

Efficient Task Scheduling Multi-Objective Particle Swarm Optimization in Cloud Computing

Entisar S. Alkayal
Faculty of Computing and
Information Technology
King Abdul Aziz University
Jeddah, Saudi Arabia
ealkayal@kau.edu.sa

Nicholas R. Jennings
Computing, Electrical, and
Electronic Engineering
Imperial College
n.jennings@imperial.ac.uk

Maysoon F. Abulkhair
Faculty of Computing and
Information Technology
King Abdul Aziz University
Jeddah, Saudi Arabia
mabualkhair@kau.edu.sa

Abstract—Task scheduling in data centers is a complex task due to their evolution in size, complexity, and performance. At the same time, customers' requirements have become more sophisticated in terms of execution time and throughput. Against this background, this work presents a new model of resource allocation that optimizes task scheduling using a multi-objective optimization (MOO) and particle swarm optimization (PSO) algorithm. In more detail, we develop a novel multi-objective PSO (MOPSO) algorithm, based on a new ranking strategy. The main insight of this algorithm is that the tasks are scheduled to the virtual machines to minimize waiting time and maximize system throughput. The algorithm leads to a reduction in execution time of 20%, a reduction the waiting time of 30%, and shows improvements of up to 40% in throughput compared to the current state of the art.

Keywords—Multi-objective Particle swarm optimization; cloud computing; task scheduling; ranking strategy

I. INTRODUCTION

Cloud computing is a parallel and distributed system containing a number of dynamically interconnected virtualized machines [5]. These virtual machines are allocated over a network to provide services to consumers (such as servers, storage and applications) according to their needs. Consumers then just pay for the resources they consume [4]. In a cloud environment, the allocation of virtual and/or physical resources is a key part of the management system, since its efficiency directly influences the performance and the cost of the entire system. Thus, an inefficient resource allocation has a direct negative effect on performance and cost. Given this, the main goal of resource allocation in such settings is to make the best use of the infrastructure resources and combine them to achieve higher throughput to solve large-scale computation problems [24]. In such environments, computing resources are allocated when a customer sends a request with their requirements.

In more detail, cloud computing data centers depend on virtualization, a technique that allows a computational resource to be partitioned into multiple independent execution environments, which are called Virtual Machines (VMs) [2]. Virtualization provides the flexibility to configure a number of virtual machines on the same physical machine. Moreover, multiple VMs can be initiated and terminated on a single host dynamically depending on the task

requirement. The number of CPU cores in each host determines the numbers of VMs that can be mapped to it. That is because each VM is executed on one core of the CPU. In most cases, the tasks are not mapped directly to the physical machine; instead, they are first assigned to virtual machines. Therefore, the task scheduling process is performed on two consecutive levels (see Figure 1). In the first, tasks are assigned to the appropriate virtual machine, while in the second the virtual machines are mapped to appropriate physical machines. In this work, we aim to improve resource allocation on the VM level through the development of a new task-scheduling algorithm. The selection of the VMs is based on the tasks' requirements such as cost and CPU power, and then the task scheduler assigns tasks to the selected VMs.

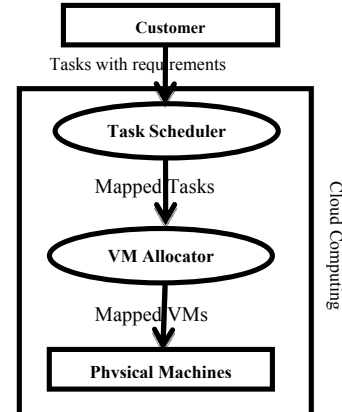


Figure 1: Resource Allocation Layers in Cloud Computing

There are multiple objectives that needed to be satisfied in cloud systems including (performance, profit, and utilization). To address the problems of dealing with multiple objectives, a number of researchers have developed techniques for multi-objective optimization (MOO) [20]. Specifically, MOO studies the search methods that are used to find solutions based on several conflicting objectives such as performance in terms of minimizing waiting time and maximizing resource utilization or maximizing profitability.

To date, many techniques have been developed to achieve MOO. In this work, we focus on Swarm Intelligence (SI) because of its proven benefits in flexibly responding to rapid changes in the system and rapidly adapting to the external

environment during run-time [7]. In addition, SI algorithms consist of a number of simple behavior rules that lead to simplicity in solving MOO problems [8]. Cloud management systems also need to be adaptive to the changes in the environment. Again, SI algorithms have a good technique record at achieving this.

In more detail, SI algorithms are used in complex systems because there is no need to have a central control structure and the system can be elastic and flexible in terms of adding or removing resources without influencing the overall structure. Several intelligent techniques for improving scheduling in cloud systems have been developed, including genetic algorithms, bee colony algorithms and ant colony algorithms [10]. Specifically, a recent advance in this area is Particle Swarm Optimization (PSO) and this has been used in a wide range of complex applications including combinatorial optimization problems such as finding optimal routes, scheduling, structural optimization, image analysis, data mining, bioinformatics, finance and business [12]. PSO is popular in such applications because of its simplicity and speed in finding solutions. Thus, we use it as the base for this work.

