Enhanced Particle Swarm Optimisation and Multi Objective Optimization for the Orchestration of Edge Cloud Clusters

Hafiz Faheem Shahid and Claus Pahl
Free University of Bozen-Bolzano, Bolzano, Italy

Keywords: Swarm Intelligence, Particle Swarm Optimization, Distributed Systems, Load Balancing, Edge Cloud Cluster.

Abstract: Load balancing and workload distribution cause challenges for the management of IoT and distributed systems in the edge computing environment. Swarm intelligence is a technology suitable for the management of distributed systems, networks, communication and routing protocols. Swarm intelligence-based PSO algorithms (particle swarm optimization) can be applied for load balancing and task scheduling in cloud computing environments operating through a broker agent. In distributed cloud environments, data is collected and then processed at the center of the cloud, rather than making decision at edge nodes closer to IoT infrastructures. Here, we develop an automated orchestration technique for clustered cloud architectures. An Autonomous Particle Swarm Optimization, called the A-PSO algorithm, is implemented that enables an edge node, such as a remote storage, to work as part of a decentralized, self-adaptive intelligent task scheduling and load balancing agant between resources in distributed systems. Using Multi Objective Optimization (MOO), complementing the A-PSO algorithm, we also include metrics such as Actual Round-Trip Time (ARTT) of tasks assignments to the remote storage to reduce the execution cost. Our A-PSO algorithm can orchestrate the distribution of large volumes of data to remote storage and back in cluster, i.e., coordinated distributed cloud environments.

1 INTRODUCTION

Task scheduling and allocation of tasks to cloud resources is a load balancing problem that needs to be optimized in cloud environments. Resource management and load balancing are an important part of any cloud computing environment in a distributed context. With the advancement of hardware and software application, a remaining problem is the optimization of distributing (i.e., storing and fetching) data in cloud clusters (1; 1; 1). The problem we are discussing here is to balance the load between the remote data storages as a sample type of a cloud resource. We can often experience that with limited resources of remote data storage in edge cloud applications, sometimes applications show performance degradations (1) resulting from may uncertainties in the environment (1). This can happen when the number of transactions increases dramatically, possibly causing downtime.

In recent years, swarm intelligence algorithms have been widely used, specifically using swarm intelligence techniques for the development of distributed systems, networks, communications and routing protocols. The two basic swarm intelligence algorithms, that are used for the development of collective behavior of applications and software are Stochastic Diffusion Search (SDS) and Particle Swarm Optimization (PSO). SDS is often seen as representing the collective behavior of social insects (such as ants) and PSO the collective behavior of social animals (birds and fish) (2). In SDS and PSO algorithms, social interaction and exchange of information is the significant key role that is practiced by the social animal and social insects in swarm intelligence. A basic strategy behind these two algorithms of SDS and PSO is to work in a swarm to achieve the target instead of working individually (3).

The aim of this paper is to explore the suitability of bio-inspired algorithms for autonomous cloud resource management. The solution shall be based on applying a swarm intelligence approach to coordinate workloads between cloud resource clusters and to manage them in dispersed settings. Swarm intelligence as a form of distributed intelligence is about implementing algorithms for distributed systems management using Particle Swarm Optimization (PSO) or Ant Colony Optimization (ACO), and adapt them to schedule task fairly to the cloud resources (storage in our case) and balance the load correspondingly between them (16). Cloud technology is moving to-

Shahid, H. and Pahl, C.
Enhanced Particle Swarm Optimisation and Multi Objective Optimization for the Orchestration of Edge Cloud Clusters.
DOI: 10.5220/0008019201550162
In *Proceedings of the 11th International Joint Conference on Computational Intelligence (IJCCI 2019)*, pages 155-162
ISBN: 978-989-758-384-1

wards multi-cloud environments with the inclusion of various devices, data storages and sensors at the edges (0). This requires the integration of data center technologies with much more constrained devices, but still using virtualized solutions to deal with scalability, flexibility and multi-tenancy concerns.

We aim at an automated orchestration technique for edge cloud PaaS (Platform as a Service) architectures based on workloads that are deployed in several local edge clusters. For edge clouds, application and service orchestration can help to manage and orchestrate applications through containers (17). In this way, computation can be brought to the edge through an orchestration technique, rather than transferring large volumes of data from the edge to the cloud.

