ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

Research Article

Open Access

# Load Balancing Based on Many-objective Particle Swarm Optimization Algorithm with Support Vector Regression in Fog Computing

**Mona Albalawi*, Entisar Alkayal, Ahmed Barnawi and Mehrez Boulares**

Faculty of Computing and Information Technology, King Abdul Aziz University, Jeddah, JE, 21589 SA

**ABSTRACT**

With the development in computing technologies, fog computing is developing as public and robust computing, which complements cloud computing to provide services, computation, and storage on the edge network. The future growth and support of 5G access networks additional advance the viability and implementation of fog networks and widen the scope of devices that can participate and serve in IoT communication. However, scaling fog computing, the number of end-users increases. Hence, the workload between fog nodes needs to be distributed efficiently. Otherwise, some of the nodes will be overloaded, and others will be under-loaded. Consequently, one of the critical factors for managing resources in fog computing efficiently and avoiding overloaded or under-loaded is load balancing. Therefore, load balancing between these resources is a challenge in fog computing. There are different techniques to balance the load, such as optimization algorithms or machine learning. This paper proposes a load balancing model in fog computing based on a many-objective particle swarm optimization (PSO) algorithm with support vector regression (SVR). The proposed load balancing model considered four metrics to optimize them while distributing the load: response time, energy consumption, resource utilization, and throughput. Besides, It combines SVR with PSO to improve PSO performance. The proposed model has been simulated and tested to evaluate the performance from different aspects. The experiments show that the proposed model efficiently balances the load with optimizing the four metrics. In addition, it improves the performance of PSO, which is used to balance the load.

***Corresponding author**
Mona Albalawi, Faculty of Computing and Information Technology, King Abdul Aziz University, Jeddah, JE, 21589 SA.
E-mail: mh.albalawi@ut.edu.sa

**Received:** January 27, 2022; **Accepted:** February 01, 2022; **Published:** February 03, 2022

**Keywords:** Optimization Algorithm, PSO, SVR, Fog Computing, Load Balancing

## Introduction

Currently, several types of devices connected to the internet, and the total devises' number continues to increase exponentially [1]. In essence, anyone and anything can connect to the internet anytime and anywhere; this is the notion of the Internet of Things (IoT) [1]. However, IoT may lack some of the data due to the constraints in IoT (limited energy, limited storage capacity, etc.) [2]. Providing consumers, the best experience when using IoT is critical since the speed of performance and high storage capacity are end-user requirements. Consequently, cloud and fog paradigms have been developed.

Fog Computing extends the cloud to handle the cloud's limitations, such as high latency, low security, etc [3]. Fog computing provides low latency, high performance, reliable, mobile, secure, and interoperable characteristics [3]. Fog computing offers service, computation, and storage to the end-users [3].

Fog computing consists of a huge number of users and heterogeneous resources spread in a wide geographic area. The increase in resources and users in IoT leads to increasing IoT traffic. This increase in IoT traffic may affect the load balance in fog computing. IoT traffic maybe over- loaded in some areas and under-loaded in others [2]. So, the processing workload must be distributed efficiently to avoid this problem. Load balancing between these resources is a significant research issue in fog computing [4,5]. Due to the importance of load balancing fog computing, the fast response to the users is very important in it [6].

Load balancing means efficiently distributing tasks among resources. Besides, load balancing helps resources perform tasks efficiently, leading to improved fog performance [7]. On the other hand, a lack of load balancing means that some fog nodes are under-loaded or idle, while others are overloaded. This problem will affect fog-computing performance, namely by increasing the response time, decreasing throughput, and increasing energy consumption. Consequently, the satisfaction of clients and service providers is negatively affected [8]. Our research aims to satisfy customers and service providers by improving fog computing to distribute the load efficiently among fog nodes. Many techniques and algorithms exist to balance the load in fog computing, such as optimization algorithms or machine learning.

The optimization algorithms are mathematical algorithms that maximize or minimize the objectives' value [9]. Optimization is the action of delivering the best possible decision under given conditions [9,10]. Regarding machine learning, it is a branch of artificial intelligence (AI) that supports the systems to learning themselves and make decisions with a little human intervention [11]. Machine learning models can be trained to make decisions based on using historical data [12]. Several researchers using these techniques to balance the load in fog computing.

This paper is organized as follows. Section 2 provides a problem definition and related works in load balancing in fog computing. Section 3 presents the proposed load balancing fog computing architecture. and explain the details about the proposed PSOSVR algorithm. Section 4 provides the details about the parameters that used in experiments. Section 5 presents the experiments results. Section 6, dis- cusses the results of the experiments and how the proposed model fulfills the contribution of this paper. Finally, Section 7 concludes this paper.

## Problem Definition

Load balancing means distributing the workload between nodes in a balanced way [2]. In fact, the number of fog users utilizing fog services is increasing exponentially. Further- more, the number of services on fog computing provided to users is also increasing rapidly. Due to many services and users on the fog, and without a suitable mechanism to distribute the workload among fog nodes, some nodes are under-loaded, and others are overloaded. Consequently, this affects the network's performance essential to the user, such as the response time and throughput because when nodes are overloaded, the nodes take more time to begin processes the task due to the huge number of tasks in a queue need to process. Moreover, it affects some parameters essential for service providers, such as energy consumption, resource utilization. To resolve this problem, the load between fog nodes needs to balance in the best way [2].

After we studying the previous works that applying optimization algorithms and machine learning to balance the load in fog computing, we noticed that the researches in this scope (i.e., load balancing in fog computing) distributes the load between fog nodes based on resource allocation or task scheduling. This means that it is not dealing with load balancing directly, and load balancing results from resource allocation or task scheduling. Therefore, we will focus on applying our proposed solution to resource allocation to balance the load. Besides, we focus on just computational tasks and do not consider any storage tasks.

Our target is balancing the load between fog nodes, improving the user's experience quality, and maximizing the service provider's benefit by decreasing response time, energy consumption, and increasing throughput and resource utilization. We utilize a hybrid load balancing algorithm named PSOSVR. We combine support vector regression (SVR) with Particle Swarm Optimization (PSO) algorithm to improve the PSO algorithm's performance. PSO has multiple advantages such as has few parameters, easy implementation, and efficiency to the optimization [13]. However, in PSO, the particles initialize randomly; this could decrease the algorithm's chance to converge to the best solution and take more time to it [14]. To solve this problem, the particles are initializing based on the prediction result from SVR.

