparisons of cost metric are bar charted in Fig. 6, where as observed an increase in tasks and VMs count, increase the cost. The advantages of MODE in cost metric, and the proportional improvement of cost values are much more steady than that of the other algorithm. This improvement is because in MODE, the cost metric at different stages is of concern. MODE and Cost-based priority consider the cost metric at sorting and allocating stages, consequently, both outperform other algorithms at large scale workloads. The details of these improvements for total cost values are tabulated in Table 8.

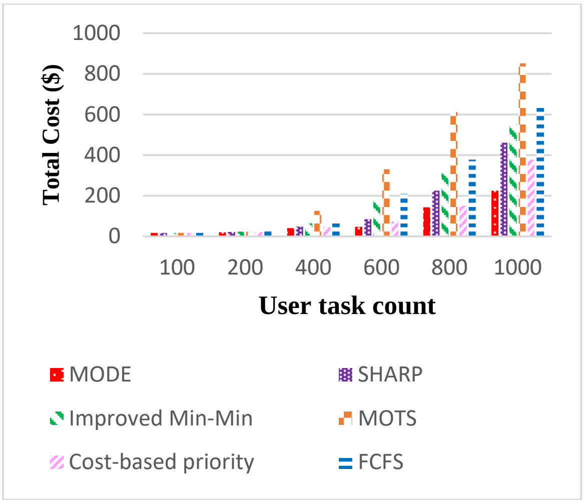

Fig. 6. Overall algorithms' total cost during variation in number of user tasks.

Table 8. Improvement percentages for MODE according to the total cost metric.

Algorithm	Number of Task					
	100	**200**	**400**	**600**	**800**	**1000**
SHARP	1/09	8/42	17/35	44/67	36/56	51/13
Improved Min-Min	2/82	14/33	36/69	73/17	54/64	59/20
MOTS	7/61	12/78	67/66	85/84	76/6	73/55
Cost-based priority	0/89	7/63	12/68	36/6	5/34	40/41
FCFS	5/75	26/6	50/6	77/62	62/06	64/57

4.3.5. The effect of variation on received user task rate

To observe this effect in MODE, the received task rate factor varies, while keeping all other factors constant. All experiments are run with 1000 user requests. As observed in Fig. 7, when the received task rate is high, a considerable effect is observed in the performance of SHARP, and MODE. Due to more received tasks per second, the overall trend of the makespan increases and service capability decreases due to applying fewer new VMs. As observed in Fig. 7, the MODE provides the smallest makespan and accepts more tasks count with less VMs count except when the received task rate is high. Even at high received task count, the difference between MODE and SHARP's makespan is about 20%. There exists a considerable increase in makespan when the received task rate is high due to high task requests accepted per VM, which in turn delay request processing. Here it can be deduced that even considering the makespan constraints on from users' part, the first choice for a SaaS provider is the MODE.

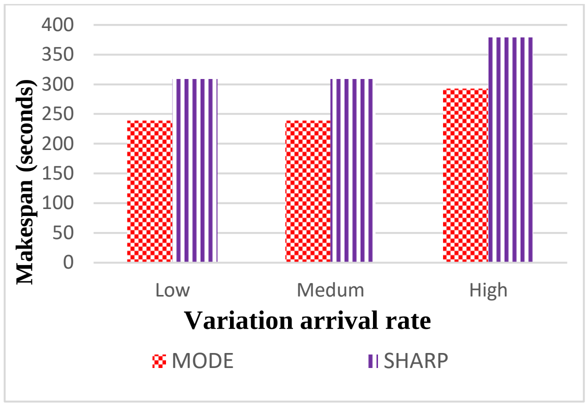

Fig. 7. Impact of arrival rate variation.

4.3.6. The effect of variation on the penalty rate

The penalty rate symbolized as (β), Eq. (14) depends on how long the user is willing to wait (r), here defined as penalty rate factor. Consequently, the larger the (r), the smaller the (β), Fig. 8. As observed in Fig. 8, the effect of variation in penalty rate is considered in MODE, because this is the only algorithm where the concept of penalty is of major concern in decreasing total cost. In this algorithm, the total cost is increased slightly and the average makespan decreases slightly when (r) changes from low to high, because when MODE accepts a few task requests with similar VMs count, the count of requests in each VM becomes smaller, leading to lower makespan for each task request.

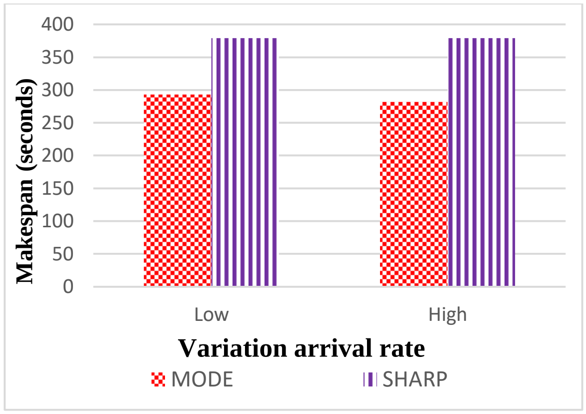

Fig. 8. Impact of arrival rate variation.