The paper is organised as follows. We start with background and related work in Section 2. Section 3 introduces our extended PSO solution in the context of related work. Section 4 describes the experimental evaluation. In Section 5, we discuss the results, before concluding in Section 6.

2 BACKGROUND

With the proliferation of internet-capable devices, workload distribution in cloud computing is becoming major problem. Here, we propose a method of load balancing at storage resource level based on Particle Swarm Optimization (PSO) algorithm. In many existing load balancing solutions, a broker node plays an important role for task scheduling to the resources. In our proposed autonomous PSO algorithm, there is no central broker node to assign and executes to the virtual machines. The data computation is held in between the edge nodes (resource hosts) instead of sending information for decisions to the central broker or controller node of the cloud computing environment.

We focus on Particle Swarm Optimization (PSO) for the development of our orchestration solution. PSO is inspired by the social behavior of birds flocking and fish schooling (4), (5). A particle in PSO reflects to a bird or fish that moves from one position to another to reach its best position to find for instance food and a safe place. The movement of the particle is determined by the velocity, which has both direction and magnitude. At any instance of time the position of the particle is influenced by its best position (pbest) and the position of the best particle in entire swarm (gbest). To calculate the performance of each particle in the swarm, we must apply an objective function. The velocity of flocking birds is continuously changing and thus the positions of birds in swarm will also need to change continuously (6), (7), (8), (9), (10).

In order to provide some background, we introduce two basic equations of PSO algorithms that update the velocity and position of a swarm particle:

$$v_i^{k+1} = wv_i^k + C1r1x(pbest_i^k - x_i^k) + C2r2x(gbest - x_i^k) \quad (1)$$

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

PSO algorithm (1) describes the updated velocity of the particle and equation (2) describes the updated position of that particle (18). The parameters of the equations are summarised in Table 1.

Table 1: PSO – Parameters.

$pbest_i^k$	best position of particle i
$gbest$	position of best particle in swarm
v_i^k	current velocity of particle i at k iteration
v_i^{k+1}	velocity of particle i at k+1 iteration
x_i^k	current position of particle i at k iteration
x_i^{k+1}	position of particle i at k+1 iteration
w	inertia
c1,c2	acceleration coefficient factor
r1,r2	random number between 0 to 1

3 PSO+MOO LOAD BALANCING

Our overall solution combines Particle Swarm Optimization (PSO) and Multi Objective Optimization (MOO), which we will introduce in sequence.

3.1 Load Balancing with PSO

We now map the PSO algorithm to our problem of load balancing in edge clouds as follows:

- we choose the cloud storage resources (RES) as particles;
- we define the velocity v_i^k of particle i at iteration k as change of load below; and
- we interpret the direction d as the number of storage resources connected.

We consider in our adapted PSO algorithm the current load CL_i^k on the RES_i at iteration k for our load balancing problem. The load on the RES_i will update to CL_i^{k+1} at iteration $k+1$, after every iteration of incoming tasks, see Equation (3). The objective function in Equation (4) calculates the fitness value of each RES to get Global best $(gbest)$ and Personal best $(pbest)$ values of the RESs. In load balancing, we use the objective function min $f_{min}(x)$ to obtain the best minimum fitness value of the resource.

$$CL_i^{k+1} = 1(pbest) + (CL_i^k – pbest) \quad (3)$$

$$f_{min}(x_i^{k+1}) = (CL_i^{k+1}/MaxStorage_i) \times d \quad (4)$$

After each Iteration, the load on the resources RES is updated by Equation (3). Incoming tasks to the next iteration could be different in size and varying in their numbers. We can set the limit of incoming tasks between any two numbers. After updating the load on resources, each RES resource needs to change its position in a queue for accepting the incoming load. In Equation (4), CL_i^{k+1} is the current load on the cloud storage resource i in any iteration $k+1$. $MaxStorage_i$ is the maximum storage capacity or upper limit of storing data at the same iteration, where d denotes the direction of the particle.