More specifically, Reyes-Sierra and Coello [20] developed the Multi-Objective Particle Swarm Optimization (MOPSO) algorithm. This is based merging the principles of PSO and MOO. The new features of MOPSO are optimizing task scheduling and allocating resources. MOPSO has been chosen as the basic of this work because we have multi-objectives in the task scheduling process and MOPSO is suitable to solve them. Specifically, MOPSO combines the advantages of the PSO algorithm and MOO that have been shown to improve the process of finding solutions fast compared with other evolutionary algorithms [23]. In particular, we use the MOPSO algorithm to optimize task scheduling, to satisfy the goals of improving performance in terms of waiting time and throughput. In dealing with multi- objectives we can either combine all of them to a single one or consider each function separately. In this work, we evaluate the multiple objectives separately based on a ranking strategy to select optimal solution. We make this decision because we need a method that considers the three objectives that are equal in the priority and weight.

In this work, we define a methodology to optimize task scheduling based on the MOPSO algorithm. The scheduler schedules tasks over VMs and decides which is the best VM for task execution. Here we use the MOPSO algorithm to select the best VM depending on the three conflicting objectives: Expected Completed Time (ECT), the Task Execution Cost (TEC) and the VM processing. These objectives conflict in that if we reduce the ECT it leads to increase the TEC. In addition, to minimize the ECT we should to maximize the VM processing which requires maximizing the cost. The MOPSO algorithm evaluates each objective separately and uses the ranking strategy to find the best VM.

The novelty of this work is to devise the best solution using a new ranking strategy based on three mentioned objectives to speed up the process of finding the best VM. Specifically, our ranking dealing with the three objectives in a balance way, which no other strategy can handle.

The research is organized as follows: section II: details the background and the related work. Section III: details of our MOPSO algorithm. Section IV: presents the experimental procedures, testing methodologies, and evaluation of the algorithm. Section V: concludes and highlights future work.

II. BACKGROUND AND RELATED WORK

PSO consists of a swarm of particles; each of which represents a solution of the problem. In this work, the particles are represented as a set of VMs that are allocated for the tasks. Every particle in the swarm behavior has two main characters: a position x that notates the suggested location and a velocity v that means the speed of moving. The best solution for one particle is called the best personal experience (pbest), while the best solution among all particles in the population is called the best group experience (gbest) solution. The position of a particle at any instant in time is influenced by its personal best position (pbest) and the position of the other best particles in the global problem space (gbest).

In more detail, PSO uses adaptive movement, which shifts a particle's position in each iteration. The way in which PSO updates particle xi at generation t is shown in equation 1 (Kennedy and Eberhart [12]):

$$xi(t) = xi(t-1) + vi(t) \quad (1)$$

where:

$xi(t)$ is the current position of particle i at iteration t

$xi(t-1)$ is the position of particle i at iteration $t-1$

$vi(t)$ is the velocity of particle i at iteration t

The velocity of particle i at time t is given in equation 2 (Kennedy and Eberhart [12]):

$$vi(t) = w \times vi(t-1) + r1 \times C1 \times (pbesti - xi) + r2 \times C2 \times (gbesti - xi(t)) \quad (2)$$

where:

$xi(t)$ is the current position of particle i at iteration t

$vi(t)$ is the velocity of particle i at iteration t

$vi(t-1)$ is the velocity of particle i at iteration $t-1$

$pbesti$ is the best position of particle i

$gbesti$ is the position of the best value for particle i in a population

w is the inertia weight

$r1, r2$ are random numbers in the range $[0, 1]$

$C1, C2$ are the acceleration coefficients.

In equation 2, w is used to control the movement of the particle and improve the convergence of PSO. The numbers $r1$ and $r2$ are used to provide randomness in the movement of particles inside the swarm in the range of $[0-1]$ to control the movement of the particles in the search space. Both $C1$ and $C2$ are positive constants, which are specific parameters to control the effect of the personal and global best particles. Their values are set to fixed equal values because the particle is influenced by its own best position (pbest) and the best position of its neighbors (gbest) [20]. In this work, we use a range values for w starting from 1.0 then decreasing by 0.1 until we reach 0.0. Values of $C1$ and $C2$ are fixed since particles should only be affected with the best position only. In our algorithm, we specified $C1=C2=2.05$ in order to provide the most promising results and search behavior as advocated by Reyes-Sierra and Coello [20].

The main general steps of the PSO algorithm are summarized in Algorithm 1 [8]. These steps are the same in most variant of PSO, but they differ in the process of dealing with the objective function to select the best value for particles. The algorithm starts by initializing the swarm, it updates the position of each particle, and then it evaluates each particle's position according to the objective function (equation 2). If a particle's current position is better than its previous best position, then it is updated. Next, it determines the best particle that depends on the particle's previous best positions. Specifically, Algorithm 1 formulates the main steps of the PSO algorithm based on one objective. It starts by initializing the swarm, velocity, position, and pbest values. The main loop from lines 3 to 9 iterates through all particles to find the best solutions. The COMPUTEFITNESS function is invoked to compute the fitness function for each particle based on the specific objective. Then the EVALUATE procedure is used to compare the fitness value with the personal best for each particle and return the best value (pbest). Procedure UPDATE (in line 7) is invoked to update the velocity and position in each iteration based on equations 1 and 2. Finally, the pbest with new best value for each particle is returned.