4.3.7. Comparison of VMs required in different approaches

The count of VMs required for processing the tasks is bar charted in Fig. 9. This section shows how the number of VMs utilized vary with respect to task requests to meet the QoS requirements of the users and complete them successfully. As observed, the count of

VMs applied in MODE approach remains approximately stable in relation to task requests count. Here, the average count of VMs applied in MODE is approximately 20% less than that of SHARP. The reason for this variation is that the MODE approach seeks to run tasks by applying unutilized VMs with the assistant of dynamic resource provisioning when lower count of tasks are received. When more count of tasks are received, the MODE maintains stability by allocating the least count of VMs according to the system load. During overloading, the servers applied by the SHARP approach are scaled up leading to an increase in the count of VMs up to 75. By comparing these two algorithms, it is revealed that MODE outperforms SHARP, because it sends tasks with higher cost and size to the VMs with higher power, and the remaining idle computing power is consumed to run the subsequent tasks requests.

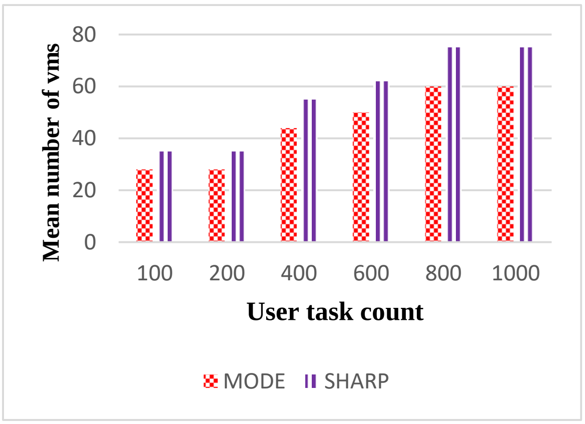

Fig. 9. Comparison of VMs required in various approaches

5. CONCLUSION AND FUTURE WORKS

Dynamic scheduling of many task requests constitutes one of the major aspects of Cloud computing. The task scheduling algorithms should concentrate on performing tasks and meeting the multiple quality of service requirements, which may have conflicting nature. Reducing makespan and deadline violation on a large count of tasks is difficult while costs are reduced. Optimizing these conflicting objectives and distribution of variable user tasks is difficult due to the unspecified execution time conditions, like elasticity. In this paper, the most important scheduling algorithms and existing methods are assessed, according to which, a multi-objective approach is proposed for dynamic task scheduling where the elasticity attribute of the cloud resource is applied. The received request rate for 100, 200, 400, 600, 800, 1000 users per second are applied and the tasks are allocated to each VM in the simulation. First through the received task requests of variable rates, the volume of received task load determines the count of VMs. Next, VMs are sorted by considering user requirements and tasks are sorted in a multi-objective manner, which will be allocated and executed with the capacity of VMs in a dynamic manner. Finally, if there is a delay in responding to a tasks' request, the provider will be penalized per time unit. During execution, according to the received task count, and applying the elasticity attribute of the cloud, if required, the new VMs will be leased and, if not required, they will be released. The evaluation results reveal that this MODE approach offers better scheduling in quality of service metrics like makespan, mean utilization of VM, total cost and deadline violation compared to SHARP, Improved Min-Min, MOTS, Cost-based priority and FCFS algorithms. In the future, attempt should be made to apply more quality of service metrics for multi-objective tasks in non-dominated sorting. In the resource allocation process, VMs can be checked before tasks allocation, in a sense that the maximum possible task space in the VM is occupied.

REFERENCES

[1] Lin W, Liang C, Wang JZ, Buyya R. "**Bandwidth-aware divisible task scheduling for cloud computing**". *Software: Practice and Experience*. Vol. 44, No. 2, pp. 63-74, 2014.

[2] Khorsand R, Ghobaei-Arani M, Ramezanpour M. "**FAHP approach for autonomic resource provisioning of multitier applications in cloud computing environments**". *Software: Practice and Experience.* Vol. 48, No. 12, pp. 2147-73, 2018.

[3] Ramezani F, Lu J, Taheri J, Hussain FK. "**Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments**." *World Wide Web*. Vol. 18, No. 6, pp. 1737-57, 2015.

[4] Zhou Z, Li F, Zhu H, Xie H, Abawajy JH, Chowdhury MU. "**An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments.**" *Neural Computing and Applications*. Vol. 8, pp. 1-1, 2019.