Kamal et al. use a heuristic min-conflicts optimization algorithm for load balancing the VM in fog architecture [15]. The fog architecture has a load balancer that is responsible for allocating requests to VM in fog computing. The metrics of the optimization lead to minimizing processing and cost. However, the architecture here is centralized and there is no fault tolerance for example use a backup. Many constraints and complexity in the system could lead to slow work and, hence, to increase response time.

Load balancing in fog computing may occur through re- source allocation improves fog computing performance and resource utilization. Zafar et al. in propose a technique for resource allocation using the Bio-inspired Bat Algorithm (BA) to optimize the load balancing on the fog nodes. The system consists of six regions [16]. Each region comprises one fog which attached to tow clusters. Each cluster contains fifteen buildings that include multiple homes, and each fog connects with Micro-grid for electricity supply. The cloud analyst simulator is used to simulate the system. The simulation proves that BA minimizing processing time and response time. However, the proposed technique produces low Makespan when the closest data centers are not avail- able.

Zahoor et al. propose a hybrid algorithm that com- bines the Ant Colony Optimization (ACO) and Artificial Bee Colony (ABC) algorithms [17]. The authors use cloud-fog computing with smart grids to provide resources to end users efficiently, and they use the proposed hybrid algorithm to balance the load between the VMs in the fog layer. The simulation proves that the proposed technique reduces energy consumption. However, the architecture is centralized, and there is no fault-tolerance technique. Furthermore, the centralized architecture has low scalability.

Talaat et al. propose a new load balancing technique called Effective Load Balancing Strategy (ELBS). ELBS balances the load between fog nodes by scheduling tasks in an efficient way [18]. ELBS consists of five modules; including Fuzzy, and the other is the Probabilistic Neural Net- work. This technique distributes tasks between resources to improve performance. The architecture consists of the following four layers: "Cloud Layer, Fog Layer, Dew Layer, and End-User Layer". All the layers as well as the fuzzy and probabilistic neural network modules have their own procedures to carry out in order to achieve the overall goal. The overall goal is to provide load balancing with de- crease response time and increase throughput. However, the model's rescheduling process will take time, and there will be many migrations that will affect system performance.

## The Proposed Fog Computing Model

The proposed model's architecture consists of a hierarchal fog computing architecture to provide more resource allocation management and distribute the load among various nodes in several regions as shown in Figure 1. The model includes three framework layers, including an access layer, a local control layer, and a global control layer.

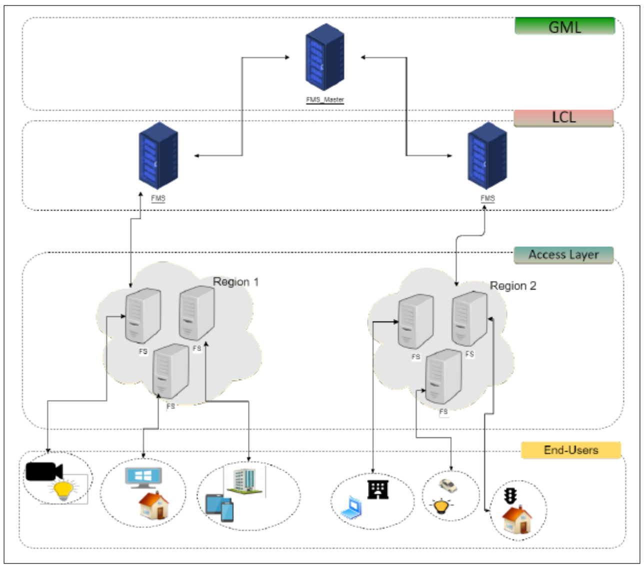


**Figure 1:** The System Architecture

• Access Layer:
In this layer of architecture, there are many Fog Server (FS) nodes. These FS are distributed into multiple regions, and each region has several FS nodes responsible for processing tasks from specific users. For instance, region-1 is a building with five floors. Hence, the region has five servers, and each server covers only one floor—the connections be- tween them are made through wireless communication links by routers and switches. The FS receives computational tasks only from end-users to process them and send the results back to the end-users.

• Local Control Layer (LCL):
LCL consists of multiple Fog Manager Server (FMS); there is only one FMS for each region. The FMS is responsible for managing the resource allocation between the fog servers in its region. This means that the FMS does not receive and process any tasks from end-users. It receives only the tasks that its FS cannot process and forwards them to the best available FS to process them. There is a table in the FMS that has information about all the fog servers in its region. This information comprises (Server ID, Task IDs that the server is still working on, and Available or not). This table is used to determine which FS is avail- able/unavailable FS in the region. Hence, there are no processing operations in the FMS; it is responsible for running the proposed load balancing PSOSVR for resource allocation and load balancing, which will explain in Section 4.

• Global Management Layer (GML):
GML is the higher layer that contains only one Fog Server Master (FMS Master). The FMS Master is responsible for finding the best available FS in all regions to process the task if the FMS in the LCL does not find any available FSs in its region to process the task. The FMS will forward the task to FMS Master to find the best available FS in another region to pro- cess it. A table in FMS Master contains information about all fog servers in all regions: Server ID, Task IDs that the server is still working on, available or not, and the server's region. This table is used to find available/unavailable FSs in all regions. Hence, there are no processing operations in the FMS Master; it is only responsible for running the proposed load balancing PSOSVR for resource allocation and load balancing, which will explain in Section 4.

## Load Balancing PSOSVR Algorithm

As explained in Section 3, FMS and FMS Master run load-balancing PSOSVR algorithm to distribute the load in a balanced way between the fog nodes. This section describe in detail the proposed load-balancing PSOSVR algorithm. The algorithm input is task T, and its output is the best FS node to process this task. Figure 2 presents the flowchart of the steps of load balancing PSOSVR algorithm. It consists of two main steps, which will explain in the next subsections in detail.