Algorithm 1: Pseudo code of a general PSO algorithm.

```

1.  Procedure PSO (
2.    INITIALIZE (Swarm , Velocity , Position, Pbest)
3.    while ( stop criteria not satisfied ) do
4.      for p ∈ P do           // iterate through all particles
5.        f=COMPUTEFITNESS(Swarm(p)); //compute fitness
6.        Pbest=EVALUATE(f)
7.        UPDATE(Velocity, Position) //using eq 1 ,2
8.      end for
9.    end while
10.   return Pbest
11. end Procedure

```

The main goal of a single objective optimization based on PSO is to find the “best” solution, which corresponds to the minimum or maximum value of a single objective function. This type of optimization is useful as a tool to provide decision makers with insights into the nature of the problem, but it cannot provide a set of alternative solutions that trade different objectives against each other. On the other hand, in a multi-objective optimization with conflicting objectives, there is no single optimal solution. The interaction among different objectives gives rise to a set of compromise solutions, known as non-dominated or Pareto-optimal solutions [23]. In more detail, the Pareto optimal set consists of multiple trade-off solutions with a wide range of values for the objectives that are non-dominated best positions (often-called leaders) that are used to guide the particles. In particular, a solution belongs to the Pareto optimal set if there is no other solution that can improve at least one of the objectives without degrading any other objective. In multi-objective optimization problems, we can distinguish two fundamental approaches that are used to deal with multi-objective PSO algorithms [20]:

- The **first approach** consists of algorithms that consider each objective function separately. In this approach, each particle is evaluated for one objective function at a time, and the determination of the best position is performed similarly to the single-objective optimization case. It combines multiple objectives using the weighted sum of them in one single objective. This is a simple approach and it is widely used when the objectives do not conflict. However, the complexity in this approach is in determining the weight values for each objective depending on the importance of the objective. Different weight values produce widely varying results.
- The **second approach** consists of algorithms that evaluate all objective functions for each particle and is based on the concept of Pareto optimality. In the context of MOO, Pareto dominance is used to compare and rank decision solutions to determine the set of solutions (leaders). The determination of the leaders is not straightforward, since there can be many non-dominated solutions in the neighborhood of a particle. However, only one solution from the leaders is selected to update the velocity. However, the Pareto dominance becomes a complex process especially when there is more than two objectives and if the objectives conflict with each other. Thus, in our work, we do not use the Pareto-optimal method. Instead, we established a new ranking strategy for the objectives to find the optimal solution (detailed in section 3).

In more detail, several researchers use the PSO algorithm to preserve one objective: for example, Suresh, Santhosh, and Karthik [22] used PSO and Fuzzy technique to balance the load over resources to handle one objective that maximizes utilization. Netjinda, Sirinaovakul and Achalakul [16] used PSO to verify cost optimization. Wang, Zhang and Si [14] improved the outcome of the cloud provider by using PSO to allocate an optimal resource for execution tasks. Pacini, Mateos and Garcia [17] to schedule the VMs over available physical machines developed a cloud scheduler based on PSO, as well as reducing the number of created VMs.

PSO based on two objectives for task scheduling was first developed in Beegom and Rajasree [3]. They focused on improving two factors: the overall execution time for all tasks and the cost of cloud resources. Subsequently, a number of researchers have extended this line of work. Specifically, Adamuthe, Pandharpatte and Thampi [1] studied multi-objective optimization with a PSO algorithm. These objectives were to maximize profit and minimize wasted resources. However, they treated multi- objectives the same as single objective by assigning the same weight to them then calculating the sum of their weight. Thus, it is not appropriate to apply this method in our work since we do not want all the objectives to be equally treated because we have conflicted objectives.

In Wang et al. [13], an Improved Particle Swarm Optimization (IPSO) was used based on multi-objective optimization to satisfy customer requirements. A resource allocation method was developed to improve the makespan, execution cost and failure rate based on PSO [15]. However, this work is also based on the weighted sum objectives, which combines all objectives into one single objective. This is not suitable for our work because while it gives good results with two objectives, it is not effective for more than two; since it deals with the three objectives as one objective [18]. In addition, there are other researches of multi-objectives that used the Pareto set approach (e.g. Tripathi, Bandyopadhyay and Pal [23]). In Feng, Wang, Zhang, and Li [9], PSO was used to find resources based on multiple factors including total execution time and the quality of service (QoS) of a task. They used Pareto-dominance to search in a multi-objective algorithm. Additionally, they used a Binary objective PSO algorithm based on two objectives: cost and QoS. Finally, Ramezani, Lu and Hussain [19] presented PSO based on two objectives makespan and cost. They used multi-objective optimization with a Pareto optimal method with two objectives but in our approach, we use the ranking of three objectives to select the solution in leaders and not use the Pareto set because it includes more numbers of comparisons. However, this approach consumes more time in computing the best values because it based on comparisons to finding the dominant solutions. The number of comparisons is increased with the increasing number of objectives, which leads to take more time in dealing with three objectives compared to two objectives (Zitzler and Thiele [25]). Thus, Pareto set takes long time in finding the solutions.