We interpret the direction d as the number of storage resources connected under control of the PSO algorithm. Thus, for our load balancing, we use the $f_{min}(x_i^{k+1})$ objective function, because we need to find the best resource (data storage) with the minimum memory load in the entire swarm of resources RES (19). The cloud storage resource that has the minimum fitness value in the cluster of RESs is the global best position of RES in the cloud.

In order to illustrate this, we use an example of three cloud resources that provide 3 different storage services with a maximum memory capacity of 100, 150, or 200 MBs in them. After 1, 2 and 3 iterations, we get the current load on each RES as follows:

Cloud Storage Resource 1 (RES_1):

1. CLi^{k+1}: current load on RES_1 = 25 MBs
2. $MaxStorage_i$: max capacity on RES_1 = 100 MBs

which we add to the objective function (4) as follows:

$$f_{min}(RES\_1) = (CL_i^{k+1}/MaxStorage_i)\ d$$
$$f_{min}(RES\_1) = (25/100)\ 3$$

with d as the number of resources in the RES swarm:

$$f_{mi}(RES\_1) = 0.75$$

Cloud Storage Resource 2 (RES_2):

1. CL_i^{k+1}: current load on RES_2 = 47 MBs
2. $MaxStorage_i$: max capacity on RES_2 = 150 MBs

which we add to the objective function (4) as follows:

$$f_{min}(RES\_2) = (CL_i^{k+1}/MaxStorage_i)\ d$$
$$f_{min}(RES\_2) = (47/150)\ 3$$

with d as above and:

$$f_{min}(RES\_2) = 0.94$$

Cloud Storage Resource 3 (RES_3):

1. CL_i^{k+1}: current load on RES_3 = 39 MBs
2. $MaxStorage_i$: max capacity on RES_3 = 200 MBs

which we add to the objective function (4) as follows:

$$f_{min}(RES\_3) = (CL_i^{k+1}/MaxStorage_i)\ d$$
$$f_{min}(RES\_3) = (51/200)\ 3$$

with d as above and:

$$f_{min}(RES\_3) = 0.76$$

After this, we have obtained the personal best values for the 3 resources, which are 0.75, 0.94 and 0.76 respectively. The global best for the entire swarm is 0.75, which corresponds to resource 1.

After each iteration of incoming load on resources, the objective function (4) calculates *pbest* values for each RES. As we are using the objective minimum function $f_{min}(x_i^{k+1})$ that reflects the resource with minimum load on it among all resources. This is said to be the *gbest* resource of the swarm.

Table 2: A-PSO Load Balancing Algorithm – Parameters.

$MaxStorage_i$:	total storage capacity on RESi
Dimension: d	number of RESs
$pbest_i^k$	personal best position of RESi at iteration k
$gbest_i^k$	global best position of RESi at iteration k
CL_i^k	current load on RESi at iteration k
	(initial velocity of particle i in PSO)
CL_i^{k+1}	updated load on RESi at iteration $k+1$
	(updated velocity of particle i in PSO)
x_i^k	current position of RESi at iteration k
x_i^{k+1}	updated position of RESi at iteration $k+1$

After updating the load on RES_i, the position of the resource at iteration k, x_i^k will also change. The *gbest* RES then moves forward in the priority queue and lead the rest of the RESs by means of balancing load. The positions of the RESs in the cloud will be updated by the load of incoming tasks at iteration $k+1$. Then x_i^{k+1} is the position of RES_i at iteration $k+1$.

When a resource reaches a threshold load of 60% or higher, then it shall be made temporarily unavailable for further tasks. In that case, the tasks in the incoming task queue will need to be dealt with by other resources in the swarm that are still in a workable state. A backup resource shall also be provided in case all primary resources have reached their threshold load of 60%.