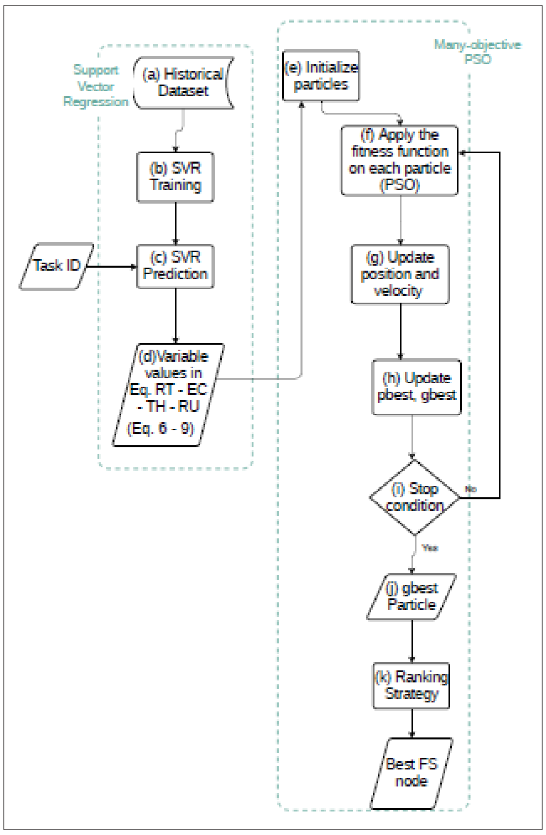


**Figure 2:** Flowchart illustrating the PSOSVR steps

## Support Vector Regression

Support vector regression (SVR) is a type of support vector machine. Proposed by Drucker et al. it supports linear

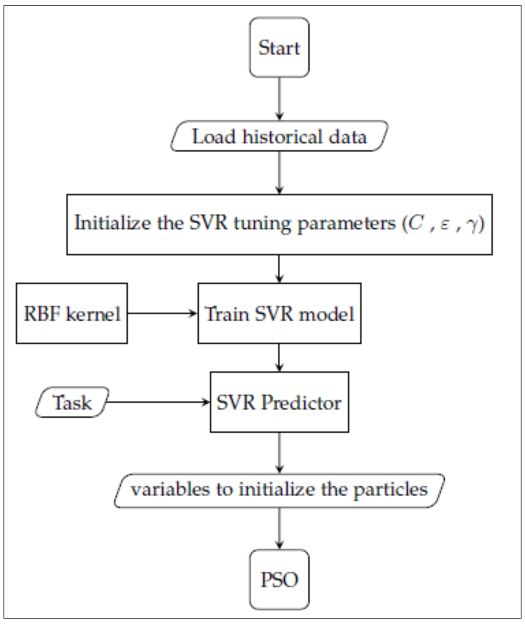

Start
Load historical data
Initialize the SVR tuning parameters ($C$, $\varepsilon$, $\gamma$)
RBF kernel
Train SVR model
Task
SVR Predictor
variables to initialize the particles
PSO

**Figure 3:** Flowchart illustrating the SVR model

and non-linear regressions [19]. Our research leverages the principle behind SVR to predict future output Z from S input samples. The prediction model is

$Z(t) = F(S(t))$, where $t$ is a data set.

To predict output $Z$ accurately, we use training sample $T = (S1, Z1)$, ....., $(Si, Zi)$ before we use the test sample [20,21]. The first step in the PSO algorithm involves initializing the particles randomly. This random initialization could reduce the algorithm's chance to converge to the best solution as well as require more time for the same [14]. Consequently, SVR can improve the performance of the PSO algorithm in terms of the execution time, average fitness value, and optimization time.

To apply SVR to predict the particle initialization, we utilized the Libsvm library in the Java environment. It is a simple, easy-to-use, and efficient software for SVR. Figure 3 shows the flowchart of the SVR predictor.

The steps in the SVR predictor in detail:
• Use a historical dataset for training and testing the SVR model as the input to the SVR. Source this historical dataset from our architecture used to run the round-robin algorithm for resource allocation five times. Implement 5000 tasks and store the output information about the variables using in Equations 1, 2, 3, and 4. Use these variables to initialize the particles:

- *Exct, Subt*
- *Pi, Ti, Pu, Tu*
- *UsedC, totalC*
- *C, T*

- Initialize the SVR parameters, namely constraint (C), gamma (γ), and epsilon (ε). Use the default values in the Libsvm library (i.e., $C = 1$, $\gamma = 1$, $\varepsilon = 1e{-}3$.
- Train the SVR model with the RBF kernel.
- After completing the SVR prediction mode, use the results to predict the variables in Equations 1 to 4, for each incoming task.
- Apply the result of the prediction as the input to the first step of the PSO algorithm (particle initialization).

## Many-objective PSO

Consequently, to obtain the result from applying many objective PSO, it will generate a PSO function for each metric, and then it will use the ranking strategy to determine the result from many-objective PSO. This means that it deal with the many-objective problem as one objective [22]. The ranking strategy uses to solve many-objective optimization problems because if the number of objectives increases, the convergence ability of particles decreases. Consequently, the ranking strategy deals with many objectives quickly by simplifying evaluation of the objective function [22,23]. Many- objective optimization algorithm optimizes problems that include more than three objectives. It is a special case from multi-objective optimization which means optimizes problems that include two or three objectives [24]. The steps of many-objective PSO used in the proposed solution with SVR to balance the load in fog computing are illustrated in the following flowchart. The proposed Many-objective PSO algorithms for the four metrics are shown in algorithm 1

**Algorithm 1: Many objective PSO algorithm**

```
1: for each $fs \in FC$ do
2: $RT(fs) \leftarrow RTPSO(fs)$
3: $EC(fs) \leftarrow ECPSO(fs)$
4: $RU(fs) \leftarrow RUPSO(fs)$
5: $TH(fs) \leftarrow THPSO(fs)$
6: end for
7: $bestFS$ = Ranking($RT(fs)$;$EC(fs)$;$RU(fs)$; $TH(fs)$) //the result
from algorithm3
8: return $best FS$
```

For each task that comes to the FMS or FMS_Master, they run many_objective PSOSVR, algorithm 1 will be executed. Steps 2, 3, 4, and 5 run in parallel for each server. The results from these steps will be the input for ranking (step 7), which returns the best FS for processing the task. The steps 2-5 are in details in the next paragraph.