[5] Ghobaei-Arani M, Khorsand R, Ramezanpour M. "**An autonomous resource provisioning framework for massively multiplayer online games in cloud environment**." *Journal of Network and Computer Applications*. 2019 Jun 7.

[6] Zhan ZH, Liu XF, Gong YJ, Zhang J, Chung HS, Li Y. "**Cloud computing resource scheduling and a survey of its evolutionary approaches**." *ACM Computing Surveys (CSUR)*. Vol. 47, No. 4, pp. 63, 2015.

[7] Pantuza Júnior G. "**A multi-objective approach to the scheduling problem with workers allocation**." *Gestão & Produção*. Vol. 23, No. 1, pp. 32-45, 2016.

[8] de Campos CP, Benavoli A. "**Joint analysis of multiple algorithms and performance measures**." *New Generation Computing*. Vol. 35, No. 1, pp. 69-86, 2017.

[9] Kamesh SP, Priya S. "**Security enhancement of authenticated RFID generation**." *Int. J. Appl. Eng.*

Res. Vol. 9, No. 22, pp. 5968-74, 2014.

[10] Srichandan S, Kumar TA, Bibhudatta S. "**Task scheduling for cloud computing using multi-objective hybrid bacteria foraging algorithm**." *Future Computing and Informatics Journal*. Vol. 3, No. 2, pp. 210-30, 2018.

[11] Khorsand R, Safi-Esfahani F, Nematbakhsh N, Mohsenzade M. "**Taxonomy of workflow partitioning problems and methods in distributed environments**." *Journal of Systems and Software*. Vol. 132, pp. 253-71, 2017.

[12] Torabi S, Safi-Esfahani F. "**A dynamic task scheduling framework based on chicken swarm and improved raven roosting optimization methods in cloud computing**." *The Journal of Supercomputing*. Vol. 74, No. 6, pp. 2581-626, 2018.

[13] Tang L, Pan JS, Hu Y, Ren P, Tian Y, Zhao H. "**A Novel Load Balance Algorithm for Cloud Computing**." *In International Conference on Genetic and Evolutionary Computing*, pp. 21-30, Springer, Cham, 2015.

[14] Babu KR, Samuel P. "**Enhanced bee colony algorithm for efficient load balancing and scheduling in cloud.**" *In Innovations in bio-inspired computing and applications*, pp. 67-78, 2016.

[15] Patel G, Mehta R, Bhoi U. "**Enhanced load balanced min-min algorithm for static meta task scheduling in cloud computing".** *Procedia Computer Science*. Vol. 57, pp. 545-53, 2015.

[16] Ali HG, Saroit IA, Kotb AM. "**Grouped tasks scheduling algorithm based on QoS in cloud computing network**." *Egyptian informatics journal*. Vol. 18, No. 1, pp. 11-9, 2017.

[17] Lakra AV, Yadav DK. "**Multi-objective tasks scheduling algorithm for cloud computing throughput optimization**." *Procedia Computer Science*. Vol. 48, pp. 107-13, 2015.

[18] Aslanpour MS, Ghobaei-Arani M, Toosi AN. "**Auto-scaling web applications in clouds: a cost-aware approach".** *Journal of Network and Computer Applications*. Vol. 95, pp. 26-41, 2017.

[19] Gabi D, Ismail AS, Zainal A, Zakaria Z, Al-Khasawneh A. "**Hybrid cat swarm optimization and simulated annealing for dynamic task scheduling on cloud computing environment**." *Journal of ICT*. Vol. 17, No. 3, pp. 435-67, 2018.

[20] Hu B, Sun X, Li Y, Sun H. "**An improved adaptive genetic algorithm in cloud computing**." *In2012 13th International Conference on Parallel and Distributed Computing*, Applications and Technologies 2012 Dec 14 (pp. 294-297). IEEE.

[21] Ghanbari S, Othman M. "**A priority based job scheduling algorithm in cloud computing**." *Procedia Engineering*. Vol. 50, No. 0, pp. 778-85, 2012.

[22] Bhoi U, Ramanuj PN. "**Enhanced max-min task scheduling algorithm in cloud computing"**. *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*. Vol. 2, No. 4, pp. 259-64, 2013.

[23] Kaleeswaran A, Ramasamy V, Vivekanandan P. "**Host Scheduling Algorithm U sing Genetic Algorithm"** *In Cloud Computing Environment. International Journal of Advances in Engineering & Technology*. 2013 Jan.

[24] Panda SK, Gupta I, Jana PK. "**Task scheduling algorithms for multi-cloud systems: allocation-aware approach**." *Information Systems Frontiers*. Vol. 21, No. 2, pp. 241-59, 2019.

[25] Islam S, Lee K, Fekete A, Liu A. "**How a consumer can measure elasticity for cloud platforms**." *In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering* 2012 Apr 22 (pp. 85-96). ACM.