3.2 PSO Load Balancing with MOO

As an enhancement of the A-PSO algorithm, we combine our load balancing PSO solution with multi objective optimization. When we have a higher number of iterations for the incoming load for resources, we naturally observe that task execution takes longer. For the management of distributed systems, our A-PSO algorithms does distribute load equally to resources and balances the load between edge nodes correspondingly, but on the other hand its cost for task

assignment cycles per seconds becomes higher. The challenges we are facing is to minimize the execution cost along with balancing load for the distributed management in the edge cloud environment. To overcome the problems, we decided to apply Multi Objective Optimization (MOO) (13) in conjunction with the PSO solution. For MOO we aim to minimize the execution cost in task scheduling along with the A-PSO algorithm to balance the load. Time costing is based on the execution of tasks as cycles per millisecond. At every iteration with new tasks, the execution of tasks at the resource can be calculated in milliseconds. A single cycle is assumed to consist here of 100 milliseconds as a default, i.e., 200 milliseconds to execute a storage request corresponds to 2 cycles.

For each iteration we know the execution cost for a swarm resource. For the next iteration, we can reduce the execution cost by assigning the task to the virtual machine that takes the minimum number of cycles to execute. We prioritize resource requests dealt with by A-PSO additionally with respect to the minimum number of cycles they take during every iteration.

Table 3: Variables for Time Costing in Load Balancing.

N:	Number of tasks
M:	Number of virtual machines
EET :	Expected Execution Time
ETT :	Expected Transmission Time
$ARTT$:	Actual Round-Trip Time
$Load_j$:	Load on RES_j
X_{ij} :	Decision to allocate task i to RES_j or not [0,1]

In order to reduce the number of cycles of task execution at the resources, we adopt the time costing definitions (5), (6) and (7) (11).

$$Min_{cost} = \sum^{n}.\sum^{m}.EET_{ij} * x_{ij} \tag{5}$$

$$Min_{cost} = \sum^{n}.\sum^{m}.ETT_{ij} * x_{ij} \tag{6}$$

$$ARTT_{ij} = EET_{ij} + ETT_{ij} \tag{7}$$

In the following, we consider cost with respect to the Expected Execution Time (EET), Expected Transmission Time (ETR) and Expected Round-Trip Time $(ERTT)$ factors.

- **Expected Execution Time** (EET) is defined as EET_{ij} of $Task_i$ on resource RES_j denoting the time of a task from initialization to the eventual termination at the execution point (11). Our goal is to minimize the EET to reduce the time costing in the load balancing process (12).
- **Expected Transmission Time** (ETT)**:** ETT_{ij} of $Task_i$ on resource RES_j is the time from the assignment of the task to the resource to the complete transmission of the actual task (6).
- **Actual Round-Trip Time** $(ARTT)$**:** we can calculate the $ARTT_{ij}$ of $Task_i$ on resource RES_j by equation (7). We sum up the time of execution EET_{ij} with the transmission time ETT_{ij} to minimize the cost of load balancing (7) (13). Dividing the $ARTT$ by 100, we get the number of execution cycles per milliseconds. The number cycles that each task management consumes indicates the cost of the load balancing.

Task scheduling and load balancing is calculated with respect to the execution cycle of tasks in milliseconds:

$$Execution\ Cycle = ARTT_{ij}/100$$

which defines the Actual Round-Trip Time of $tasks_i$ to RES_j, i.e., $ARTT_{ij}$.

The prioritization of the resources based on the time consumption considers:

- the current load on the storage resource (minimum value of fitness by object function Equation (4)),
- that the resource takes minimum cycle per milliseconds to execute tasks based on Equation (7).

We set each iteration weight for an automated solution of PSO for task scheduling and load balancing from 0 to 10. 0 is the minimum number and 10 is the maximum weight of each iteration. If the weight of the iteration is between 0-7 (random value generated at each iteration between 0-10), then the system assigns tasks to storage resources with respect to the fitness value of the resources. A global best $RES(gbest)$ will be one with the minimum $f.v$ following equation (4). If the iteration weight is between 8-10, then the solution assigns the incoming task to the resource that takes the minimum number of cycles to execute them. In this case, the global best $RES(gbest)$ is the one that costs the minimum number of cycles to execute.

Below, we present the enhanced A-PSO algorithm for task scheduling and load balancing.