**Algorithm 2: Standard PSO**

```
1: Initialize ($position$; $velocity$; $pbest$; $gbest$) //Particle initialization
based
on SVR prediction result.
2: while ($t < iteration$) do
3: for each particle $p$ do
4: $best \leftarrow FitnessFunction$(p) //Using Eq. 1 to 4
5: if ($best < pbest$) then
6: $pbest \leftarrow best$
7: end if
8: end for
9: if ($best < gbest$) then
10: $gbest \leftarrow best$
11: end if
12: Update ($position$ and $velocity$)
13: end while
14: return $gbest$
```

As presented in the algorithm 2, the particle's best fitness functions are calculated according to the fitness function. The fitness function is the equation of WT, EC, RU, or TH. Details of these metrics are as follows:

• Response Time (RT): The response time of a task is measured by computing the difference between the time the task is submitted to the system and the time of starting the execution of the task, as shown in Equation 1.
$RT(fs) = Exct(t) - Subt(t)$ (1)

where: $ExcT(t)$ indicates the start time of execution of the task $t$
$SubT(t)$ indicates the time for submission of the task $t$

• Energy consumption (EC): The energy consumption of each server fs can be represents as Equation 2
$EC(fs) = Pi * Ti + Pu * Tu$ (2)

where:
$Pi$ denotes the power consumption while idle
$Ti$ is the total idle time
$Pu$ is the power consumption while utilized
$Tu$ is the total time utilized

• Resource Utilization (RU): In our research, we focus on CPU utilization only because we focus only on computational tasks. Therefore, CPU utilization is computed by Equation 3
$CU(fs) = (UsedC(fs)/totalC(fs)) * 100$ (3) Where: $UsedC$ is the used CPU for all tasks executed in server $fs$
$totalC(fs)$ is the total CPU of server $fs$

• Throughput (TH): Throughput is the number of tasks that execute in a simulation time.
$TH(fs) = (C(fs)/T$ (4)

where: $TH$ indicates the throughput of server $fs$
$C$ is the total number of completed tasks
$T$ is the simulation time in seconds

As shown in Algorithm1, after determining each server's best particle in the four metrics, the ranking strategy will execute (step7). Our research computes the many-objective optimization algorithm in a short time based on a modified ranking strategy developed by to simplify objective function evaluation [23].

**Algorithm 3: Ranking Strategy for many-objectives**

```
1: for each $fs$ do
2: $f1(fs) = Min(RT(fs);EC(fs);RU(fs); TH(fs))$ Using Equation 5
3: $f2(fs) = Sum(RT(fs);EC(fs);RU(fs); TH(fs))$Using Equation 6
4: $rank(fs) = f1(fs)\ 0{:}5 + f2(fs) * 0{:}5$
5: end for
6: bestfs ← min(rank)
7: return $bestfs$
```

The ranking strategy is based on the minimum ranking and sum ranking strategies. The minimum ranking (step 2) in algorithm 3 is Equation 5. The sum of ranking (step 3) is shown in Equation 6. Finally, in step 4, a weighted sum is computed for the minimum and sum ranks for all servers to find the solution's final rank, as presented in Equation 7.

$$f1(fs) = Min(RT(fs), EC(fs), RU(fs), TH(fs)) \quad (5)$$

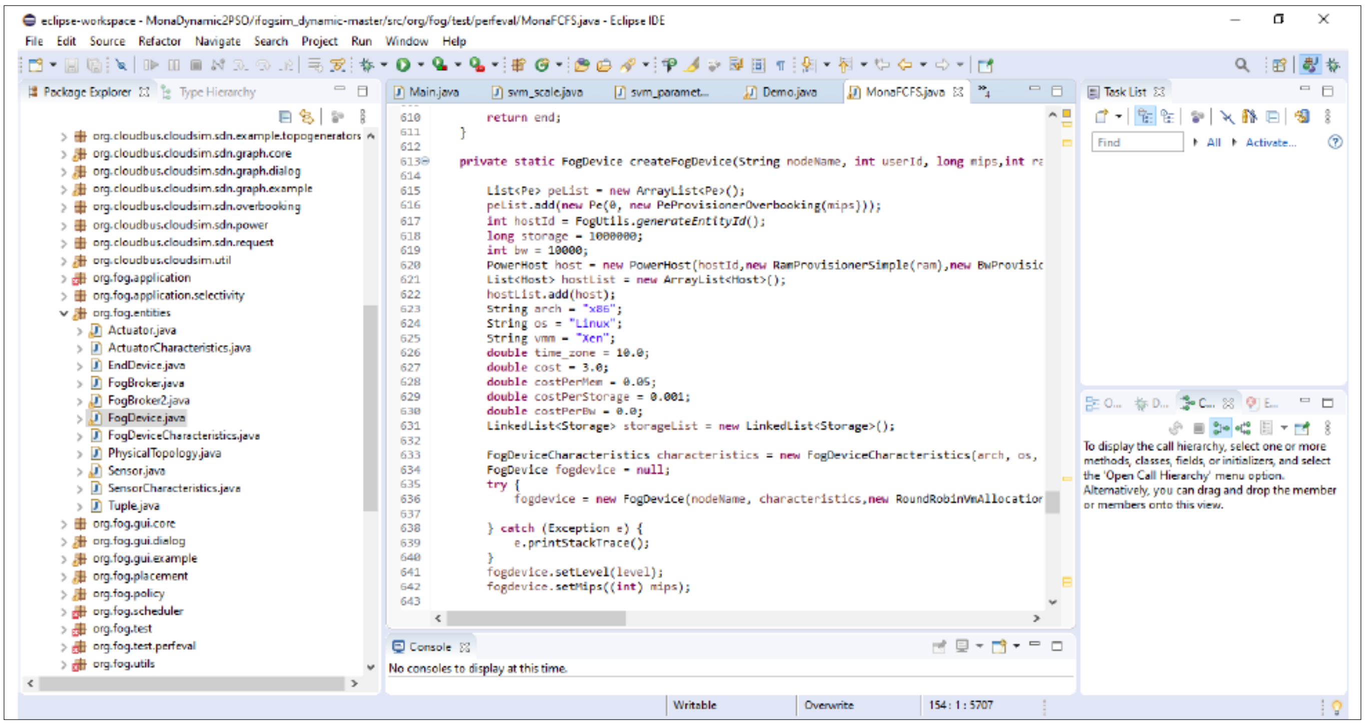
$$f2(fs) = Sum(RT(fs), EC(fs), RU(fs), TH(fs)) \quad (6)$$

$$rank(fs) = f1(fs) * 0.5 + f2(fs) * 0.5 \quad (7)$$

Finally, after all steps of the proposed load balancing PSOSVR are applied, the best FS is determined as the best server to process the incoming task. Therefore, FMS or FMS_Master will send the task to this best FS for processing.