[26] Shawky DM, Ali AF. "**Defining a measure of cloud computing elasticity."** *In2012 1st International conference on systems and computer science (ICSCS)* 2012 Aug 29 (pp. 1-5). IEEE.

[27] Beltrán M. BECloud: "**A new approach to analyse elasticity enablers of cloud services**." *Future Generation Computer Systems*. Vol. 64, pp. 39-49, 2016.

[28] Ghobaei-Arani M, Jabbehdari S, Pourmina MA. "**An autonomic approach for resource provisioning of cloud services**." *Cluster Computing*. Vol. 19, No. 3, pp. 1017-36, 2016.

[29] Bansal N, Maurya A, Kumar T, Singh M, Bansal S. "**Cost performance of QoS Driven task scheduling in cloud computing**." *Procedia Computer Science*. Vol. 57, pp. 126-30, 2015.

[30] Banerjee S, Adhikari M, Kar S, Biswas U. "**Development and analysis of a new cloudlet allocation strategy for QoS improvement in cloud**." *Arabian Journal for Science and Engineering*. Vol. 40, No. 5, pp. 1409-25, 2015.

[31] Gawali MB, Shinde SK. "**Task scheduling and resource allocation in cloud computing using a heuristic approach**." *Journal of Cloud Computing*. Vol. 7, No. 1, pp. 4, 2018.

[32] Zhang L, Zhang Y, Jamshidi P, Xu L, Pahl C. "**Service workload patterns for Qos-driven cloud resource management**." *Journal of Cloud Computing*. Vol. 4, No. 1, pp. 23, 2015.

[33] Sun Y, Lin F, Xu H. "**Multi-objective optimization of resource scheduling in Fog computing using an improved NSGA-II**." *Wireless Personal Communications*. Vol. 102, No. 2, pp. 1369-85, 2018.

[34] Bikas MA, Alourani A, Grechanik M. "**How elasticity property plays an important role in the cloud: a survey**." *In Advances in Computers, Elsevier,* Vol. 103, pp. 1-30, 2016.

[35] Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. "**CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms**." *Software: Practice and experience*. Vol. 41(1), pp. 23-50, 2011.

[36] Kamalinasab S, Safi-Esfahani F, Shahbazi M. "**CRFF. GP: cloud runtime formulation framework based on genetic programming**." *The Journal of Supercomputing*. pp. 1-35, 2019.

[37] Vecchiola C, Chu X, Mattess M, Buyya R. "**Aneka—integration of private and public clouds."** *Cloud Computing Principles and Paradigms*. Hoboken, NJ,

USA: Wiley. pp. 251-74, 2011.

[38] Wu L, Garg SK, Versteeg S, Buyya R. "**SLA-based resource provisioning for hosted software-as-a-service applications in cloud computing environments**." *IEEE Transactions on services computing*. Vol. 7, No. 3, pp. 465-85, 2013.

[39] Khorsand R, Safi-Esfahani F, Nematbakhsh N, Mohsenzade M. "**ATSDS: adaptive two-stage deadline-constrained workflow scheduling considering run-time circumstances in cloud computing environments**." *The Journal of Supercomputing*. Vol. 73, No. 6, pp. 2430-55, 2017.

[40] Ma L, Lu Y, Zhang F, Sun S. "**Dynamic task scheduling in cloud computing based on greedy strategy**." *In International Conference on Trustworthy Computing and Services, Springer, Berlin, Heidelberg*, pp. 156-162, 2012.

[41] Safari M, Khorsand R. "**PL-DVFS: combining Power-aware List-based scheduling algorithm with DVFS technique for real-time tasks in Cloud Computing**." *The Journal of Supercomputing*. Vol. 74, No. 10, pp. 5578-600, 2018.

[42] Safari M, Khorsand R. "**Energy-aware scheduling algorithm for time-constrained workflow tasks in DVFS-enabled cloud environment**." *Simulation Modelling Practice and Theory*. Vol. 87, pp. 311-26, 2018.

[43] Wang G, Yu HC. "**Task scheduling algorithm based on improved Min-Min algorithm in cloud computing environment.**" *InApplied Mechanics and Materials, Trans Tech Publications,* Vol. 303, pp. 2429-2432, 2013.

[44] Salot P. "**A survey of various scheduling algorithm in cloud computing environment**." *International Journal of Research in Engineering and Technology*. Vol. 2, No. 2, pp. 131-5, 2013.

[45] Komarasamy D, Muthuswamy V. "**ScHeduling of jobs and Adaptive Resource Provisioning (SHARP) approach in cloud computing**." *Cluster Computing*. Vol. 21, No. 1, pp. 163-76, 2018.

[46] Hemasian-Etefagh F, Safi-Esfahani F. "**Dynamic scheduling applying new population grouping of whales meta-heuristic in cloud computing**." The Journal of Supercomputing. pp. 1-65, 2019.