III. OUR MOPSO TASK SCHEDULING ALGORITHM

Our task-scheduling algorithm over available VMs is described in this section. In terms of task scheduling, tasks are prioritized depending on their length in terms of their numbers of million instructions (MI). In this work, the tasks are defined based on the required number of processing elements and a utilization model that states the task's execution rate through defining the current requested processing elements.

In more detail, Algorithm 2 describes our task-scheduling algorithm based on MOPSO algorithm. The algorithm is consist of the main processes of PSO (as in Algorithm 1) but instead of using one objective, here we use multiple objectives. The algorithm starts with a loop that calculates the ECT and TEC for each task in all available VMs using CALCULATEECT and CALCULATETEC functions. These two functions apply equations 3 and 7, as will be discussed in the following section. Then, the steps from lines 8 to 16 are the main functions that represents MOPSO algorithm that we are used in scheduling tasks to VMs.

In our algorithm we improve the MOPSO by adding the COMPUTEFITNESS function (as shown in Algorithm 3), which includes the process of computing the fitness function of the three objectives, based on ranking strategy. Specifically

the rank value for each particle (i.e.VM) is done by computing the value of each objective, and then the smallest value of the rank value for each particle is selected. Thus, the particle with the smallest rank among all the corresponding three objective values is selected as the best solution (i.e. Pbest variable in the Algorithm 2).

Algorithm 2 : Task scheduling algorithm.

```

1. Procedure TASKSCHEDULING (T, VMs)
2.   for t ∈ T do                                // iterate all tasks
3.     for v ∈ VMs do
4.       TEC(t,v) ← CALCULATETEC(t,v)           // cost of each task
5.       ECT(t,v) ← CALCULATEECT(t,v)           // task execution time
6.     end for
7.   end for
8.   INITIALIZE (Swarm , Velocity , Position, Pbest)
9.   for i ∈ T do
10.    for v ∈ VMs do                                // iterate through all particles
11.      f ← COMPUTEFITNESS((VMs, ECT, TEC))
12.      Pbest (i) = EVALUATE(f)
13.      UPDATE (Velocity(i), Position(i))         // using eq 1 ,2
14.    end for
15.  end for
16.  return Pbest
17. end Procedure

```

Specifically, in our algorithm, we developed a new method based on ranking objectives instead of using the concept of Pareto set. Methods based on the Pareto set are inappropriate for our setting because they involve many comparisons to find the dominant set of solutions. Subsequently they take a long time. Our ranking method overcomes this shortcoming by simplifying the way of finding the best solution, which is based on sorting the lists only (Algorithm 3). We rank the solutions based on the objective values in the MOPSO algorithm to select the suitable VM for each task. Furthermore, this work does not use the dominance methods to find Pareto set solution to control the leaders archive. Instead, we devise a new method to evaluate the objective functions based on ranked particles to find the optimal solution.

For more detail, our strategy of ranking the objectives involves evaluating the three objectives separately to find the best solution. The first objective function is to minimize the task execution cost (TEC) (as per equation 3); the second is to minimize the ECT, which is presented in equation 7; the third is the VM processing, which is represented by the VM CPU speed. The particles (which represent VMs in this work) best fitness values are calculated according to the three mentioned factors (i.e. the least TEC and ECT, and highest VM processing speed). Here the TEC is the sum of the task processing cost and the Input Output transfer cost, which is the cost of transferring the tasks data files.

$$TEC(i, j) = TPC(i, j) + IOCost(i) \quad (3)$$

where:

TEC is a task execution cost of task i on a VM j

i is the id of task in task list

j is the id of VM in VM list

TPC (i, j) is cost of executing task i in a VM j

IOCost (i) is the cost of transferring the data files of the task i

The task processing cost (TPC) is computed depending on the time of processing the task on VM and the price of the VM. The price of VM varies based on the type of the VM. Here the VM type is characterized depending on the CPU power and storage size. In this work, we have four types of VMs. In particular, the task processing cost is calculated as shown in equation 4.

$$TPC(i, j) = PT(i, j) \times VMP(j) \quad (4)$$

where:

$PT(i, j)$ is the processing time for task i in a VM j

$VMP(j)$ is the price of VM j

The processing time (PT) is calculated by dividing the task length measured in MI by the VM processing measured in million instruction per second (MIPS) as shown in equation 5.

$$PT(i, j) = TL(i) / PSV(j) \quad (5)$$

where:

$TL(i)$ is the length of task i in MI

$PSV(j)$ is the processing speed of VM j in MIPS

The input output transfer cost (IOCost) for each task is computed depending on the size of the task's input and output files and the cost of bandwidth, as shown in equation 6.

$$IOCost(i) = FS(i) \times CB \quad (6)$$

where:

$FS(i)$ is the size of the task i input and output file in MI

CB is the cost of the bandwidth

The ECT is computed for each task in each VM, represented in a $i \times j$ matrix where i denotes tasks and j represents VMs. For each task, the ECT at each VM is computed by considering the following parameters: the task length, measured in Million Instructions (MI) and the VM processing speed in MIPS. The ECT for any task is calculated as the shown in equation 5.