## Experiments

To simulate the proposed architecture, iFogSim simulation is used to simulate the proposed fog computing [25]. iFogSim is an open-source toolkit in Java language used to simulate fog computing environments. It is used to evaluate the resource allocation and scheduling algorithms in a fog computing environment; thus, we could measure the effect of energy consumption, operational cost, latency, and other parameters on the suggested strategies. Figure 4 presents a screenshot of the iFogSim environment.


eclipse-workspace - MonaDynamic2PSO/ifogsim_dynamic-master/src/org/fog/test/perfeval/MonaFCFS.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Type Hierarchy
Main.java
svm_scale.java
svm_paramet...
Demo.java
MonaFCFS.java
Task List
Find
All
Activate...
Console
Writable
Overwrite
154 : 1 : 5707

```
org.cloudbus.cloudsim.sdn.example.topogenerators
org.cloudbus.cloudsim.sdn.graph.core
org.cloudbus.cloudsim.sdn.graph.dialog
org.cloudbus.cloudsim.sdn.graph.example
org.cloudbus.cloudsim.sdn.overbooking
org.cloudbus.cloudsim.sdn.power
org.cloudbus.cloudsim.sdn.request
org.cloudbus.cloudsim.util
org.fog.application
org.fog.application.selectivity
org.fog.entities
    Actuator.java
    ActuatorCharacteristics.java
    EndDevice.java
    FogBroker.java
    FogBroker2.java
    FogDevice.java
    FogDeviceCharacteristics.java
    PhysicalTopology.java
    Sensor.java
    SensorCharacteristics.java
    Tuple.java
org.fog.gui.core
org.fog.gui.dialog
org.fog.gui.example
org.fog.placement
org.fog.policy
org.fog.scheduler
org.fog.test
org.fog.test.perfeval
org.fog.utils
```

```java
        return end;
    }

    private static FogDevice createFogDevice(String nodeName, int userId, long mips,int ra

        List<Pe> peList = new ArrayList<Pe>();
        peList.add(new Pe(0, new PeProvisionerOverbooking(mips)));
        int hostId = FogUtils.generateEntityId();
        long storage = 1000000;
        int bw = 10000;
        PowerHost host = new PowerHost(hostId,new RamProvisionerSimple(ram),new BwProvisic
        List<Host> hostList = new ArrayList<Host>();
        hostList.add(host);
        String arch = "x86";
        String os = "Linux";
        String vmm = "Xen";
        double time_zone = 10.0;
        double cost = 3.0;
        double costPerMem = 0.05;
        double costPerStorage = 0.001;
        double costPerBw = 0.0;
        LinkedList<Storage> storageList = new LinkedList<Storage>();

        FogDeviceCharacteristics characteristics = new FogDeviceCharacteristics(arch, os,
        FogDevice fogdevice = null;
        try {
            fogdevice = new FogDevice(nodeName, characteristics,new RoundRobinVmAllocatior

        } catch (Exception e) {
            e.printStackTrace();
        }
        fogdevice.setLevel(level);
        fogdevice.setMips((int) mips);

```

No consoles to display at this time.

To display the call hierarchy, select one or more methods, classes, fields, or initializers, and select the 'Open Call Hierarchy' menu option. Alternatively, you can drag and drop the member or members onto this view.

**Figure 4:** iFogSim environment

Table 1 presents the configuration of our architecture. We use the default configuration in the iFogSim simulation as in iFogSim tutorial paper [26].

**Table 1: System Configuration for Experiments**

| Parameters | The number of parameters | Up link Bandwidth | Down link Bandwidth | CPU (MIPS) | Memory (Ram) |
|---|---|---|---|---|---|
| Fog Server(FS) | 50 | 10000 | 10000 | 1500 - 4100 | 4000 - 6300 |
| Fog Manager Server (FMS) | 10 | 10000 | 10000 | 3550 - 8000 | 4000 - 8000 |
| Fog Manager Master (FMS Master) | 1 | 10000 | 10000 | 44800 | 40000 |
| End Users | 500 | - | - | - | - |

The testing relied on many methodologies for experiments and covered various aspects to measure the performance. After running the simulation, particular aspects are measured based on our contributions. After running the simulation, particular aspects are used to measure the performance are the following:

- **Response time:** calculate the average of the difference between the time of start to execute the tasks and the submitted time of these tasks. Equation 1 is used to measure response time.
- **Energy consumption:** is the average of the energy consumption of each server. Equation 2 is used to measure energy consumption.
- **Throughput:** The number of completed tasks within a simulation time. The equation of throughput has defined in Equation 4.
- **Resource Utilization:** Equation 3 used to calculate CPU utilization of fog servers and calculate the average for each experiment.
- **The Imbalance Degree:** is the degree to measure the load balancing between nodes. The small value of the imbalance degree means that the load is more balanced [27,23]. The equation of imbalance presents in the Equation 8.

$$IMD = (TE_{max} - TE_{min})/TE_{avg} \qquad (8)$$

Where:
*IMD* is the imbalance degree

$TE_{max}$ is the maximum execution time of tasks
$TE_{min}$ is the minimum execution time of tasks
$TE_{avg}$ is the average execution time of tasks

**Execution Time:** is the overall time to run the simulation. The simulation program calculates the executiontime.

*Simulation_End_Time-Simulation_Start_Time* (9)

• **The optimization time or convergence time:** is used to determine the time taken by the algorithm to converge to the best solution [28]. It is calculated as the following equation:

$$OT = \sum_{i=0}^{i=max} T_i. \qquad (10)$$

Where:
$T_i$ is the time taken for $i^{th}$ iteration

$\sum_{i=0}^{i=max} T_i$ is the total time for complete iteration

**The average fitness value:** is determine if or if not, the algorithm falls in local minima [23,28]. It calculates the average of all values of the fitness function.

$$FV = \sum_{iT=0}^{iT=max} F(iT). \qquad (11)$$

Where:
*FV* is the average fitness value for all tasks
*iT* is the index of the task
*F*(*iT* ) is the value of fitness function for all solutions

**Results and Discussions**
There are three experiments to evaluate different aspects of the proposed architecture, each experiment was run ten times and calculate the average value. The next sub-sections will present details of the three methodologies used in the test phase.