A-PSO Load Balancing and Scheduling

1. For all tasks (ti) ∈ T do
   Determine incoming tasks
   Determine size of each task
2. End for
3. For all resources (RESi) ∈ RES do
   Determine current load on RESi by Eq. (4)
   //Velocity of particle i in PSO algorithm
   Determine real load on RESi
   //Max storage of RES
4. End for
5. Initialize RES

6. While (current load on RESi $< 60\%$) stop condition
7. For all resources (RESi) do
   Calculate fitness value of each RES by using objective function, Eq. (4)
   If
   (current $f.vi > pbest_i$)
   $pbest_i = current f.v_i$
   End if
   If
   ($pbest_i < \forall$ RESs $personalbest$)
   $gbest = pbest_i$
   End if
8. End for
9. For all resources RES do
   Calculate execution time cost (7) in msec
   Calculate task assignment to RES in msec
10. End for
11. Prioritize all tasks according to size
12. Prioritize all RESs with respect to minimum $f.v(4)$ || Prioritize all RES according to min task exec cycles/msec
13. For all Prioritized tasks $(t_i) \in$ T do
    For all Prioritized resources (RESj) $\in$ RES do
    If (iteration weight between 0-7 && current load in $RES_j <= 60\%$)
    assign task(t_i) to resource (RES_j)
    remove task from sorted tasks queue
    current load RES_j++
    End if
    Else If (iteration weight between 8-10 & current load in $RES_j <= 60\%$)
    assign task (t_i) to resource (RES_j)
    remove task from sorted task queue
    current load RES_j++
    End if
    Else
    Break // (assign next task (t_i) in queue to next (RES_j))
    End for
14. End for
15. Update current load on RESs // Equation (3)
16. Update position of RES // objective function (4)
17. End for

We distinguish non-prioritized requests of incoming tasks with different sizes that will receive a cloud resource storage position after updating the overall load, and prioritized ones. The latter are incoming tasks sorted with respect size in descending order for execution. Also, resources are sorted wrt. their position, considering minimum fitness value in descending order for receiving incoming task load and also minimum cycles to execute incoming tasks.

4 IMPLEMENTATION

In order to implement an autonomous solution for a decentralised architecture, we need to avoid an often used central broker node from the system that assigns the task to remote storage. For instance, many tools for the experimentation of load balancing and task scheduling exist, like CloudSim (7), (11), (14) as a simulator. CloudSim for instance works with a central broker node that has complete information of resources as well as complete information of the tasks. We implemented our autonomous A-PSO algorithm in Java that works without centralised control.

In a sample experiment, we illustrate that how our A-PSO algorithm works. For this scenario, we assume three storage resources with different capacity sizes. The tasks are also different in size. Three input tasks are assigned to resources.

In the 1*st* iteration, we start with the following sample task queue (Table 4):

Table 4: 1*st* Iteration – unsorted task queue.

Tasks	T1	T2	T3
Size	9	6	13

The algorithm groups the incoming tasks (this is the task information phase of general load balancing systems) according to storage size. The task with the maximum size will execute first (Table 5):

Table 5: 1*st* Iteration – sorted task queue.

Tasks	T1	T2	T3
Size	6	9	13

Now the solution assigns these tasks to the resources in a way that they should balance the load using the PSO algorithm that calculates the gbest value of the resources using the objective function in Equation (4). In a group of prioritized resources, the gbest resource with the least load is moved to the front of the task allocation process. The personal best positions of the resources x_i^k are then calculated. Afterwards, we move the tasks to resources, i.e., map from Table 5 to Table 6, which is illustrated in Fig. 1.

Table 6: 1*st* Iteration – initial position of storage resources.

Resource (RES)	Current Load on RES
c	undefined
b	undefined
a	undefined

After updating the load on the resources (velocity of particles in PSO) CL_i^{+1} at iteration step $k+1$, the resources change their positions x_i^{k+1} accordingly. Here, the objective function will calculate the fitness value of each resource to find the $pbest_i^{k+1}$ and $gbest_i^{k+1}$ at iteration step $k+1$.

As discussed before, the task with the maximum size will be assigned to the resource positioned at the

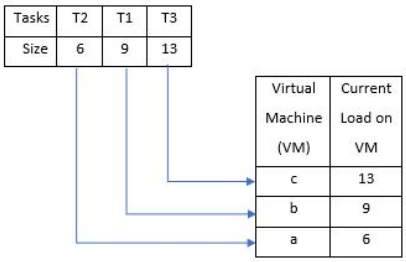

Figure 1: 1st Iteration mapping of tasks to resources.