The new contribution of this work is the method that is used to evaluate the objective functions based on ranking each objective to select the best solution in MOPSO algorithm. In MOPSO algorithm, the COMPUTEFITNESS function is invoked to evaluate the objectives in MOPSO. In this function, we applied the ranking strategy to evaluate the solutions in each swarm iteration. Each objective is represented by a two-dimensional matrix. For more detail about our proposed ranking, Algorithm 3 illustrates the main steps for computing fitness value based on ranking the solution according to the three objective functions to improve the process of MOPSO algorithm.

The procedure COMPUTEFITNESS as shown in the Algorithm 3 starts by sorting VMs based on ECT, TEC, and VM processing speed and storing each result in separate matrix (lines 3 to 5). The loop from lines 6 to 8 computes the rank for each task in each VM by accumulating the rank of the three matrices. Then the rank is sorted in an ascending order. Finally, the rank is set for the task based on the first element in each matrix, which includes the smallest value. To illustrate

this approach, suppose that we have eight tasks with five VMs and we compute the three objectives for each solution as in equations 3 and 6. The results are shown in Tables 1, 2, and 3.

Algorithm 3: COMPUTEFITNESS procedure

```

1. Procedure COMPUTEFITNESS (VMs, ECT, TEC)
2.   INITIALIZE (Rank)
3.   V1 ← SORT(VMs, ECT) // sort ascending based on ECT
4.   V2 ← SORT(VMs, TEC) // sort ascending based on TEC
5.   V3 ← SORT(VMs) // sort descending based on VM speed
6.   for  $v \in$  VMs do // iterate through all VMs
7.     Rank(v) ← V1(v) + V2(v) + V3(v)
8.   end for
9.   SORT(Rank) // sort r ascending based on rank
10.  return Rank(0) // return smallest of particle in 0 index
11. end Procedure

```

In more detail, for ECT and TEC, we rank the virtual machines based on the least values by sorting the virtual machines in an ascending order. For the VM processing, we sort the virtual machines in a descending order because our aim is to maximize this objective. Then we calculate the rank of the three objectives by summing the ranks for each objective to select the virtual machine with lowest rank value for each task as shown in Table 4. All shaded cells in the Tables (1-4) represent the best solution, i.e. the best VM for each task. In our example, the best VM for task 1 (i.e. T1) is the virtual machine number 1 (i.e. V1), because its rank is the smallest value (5) (which is $2+1+2=5$) compared with other virtual machines. For task 2 (T2) the smallest rank is eight using virtual machines number 3 and 4 (since their rank is equal). Our strategy in the case if there are many virtual machine with equal smallest value is to get the first one in the sequence, so the best virtual machine for task 2 (T2) is V3. The same way for task 4 (T4) the best virtual machine is number one (V1).

Table 1: Task Execution Cost (TEC)

The values of TEC						The rank value after sorting					
Task	V1	V2	V3	V4	V5	Task	V1	V2	V3	V4	V5
T1	10	12	8	15	12	T1	2	4	1	5	3
T2	15	11	13	12	10	T2	5	2	4	3	1
T3	12	14	10	10	8	T3	4	5	2	3	1
T4	14	18	16	22	19	T4	1	3	2	5	4
T5	18	12	19	17	15	T5	4	1	5	3	2
T6	18	16	19	15	14	T6	4	3	5	2	1
T7	12	18	22	20	19	T7	1	2	5	4	3
T8	24	20	18	14	17	T8	5	4	3	1	2

Table 2: Expected Completion Time (ECT)

The values of ECT						The rank value after sorting					
Task	V1	V2	V3	V4	V5	Task	V1	V2	V3	V4	V5
T1	21	24	30	22	25	T1	1	3	5	2	4
T2	24	21	19	25	27	T2	3	2	1	4	5
T3	18	21	21	26	16	T3	2	3	4	5	1
T4	25	20	24	19	21	T4	5	2	4	1	3
T5	30	25	23	22	26	T5	5	3	2	1	4
T6	32	30	21	25	28	T6	5	4	1	2	3
T7	19	17	18	22	23	T7	3	1	2	4	5
T8	12	19	21	17	20	T8	1	3	5	2	4

Table 3: VM Processing

The Values of VM processing						The rank value after sorting					
Task	V1	V2	V3	V4	V5	Task	V1	V2	V3	V4	V5
T1	20	14	12	21	15	T1	2	4	5	1	3
T2	21	15	20	24	18	T2	2	5	3	1	4
T3	27	18	24	30	25	T3	2	5	4	1	3
T4	27	23	21	24	30	T4	2	4	5	3	1
T5	29	23	22	28	26	T5	1	4	5	2	3
T6	32	26	25	23	22	T6	1	2	3	4	5
T7	35	28	21	24	20	T7	1	2	4	3	5
T8	21	28	25	23	25	T8	5	1	2	4	3