**Improve the PSO Algorithm Performance**
This experiment in our research is to test the effect of integrating SVR with PSO. Test if it does or does not improve PSO algorithm performance in terms of execution time, average fitness value, and optimization time. In this test, we compare PSOSVR and PSO algorithms.

The Execution Time:
It is the overall time to run the simulation.
Figure 5 show the comparison between PSO and PSOSVR results according to the execution time.

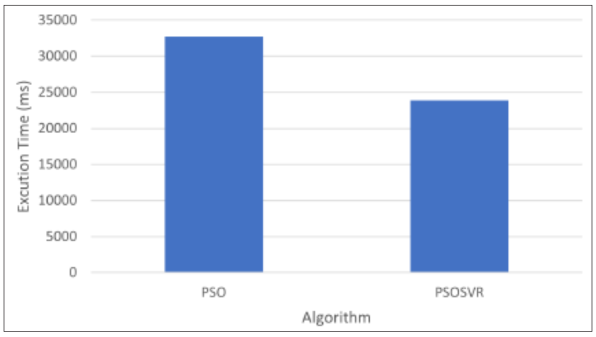


**Figure 5:** PSO vs. PSOSVR in term of execution time

Figure 5 shows the effectiveness of integrating SVR with PSO. It reduces the execution time of the algo- rithm, thus improve the performance of the PSO al- gorithm. The execution time was reduced by about 27%. The decreasing execution time results from ini- tializing the particles position based on prediction results from SVR. Because the random initialization increases the number of walking particles, and that this has a negative influence on convergence time in PSO, consequently, the execution time of the algorithm increases [29].

**2) The optimization time:**
The optimization time called convergence time is used to determine the time taken by the algorithm to converge to the best solution [28]. As we mentioned earlier, if PSO particles initialize randomly, this could decrease the algorithm's chance to converge to the best solution and take more time to it [14]. To solve this problem, the particles are initializing based on the prediction result from SVR.

Figure 6 illustrates the comparison between PSO and PSOSVR results according to the optimization time.

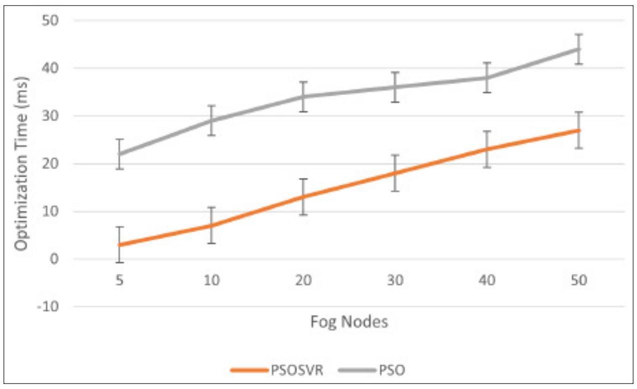


**Figure 6:** PSO vs. PSOSVR in term of optimization time

Figure 6 shows the effectiveness of initializing particles based on the result of SVR prediction. It reduces the optimization time taken by the algorithm to converge to the best solution, thus improving the PSO algorithm's performance. We test it by applying the Equation 10 and measure it according to changing the numbers of fog nodes. As shown in Figure 6, the PSO and PSOSVR algorithms have a linear relationship between the optimization time and the numbers of fog nodes. However, PSOSVR reduces the optimization time by 58%. This result means PSOSVR optimizes the solution faster than PSO.

**3) The average fitness value:**
It determines if or if not the algorithm falls in local minima. It calculates the average of all values of the fitness function. If the fitness function values for all solutions are the same or close to each other, this means the algorithm falls in local minima, and the personal best position (pbest) may not change across several iterations [23,28].

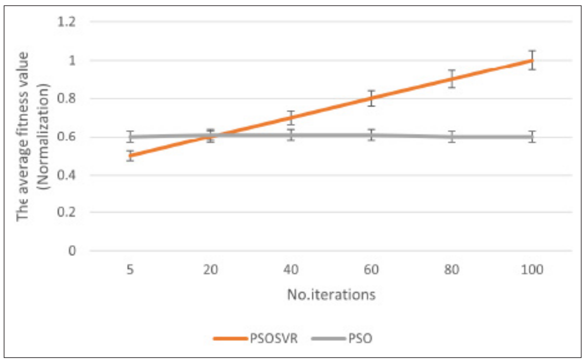


**Figure 7:** PSO vs. PSOSVR in term of the average fitness value

Figure 7 shows the effectiveness of SVR on PSO in according prevent PSO from falling in local optima. We measure the average fitness value according to changing the iteration numbers in PSO. The iterations are the step 2 in algorithms 2. We notice in Figure 7 that the PSO fitness value did not change; it is the same while the number of iterations changing. This result means the PSO falls in local optima, and the personal best (pbest) position is approximately the same. In contrast to, PSOSVR the average fitness value changes as the number of iterations change. This result proves PSOSVR did not fall in local optima.

**Proving load balancing between fog nodes**
In this experiment, we measured if the proposed algorithm balances the load between fog nodes or not. We calculated the imbalance degree by Equation 8. We calculated the difference between the maximum execution time of all tasks and the minimum execution time of all tasks then divided it by the average execution time of all tasks. If the imbalance degree is small, this is means there is a balancing between fog nodes. The next Table illustrates a comparison between the proposed PSOSVR, RR, and FCFS according to imbalance degree.

**Table 2: The Imbalance Degree and No. of Fog Nodes**

| No. of Fog Nodes | PSOSVR | RR | FCFS |
|---|---|---|---|
| 4 | 7.06 | 43.54 | 39.37 |
| 6 | 7.06 | 42.06 | 82.30 |
| 8 | 5.94 | 37.06 | 41.07 |
| 10 | 0.21 | 32.33 | 25.60 |
| 15 | 0.11 | 40.42 | 8.40 |
| 20 | 0.21 | 39.65 | 41.82 |
| 25 | 0.30 | 39.76 | 39.59 |
| 30 | 0.11 | 16 | 8.30 |
| 35 | 0.42 | 15.72 | 25.01 |
| 40 | 0.11 | 15.72 | 24.96 |
| 45 | 0.11 | 42.08 | 12.51 |
| 50 | 0.12 | 42.08 | 9.98 |

As shown in Table 2, we note that the proposed algorithm PSOSVR has the lowest imbalance degree comparing to other algorithms. No matter how the number of fog nodes varies, the proposed algorithm achieves the load balance according to the lower imbalance degree, so there is a very high load balance between them. Another observation is that if we look at the column of the PSOSVR algorithm, we will notice approximately the imbalance degree decreases as we increase the number of nodes. Thus, the more nodes we increase, the load balance is achieved more between nodes. We conclude from this: the significant number of fog nodes does not affect the load balance, but on the contrary, it increases the load balance.