Table 7: 1*st* Iteration – updated positions of resources.

Resource (RES)	Current Load on RES
a	6
b	9
c	13

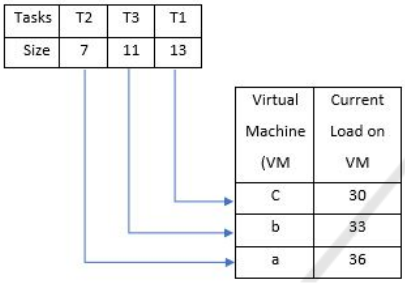

Figure 2: 4th Iteration mapping of tasks to resources.

top of the queue with the least load on it. Table 7 shows the updated positions of the resources x_i^{k+1}. After the 2*nd* and 3*rd* iteration, the updated load on the resources is shown in Table 8.

Table 8: 3*rd* Iteration – updated load/position of resources.

Resource (RES)	Current Load on RES
c	30
b	33
a	36

We continue this experiment with a fourth iteration with a different size of input, but using the same method as in iteration 1, see Tables 9 and 10.

Table 9: 4*th* Iteration – unsorted task queue.

Tasks	T1	T2	T3
Size	13	7	11

Table 10: 4*th* Iteration – sorted task queue.

Tasks	T1	T2	T3
Size	7	11	13

The sorted task requests are then as shown in Table 10. Assigning tasks to resources is a mapping

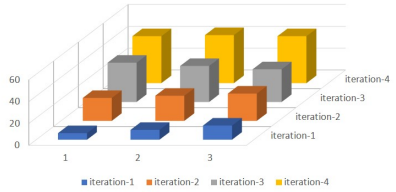

Figure 3: Results – for each of the resources 1, 2 and 3 and each of the 4 iterations, allocated task loads are shown.

from Table 10 to Table 11, illustrated in Fig. 2. Resources with incoming inputs of tasks update their load CL_i^{k+1} and change position x_i^{k+1} by calculating the fitness value (objective function in Equation (4)).

Table 11: 4*th*-Iteration – updated load/postion of resources.

Resource (RES)	Current Load on RES)
b	44
a	43
c	43

Based on the resulting stage in Table 11, we can see that after the 4*th* iteration the load is balanced among all resources connected in cluster.

Fig. 3 visualizes the task load allocated to the resources for all iterations and resources.

5 EXPERIMENTAL VALIDATION

This section illustrates the experiments and their results that were obtained from evaluating the implementation of the proposed method.

In order to evaluate the research questions around the load balancing problem for particularly edge nodes in a distributed cloud environment, there are some parameters that need to be defined. We define the following key variables in the proposed autonomous A-PSO algorithm along with the cost estimation cycles for task scheduling and load balancing in cloud storage resources:

- Resources RES (cloud storage): range (2-5)
- Number of tasks in each iteration: range (2-6)
- Size (memory) of tasks randomly in each iteration: range (0-100)
- Number of iterations: range (10-20)
- Iteration weight: a.) position of RESs by fitness value: range (0-7); b.) position RESs by executions time cycles/ms: range (8-10)
- Max storage of RES1, RES2, RES3, RES4, RES set to 300, 500, 200, 250, 400 MBs, resp.

Table 12: Experimental Evaluation: Test cases and Results.