Table 4: VMs Rank Values

Task	V1	V2	V3	V4	V5	Best
T1	2+1+2=5	4+3+4=11	1+5+5=11	5+2+1=8	3+4+3=10	V1
T2	5+3+2=10	2+2+5=9	4+1+3=8	3+4+1=8	1+5+4=10	V3
T3	4+2+2=8	5+3+5=13	2+4+4=10	3+5+1=9	1+1+3=5	V5
T4	1+5+2=8	3+2+4=9	2+4+5=11	3+5+1=9	4+3+1=8	V1
T5	4+5+1=10	1+3+4=8	5+2+5=12	3+1+2=5	2+4+3=9	V4
T6	4+5+1=10	3+4+2=9	5+1+3=9	2+2+4=8	1+3+5=9	V4
T7	1+3+5=9	2+1+2=5	5+2+4=11	4+4+3=11	3+5+5=13	V2
T8	5+1+5=11	4+3+1=8	3+5+2=10	1+2+4=7	2+4+3=9	V4

IV. MOPSO IMPLEMENTATION AND EVALUATION

Designing and testing applications in a real cloud-computing environment can be expensive, which is one of the reasons why researchers have developed cloud simulations [21]. In addition, using simulation provides flexibility in defining configurations and is easy to use for both utilization and customization. In this work, we use CloudSim because it is an open source package so it is available for public use. In addition, it includes different modules that simulate cloud environments and it provides the flexibility in customizing the simulation by adding and modifying modules depending on the desired design. Specifically, we implemented our model using CloudSim 3.0.3 (which is the most up to date version of CloudSim as of April 2014).

In more detail, CloudSim is a general and extensible simulation framework that allows the modeling, simulation, and experimentation of cloud computing infrastructures and services [6]. It supports several functionalities, such as the queuing and processing of events, the creation of cloud system entities (services, hosts, data centers, brokers, and virtual machines), communication between components, and management of the simulation clock [6]. More specifically, our simulation model involved one data center and many customers. We created virtual machines to provide cloud services provided by the data center, and we simulated many cloud tasks by using data from the data set file.

To evaluate our model, we created a number of experiments to measure if the model satisfies the objectives discussed in section one. First, we tested our system functions by managing tasks over available VMs and mapping them to physical machines in the data center. Second, we performed experiments to compare the execution of tasks in our model with the weighted sum. These algorithms are the most common used ones in task scheduling and allocation in cloud systems as we discussed in section two. In more detail, to evaluate our model, we used four types of host machines (e.g.

Amazon's Elastic Compute Cloud EC2's instances: high-CPU, extra, small, and micro), with the following characteristics for each host based on EC2 specification respectively:

- CPU MIPS: {2000, 2500, 3000, 3500} // MIPS
- RAM Size: {4096, 4096, 4096, 4096} // MB
- Storage Size: 1000000; // 1GB
- Bandwidth = 1000000 // 1Gb/s
- Number of Cores in each host: 2

We created the VMs with this specification to fit in the hosts with the previous characteristics. We had four types of VMs that varied in CPU MIPS and Memory size, which led to differences in price for each of them. The VMs are created depending on the task's requirements. The specific characteristics for each VM are as follows:

- CPU MIPS: { 2500, 2000, 1000, 500 } // MIPS
- RAM Size : { 870, 1740, 1740, 613 } // MB
- Bandwidth: 10000 // 100 Mbit/s
- Storage Size: 10000; // 10 MB
- VM Price: {0.200, 0.148, 0.047, 0.020} // \$
- Number of Cores in each VM: 1

To simulate the task we used the Large Hadron Collider Computing Grid (LCG) data set [11]. This describes the work of 11 days of activity from multiple nodes that make up the LCG. It includes nearly 200.000 jobs, and each line in the file contains information about the completed job, with some information such as the submission time, customer ID, compute element name, and job runtime. We use this data set because it includes task information relative to what we need in our simulation. The workloads that were not included in our model were tightly coupled tasks and complex workflow applications. The tasks are simulated to be submitted to our system, which includes 1000 hosts with random types based on the predefined specification. In our evaluation, we use numbers of VMs ranging from (500- 2000), each host can run up to two VMs because each host had two cores. The numbers of VMs that are created in each experiment depend on the numbers and requirement of tasks. Each experiment was run ten times, and then the average is computed in such that in each experiment we use different host types because we randomly created them. We submitted different numbers of tasks ranging from 500 to 4000 since there was no significant change shown in the results with more than 4000 tasks.