**Performance Comparison between PSOSVR, FCFS and RR**
This experiment Compare PSOSVR with FCFS and RR to prove that the proposed PSOSVR decrease response time, energy consumption, and increase throughput, resource utilization better than RR and FCFS. Figure 8 show the simulation results according to the four aspects, which are RT, EC, TH, RU.

Figure 8 (a) shows that the relationship between RT values and the number of tasks is a linear relationship as the value of RT changes linearly with the number of tasks increases. It is observed that the proposed PSOSVR method has achieved the lowest RT compare to RR and FCFS, the RT was reduced by about 31%. Hence, the first objective of our optimization algorithm of load balancing is achieved.

Similarly, Figure 8 (b) shows the linear relationship be- tween EC values (normalized to the maximum value) and the number of tasks. It is observed that the proposed solution has performed significantly better than both RR and FCFS by about 56%. However, it is observed that the performance of FCFS and RR are almost similar to one another because there is no sleeping mode feature on the servers that do not have tasks to process, so these servers remain working all the time even if they do not have any task to process; this explains why the performance of FCFS and RR is almost similar to one another in energy consumption. Contrary to our proposal model, the load is distributed based on reducing energy consumption. As far as possible, the servers that do not have any load should not receive tasks to process, and these servers go to sleeping mode to reduce the energy.

On the other hand, Figure 8 (c) shows how the CPU utilization is affected as the number of tasks increases. We can observe that the proposed solution has achieved the highest utilization level in comparison to RR and FCFS by 26%. This reflects that, on the one hand, our system utilizes better the available resources, and on the other that the utilization of the system is not exponentially consumed. This is because, as we explained earlier, in the consumption of energy to reduce it, we try as much as possible servers that do not have a load to go to sleep mode. This explains to us that the resources are consumed at the highest possible value, taking into account the reduction in energy consumption.

Finally, Figure 8 (d) shows that the proposed solution has achieved the highest TH (in terms of byte/second) than RR and FCFS. It is shown that we can improve the throughput by 46%, and thus the fourth objective of our optimization algorithm of load balancing is achieved, which is to increase the throughput.

## Discussion

The aim of proposing the load balancing PSOSVR algorithm is balancing the load between fog nodes with optimizing the response time, energy consumption, throughput, and re- source utilization. SVR combined with the PSO algorithm to improve the PSO algorithm performance. The experiments aimed to evaluate PSO performance after utilizing SVR to initialize the particle in PSO. In addition, they evaluate the proposed PSOSVR algorithm in terms of distributes the load in a balanced way. Finally, they evaluate the proposed PSOSVR algorithm optimizes the response time, energy consumption, resource utilization, and throughput while distributing the load.
The main results gathered from the experiments are as follows:

• Comparing PSOSVR and PSO to prove that initialize the particles by SVR provides a positive effect on the PSO performance:
– Combining SVR with PSO reduces the execution time by about 27% compared to the PSO.
– Combining SVR with PSO reduces the optimization time by about 58% compared to the PSO.This result means PSOSVR optimizes the solution faster than PSO.
– By evaluate the average fitness value of PSO and PSOSVR and compared them, we conclude that PSOSVR did not fall in local optima like PSO.

• Comparing imbalance degree results of PSOSVR, RR, and FCFS prove that:
– PSOSVR has the lowest imbalance degree comparing to other algorithms. No matter how much fog nodes in the architecture, PSOSVR achieves the load balance according to the lower imbalance degree, so there is a very high load balance between the nodes.