Sr.	RES	Tasks	Task Size	No Iteration	Iteration weight	Round-Trip Time (RTT)					Autonomous-PSO Algorithm					BK
						RES1	RES2	RES3	RES4	RES5	RES1	RES2	RES3	RES4	RES5	
1.	5	3	0-15	15	0-7 8-10	14	18	16	1	7	16	14	32	23	15	N
2.	5	4	0-30	10	0-7 8-10	19	24	18	21	20	48	32	43	38	44	N
3.	4	3	0-15	15	0-7 8-10	26	28	21	4	-	27	25	29	25	-	N
4.	3	3	0-15	15	0-7 8-10	15	16	14	-	-	38	38	39	-	-	N
5.	3	3	0-30	10	0-7 8-10	13	16	15	-	-	38	35	43	-	-	N
6.	3	5	0-30	10	0-7 8-10	21	21	19	3	-	67	31	71	16	-	Y
7.	3	4	0-25	12	0-7 8-10	25	26	25	-	-	57	57	57	-	-	N
8.	4	5	0-25	10	0-7 8-10	22	25	25	19	-	58	55	58	60	-	N
9.	4	5	0-25	15	0-7 8-10	27	24	26	24	-	62	63	64	62	-	N
10.	2	3	0-20	10	8-10 8-10	18	20	-	-	-	34	34	-	-	-	N
11.	2	3	0-50	10	0-7 8-10	13	12	1	-	-	72	61	17	-	-	Y
12.	2	5	0-30	15	0-7 8-10	23	23	17	-	-	65	93	59	-	-	Y
13.	2	6	0-25	15	0-7 8-10	31	33	27	-	-	63	65	67	-	-	Y
14.	3	2	0-100	10	0-7 8-10	13	14	1	1	-	72	67	80	10	-	Y
15.	4	2	0-70	20	0-7 8-10	13	15	5	2	7	81	63	78	63	54	Y

Figure 4: Experimental Results – here the load for resources RES hosting the storage tasks.

- Backup (BK) RES storage set to 1000 MBs

These are applied and shown in Table 12 below.

The test cases have been executed with both suggested methods, thus combining the autonomous PSO algorithm with the MOO-based execution cost cycles for task scheduling and load balancing. The results are presented in Table 12.

For instance, in the experimental test case 3, we took 4 resources and 3 tasks in each iteration, and we limited the size of tasks to between 0 to 15 (MBs). The total number of iterations was 15. In the results, we can see that the tasks are assigned to the cloud storages equally by the autonomous PSO algorithm and execution cycle management approach. Effectively, the load is balanced between resources (cloud storages) with minor difference in usage percentage.

Similarly, in experiment 4, we took 3 storage resources, 3 tasks in each iteration, 15 as the number of iterations and the size limits of incoming tasks were 0 to 15 MBs. As a results, we found that the total number of execution cycles per milliseconds were 15, 16, 14 in RES1, RES2 and RES3, respectively. The memory consumption was 38, 38 and 39 percent in RES1, RES2 and RES3, respectively.

If we look at the results of test cases 7 and 10, the tasks are scheduled to cloud storage resources equally, with an exactly balanced corresponding load. In test case 7, all 3 resources used up to 57 % of the total memory. In test case 10, 2 resources with 3 tasks in each iteration consumed 34 % of the total memory to execute the tasks until their termination.

A summary view of these test cases is provided in Fig. 4 that shows the continuous operation of the selected use cases and the resulting successfully balanced task load.

6 CONCLUSIONS

Our aim was to investigate the use of swarm intelligence for task scheduling and load balancing in distributed systems, specifically considering distributed cloud and edge computing environments. For this, we introduced an autonomous Particle Swarm Optimization (A-PSO) algorithm that we combined with Multi-Objective Optimization (MOO) .

In many investigations and applications of PSO algorithms for distributed systems management, for instance simulation techniques include the involvement of a central manager entity. Our work adopts and extends the standard PSO algorithm, but with the extension of s decentralized autonomous aspect. In addition, we also examined the cost of time to execute iteratively incoming task allocation requests to storage resources using multi objective optimization. By referring to Actual Round-Trip Time, we can calcu-

late the execution time of one single task allocated to a remote storage in cycles per milliseconds. The experimental results show that our enhanced A-PSO algorithm effectively balanced the load across the resources available. Thus, the autonomous coordination of task scheduling and load balancing here is realized by combining the two approaches – firstly, the A-PSO algorithm and secondly, the time costing together as a form of Multi Objective Optimization (MOO) for task scheduling, load balancing and reducing the execution cost of the incoming iterations of tasks for distributed system in the cloud computing environment.

In the future, we plan to extend the applicability of the solution to a wider range of distrbuted systems architecture, taking also other resource types such as compute and network into account.

REFERENCES

Mishra, S. K., Sahoo, B., and Parida, P. P. (2018). Load balancing in cloud computing: A big picture. Jrnl of King Saud University - Comp and Inf Sciences.