After running the simulation, we measured the following parameters:

- **Average Waiting Time:** This is defined as the ratio of the sum of waiting times of all tasks to the total number of tasks waiting execution. The average waiting time is measured by computing the difference between the submission time of the task to the system and the starting time of the execution of all tasks in the experiment as shown in equation 7. The waiting time includes the time taken for mapping tasks to VMs.

$$WT = \frac{\sum_{i=1}^m (start\ time(t_i) - submit\ time(t_i))}{m} \quad (7)$$

where m denotes the number of tasks running in the system at a given this instant.

- **Average Execution time:** This is defined as the ratio of the sum of the execution times of all tasks to the total number of executed tasks. The average task execution time is measured by computing the difference between starting task execution and ending task execution for each task in the experiment as shown in equation 8.

$$ET = (\sum_{i=1}^m \text{end execution time}(i) - \text{start time}(i)) / m \quad (8)$$

where m denotes the number of tasks running in the system at a given unit of simulation time.

- **Throughput:** The throughput measures the overall performance of the system. It indicates the number of tasks that the model can executed in a specific time as shown in equation 9.

$$\text{Throughput} = \sum_{i=1}^m \text{completed tasks} / T \quad (9)$$

where T denotes the simulation time and m is the total number of tasks that finish execution.

In this section, we illustrate the effectiveness of our method in dealing with multi-objectives as three objectives compared with treating them as one weighted sum of objectives. In addition, we compared the results of our algorithm with using Pareto set method. In this work, the weighted sum is computed by multiplying each objective with the weight that represents its importance in the system. The values of objectives that we want to maximize are positive and the other that we should to minimize are negative. In this experiment, we give all objectives the same weight as equation 10 such as follow:

$$\text{Rank} = \sum (\text{objective value}(k) \times \text{weight}(k)) \quad (10)$$

where:

k represents the objective value, and $k=1, 2, 3$

The weight is a nonnegative integer such that $\sum \text{weight}(k) = 1$ where $k=1, 2, 3$.

For simplicity, we use equal weights for each objective because all objectives have the same importance in our model. In this evaluation, we focused on the results of performance factors (i.e. execution time, waiting time, and throughput). The results of execution time, waiting time, and throughput are presented in Figures 2, 3, and 4).



Figure 2: Execution Time (data shown with 95% confidence).

In more detail, as shown in Figure 2, the execution time in our model is reduced by more than 25% compared to the weighted sum approach and by 15% compared to Pareto set.

In respect to waiting time, our model reduces the waiting time by about 25% compared to weighted sum as shown in Figure 3. The waiting time using Pareto set is the highest because it consumes more time for finding solutions.

The throughput in our model also has an improvement of about 20% compared with weighted sum and by 40% compared with Pareto set as shown in Figure 4. This improvement occurs because the waiting time is minimized by improving the performance in the system, so many tasks are executing.

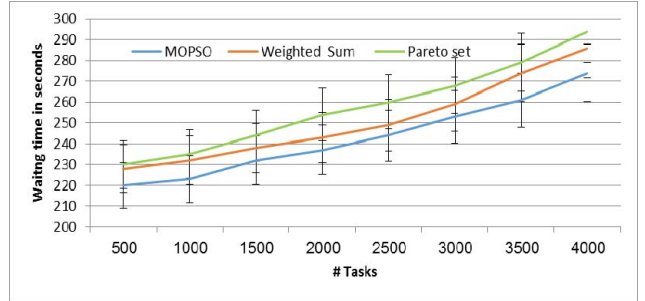


Figure 3: Waiting Time (data shown with 95% confidence)

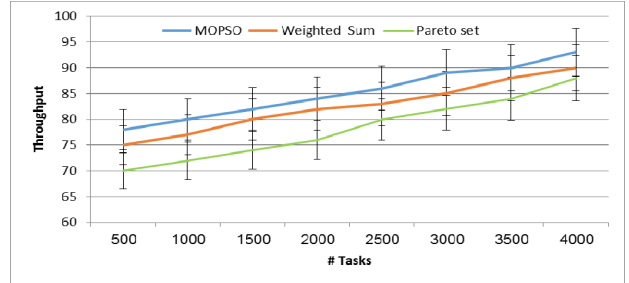


Figure 4: Throughput Results (data shown with 95% confidence).

V. CONCLUSIONS

We proposed a model for allocating resources in cloud computing to improve task scheduling. We improved the task scheduling method by using a multi-objective PSO algorithm based on ranking strategy. In doing so, this work advanced the state of the art by improving task scheduling by using MOPSO based on three objectives. Our work devises a new ranking strategy to evaluate the three objective functions: ECT, TEC, and VM processing. The results of the ranking strategy are used in MOPSO task scheduling to find the optimal VM for each task. This improvement speeds up and simplifies the process of evaluating objectives in MOPSO algorithms instead of using Pareto set method or weighted sum. Specifically, from these improvements, our work increased the performance in terms of waiting time and throughput by improving the MOPSO algorithm. Our MOPSO task-scheduling algorithm leads to an increased throughput and reduced waiting time. In addition to showing an improvement for up to 20% in execution time comparing to the weighted sum objective.