Pahl, C., Jamshidi, P., and Zimmermann, O. (2018). Architectural principles for cloud software. ACM Transactions on Internet Technology (TOIT) 18 (2), 17.

al-Rifaie, M. M., Bishop, J. M., and Caines, S. (2012). Creativity and autonomy in swarm intelligence systems. Cognitive computation 4.3: 320-331.

von Leon, D., Miori, L., Sanin, J., El Ioini, N., Helmer, S., and Pahl, C. (2018). A performance exploration of architectural options for a middleware for decentralised lightweight edge cloud architectures. Intl Conf on Internet of Things, Big Data and Security.

Jamshidi, P., Pahl, C., and Mendonca, N. C. (2016). Managing uncertainty in autonomic cloud elasticity controllers. IEEE Cloud Computing 3 (3), 50-60.

Tan, Y., Shi, Y., and Ji, X. (2012). Advances in Swarm Intelligence: Third International Conference ICSI.

Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. International Symposium on Micro Machine and Human Science. IEEE.

von Leon, D., Miori, L., Sanin, J., El Ioini, N., Helmer, S., and Pahl, C. (2019). A Lightweight Container Middleware for Edge Cloud Architectures. Fog and Edge Computing: Principles and Paradigms, 145-170.

Scolati, R., Fronza, I., El Ioini, N., Samir, A., and Pahl, C. (2019). A Containerized Big Data Streaming Architecture for Edge Cloud Computing on Clustered Single-Board Devices. CLOSER.

Kennedy, J. (2010). Particle swarm optimization. Encyclopedia of machine learning: 760-766.

Visalakshi, P. and Sivanandam, S. N. (2009). Dynamic task scheduling with load balancing using hybrid particle swarm optimization. Int. J. Open Problems Compt. Math 2.3:475-488.

Al-Maamari, A. and Omara, F.A. (2015). Task scheduling using PSO algorithm in cloud computing environments. Intl Journal of Grid and Distributed Computing 8.5:245-256.

Zhang, L. et al. (2008). A task scheduling algorithm based on PSO for grid computing. Intl Journal of Computational Intelligence Research 4.1:37-43.

Sharma, S. and Agnihotri, M. (2016). A Particle Swarm Optimization based Technique for Scheduling Workflow in Cloud DataCenter. Intl Journal of Engineering Trends and Applications 3.4.

Pandey, S., Wu, L., Guru, S. M., and Buyya, R. (2010). A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. Intl Conference on Advanced Information Networking and Applications.

Awad, A. I., El-Hefnawy, N. A., and Abdel_kader, H. M. (2015). Enhanced particle swarm optimization for task scheduling in cloud computing environments. Procedia Computer Science 65:920-929.

Selvarani, S., and Sudha Sadhasivam, G. (2010). Improved cost-based algorithm for task scheduling in cloud computing. International Conference on Computational Intelligence and Computing Research.

Awad, A.I., El-Hefnawy, N.A., and Abdel_kader, H.M. (2015). Dynamic Multi-objective task scheduling in Cloud Computing based on Modified particle swarm optimization. Advances in Computer Science: an International Journal 4.5:110-117.

Katyal, M. and Mishra, A. (2014). A comparative study of load balancing algorithms in cloud computing environment. arXiv preprint arXiv:1403.6918.

Acharya, J., Mehta, M., and Saini, B. (2016). Particle swarm optimization based load balancing in cloud computing. Intl Conf on Communication and Electronics Syst.

Mishra, R. and Jaiswal, A., (2012). Ant colony optimization: A solution of load balancing in cloud. Intl Journal of Web & Semantic Technology 3.2:33.

Pahl, C. and Lee, B. (2015). Containers and clusters for edge cloud architectures - a technology review. Intl Conf on Future Internet of Things and Cloud.

Shi. Y. (2001). Particle swarm optimization: developments, applications and resources. Proceedings Congress on Evolutionary Computation Vol. 1.

Kalpana, C., Karthick Kumar, U., and Gogulan, R. (2012). Max-Min Particle Swarm Optimization Algorithm with Load Balancing for Distributed Task Scheduling on the Grid Environment. Intl Journal of Computer Science Issues 9.3.