REFERENCES

- [1] Adamuthe, A. C., Pandharpatte, R. M., & Thampi, G. T. (2013, November). Multiobjective Virtual Machine Placement in Cloud Environment. In *Cloud & Ubiquitous Computing & Emerging Technologies (CUBE), 2013 International Conference on* (pp. 8-13). IEEE.
- [2] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., & Warfield, A. (2003). Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5), 164-177.
- [3] Beegom, A. A., & Rajasree, M. S. (2014). A Particle Swarm Optimization Based Pareto Optimal Task Scheduling in Cloud Computing. In *Advances in Swarm Intelligence* (pp. 79-86). Springer International Publishing.
- [4] Buyya, R., Beloglazov, A., & Abawajy, J. (2010). Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308*.
- [5] Buyya, R., Vecchiola, C., & Selvi, S. T. (2013). *Mastering cloud computing: foundations and applications programming*. Newnes.
- [6] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1), 23-50.
- [7] Dorigo, M., & Birattari, M. (2010). Ant colony optimization. In *Encyclopedia of machine learning* (pp. 36-39). Springer US.
- [8] Durillo, J. J., García-Nieto, J., Nebro, A. J., Coello, C. A. C., Luna, F., & Alba, E. (2009, January). Multi-objective particle swarm optimizers: An experimental comparison. In *Evolutionary Multi-Criterion Optimization* (pp. 495-509). Springer Berlin Heidelberg.
- [9] Feng, M., Wang, X., Zhang, Y., & Li, J. (2012, October). Multi-objective particle swarm optimization for resource allocation in cloud computing. In *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on* (Vol. 3, pp. 1161-1165). IEEE.
- [10] Guo, L., Shao, G., & Zhao, S. (2012, September). Multi-objective task assignment in cloud computing by particle swarm optimization. In *Wireless Communications, Networking and Mobile Computing (WiCOM), 2012 8th International Conference on* (pp. 1-4). IEEE.
- [11] Li, H., Muskulus, M., & Wolters, L. (2007, January). Modeling job arrivals in a data-intensive grid. In *Job Scheduling Strategies for Parallel Processing* (pp. 210-231). Springer Berlin Heidelberg.
- [12] Kennedy, J., and Eberhart, R. (1995). Particle swarm optimization. In *Fourth IEEE International Conference on Neural Networks*, pages 1942-1948.
- [13] Wang, Y., Wang, J., Wang, C., & Song, X. (2013). Research on resource scheduling of cloud based on improved particle swarm optimization algorithm. In *Advances in Brain Inspired Cognitive Systems* (pp. 118-125). Springer Berlin Heidelberg.
- [14] Wang, Z., Zhang, J., & Si, J. (2014, July). Application of particle swarm optimization with stochastic inertia weight strategy to resources scheduling and assignment problem in cloud manufacturing environment. In *Control Conference (CCC), 2014 33rd Chinese* (pp. 7567-7572). IEEE.
- [15] Moorthy, R. S., Somasundaram, T. S., & Govindarajan, K. (2014, September). Failure-aware resource provisioning mechanism in cloud infrastructure. In *Global Humanitarian Technology Conference-South Asia Satellite (GHTC-SAS), 2014 IEEE* (pp. 255-260). IEEE.
- [16] Netjinda, N., Sirinaovakul, B., & Achalakul, T. (2012, May). Cost optimization in cloud provisioning using particle swarm optimization. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2012 9th International Conference on* (pp. 1-4). IEEE.
- [17] Pacini, E., Mateos, C., & García Garino, C. (2014). Dynamic scheduling based on particle swarm optimization for cloud-based scientific experiments. *CLEI Electronic Journal*, 17(1), 3-3.
- [18] Parsopoulos, K. E., & Vrahatis, M. N. (2002). Recent approaches to global optimization problems through particle swarm optimization. *Natural computing*, 1(2-3), 235-306.
- [19] Ramezani, F., Lu, J., & Hussain, F. (2013). Task scheduling optimization in cloud computing applying multi-objective particle swarm optimization. In *Service-Oriented Computing* (pp. 237-251). Springer Berlin Heidelberg.
- [20] Reyes-Sierra, M., & Coello, C. C. (2006). Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, 2(3), 287-308.
- [21] Sakellari, G., & Loukas, G. (2013). A survey of mathematical models, simulation approaches and testbeds used for research in cloud computing. *Simulation Modelling Practice and Theory*, 39, 92-103.
- [22] Suresh, M., Santhosh, K. B., & Karthik, S. (2014, March). A Load Balancing Model in Public Cloud Using ANFIS and GSO. In *Intelligent Computing Applications (ICICA), 2014 International Conference on* (pp. 85-89). IEEE.
- [23] Tripathi, P. K., Bandyopadhyay, S., & Pal, S. K. (2007). Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients. *Information Sciences*, 177(22), 5033-5049.
- [24] Vinothina, V., Sridaran, R. and Ganapathi, P. (2012) "A survey on resource allocation strategies in cloud computing." *International Journal of Advanced Computer Science and Applications* 3.6: 97-104.
- [25] Zitzler, E., & Thiele, L. (1999). "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach". *evolutionary computation, IEEE transactions on*, 3(4), 257-271.