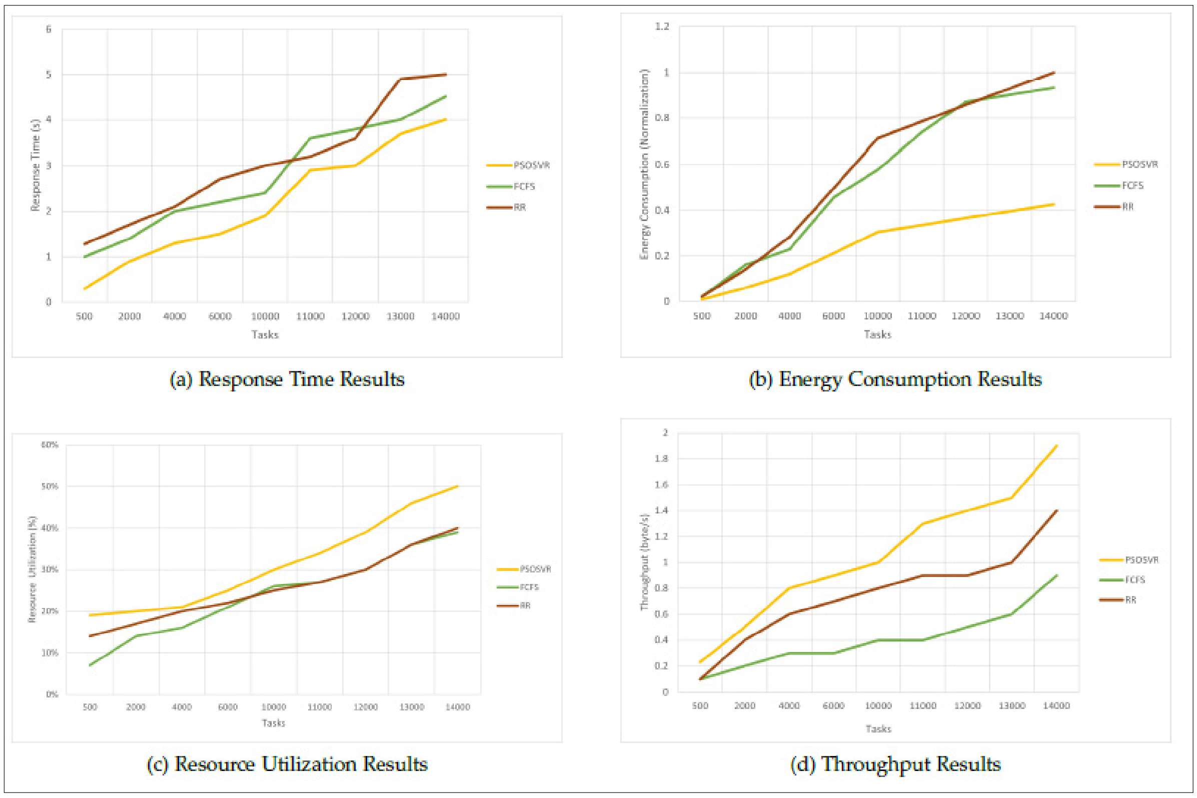


**Figure 8:** The First Experimental Methodology Results

• Comparing PSOSVR results with FCFS and RR re- sults prove that the proposed PSOSVR:
– Decrease response time by about 31%.
– Decrease energy consumption by about 56%.
– Increase throughput by 26%.
– Increase resource utilization by 46%.

## Conclusion

With the fast growth of IoT, fog computing is becoming one of the most robust paradigms for processing IoT applications. The future growth and support of 5G access networks additional advance the viability and implementation of fog networks and widen the scope of devices that can participate and serve in IoT communication. However, scaling fog computing, the number of end-users increases. Hence, the workload between fog nodes needs to be distributed efficiently. Consequently, one of the critical factors for man- aging resources in fog computing efficiently and avoiding overloaded or under-loaded is load balancing. Therefore, load balancing between these resources is a challenge in fog computing. However, there is an issue in load balancing, which has not yet been fully solved. Load balancing is an essential field in fog computing to avoid over/under loading. Various algorithms are used to optimize load balancing in fog environments for giving a better quality of service. This paper proposes a many-objective PSOSVR to distribute the tasks among resources in an efficient way to satisfy load balancing. The proposed load balancing model considered four metrics to optimize them while distributing the load: response time, energy consumption, resource utilization, and throughput. Besides, It combines SVR with PSO to improve PSO performance. The proposed model has been simulated and tested to evaluate the performance from different aspects. The experiments show that the proposed model efficiently balances the load with optimizing the four metrics. Besides, it improves the performance of PSO, which is used to balance the load. In the future we will deal with the storage tasks and resources (size of the disk and the memory size) when distributing the load, improve the proposed architecture, and use real tasks dataset to test the solution.

## Acknowledgments

I would like to send my most heartfelt thankfulness to all those who supported me in completing this paper. Send my gratitude to my supervisors, Prof. Ahmed, Dr. Entisar, for their incredible personal and professional support to me. Also, I want to thank Dr. Mehrez for his helping and support. Moreover, my biggest thanks to my family for all the support you have shown me through this thesis.

## References

1. Alfaqih TM and Al-Muhtadi J (2016) Internet of things security based on devices architecture. International Journal of Computer Applications 975: 8887.
2. Chowdhury A, Raut SA (2018) A survey study on internet of things resource management. Journal of Network and Computer Applications 120: 42–60.
3. Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. Proceedings of the first edition of the MCC workshop on Mobile cloud computing 13-16 ACM.
4. Hu P, Dhelim S, Ning H, Qiu T (2017) Survey on fog comput- ing: architecture, key technologies, applications and open issues. Journal of network and computer applications 98: 27-42.
5. Mukherjee M, Shu L, Wang D (2018) Survey of fog computing: Fundamental, network applications, and research challenges. IEEE Communications Surveys & Tutorials 20:1826- 1857.
6. Talaat FM, Saraya MS, Saleh AI, Ali HA, Ali SH (2020) A load balancing and optimization strategy (lbos) using reinforcement learning in fog computing environment. Journal of Ambient Intelligence and Humanized Computing 1-16.
7. Chou TCK, Abraham JA (1982) Load balancing in distributed systems," IEEE Transactions on Software Engineering 4: 401-412.
8. Mir RN (2018) Resource management in pervasive internet of things: A survey. Journal of King Saud University-Computer and Information Sciences.
9. Rai D, Tyagi K (2013) Bio-inspired optimization techniques: a critical comparative study. ACM SIGSOFT Software Engineering Notes 38: 1-7.
10. Deb K (2014) Multi-objective optimization in Search methodologies Springer 403-449.
11. Mitchell TM, Carbonell JG, Michalski RS (1986) Machine learning: a guide to current research. Springer Science & Business Media 12.
12. Kashyap P (2018) Machine Learning for Decision Makers: Cognitive Computing Fundamentals for Better Decision Making Apress.
13. Wang Y (2019) Research of improved particle swarm optimization algorithm based on big data," in 2019 International Conference on Robots & Intelligent System (ICRIS) 287-290.
14. Alkhashai HA, Omara FA (2016) "An enhanced task scheduling algorithm on cloud computing environment. International Journal of Grid and Distributed Computing 9: 91-100.
15. Kamal MB, Javaid N, Naqvi SAA, Butt H, Saif T, et al. (2018) Heuristic min-conflicts optimizing technique for load balancing on fog computing in International Conference on Intelligent Networking and Collaborative Systems Springer 207-219.
16. Zafar F, Javaid N, Hassan K, Murtaza S, Rehman S, et al. (2018) Resource allocation over cloud-fog framework using ba," in International Conference on Network-Based Information Systems. Springer 222-233.
17. Zahoor S, Javaid S, Javaid N, Ashraf M, Ishmanov F, et al. (2018) Cloud–fog–based smart grid model for efficient re- source management. Sustainability 10: 2079.
18. Talaat FM, Ali SH, Saleh AI, Ali HA (2019) Effective load balancing strategy (elbs) for real-time fog computing environment using fuzzy and probabilistic neural networks. Journal of Network and Systems Management 27: 883-929.
19. Drucker H, Burges CJ, Kaufman L, Smola AJ, Vapnik V (1997) Support vector regression machines in Advances in neural information processing systems 155–161.
20. Devia Agustina S, Bella C, Anang Ramadhan M (2018) Support vector regression algorithm modeling to predict the availability of foodstuff in indonesia to face the demographic bonus. JPhCS 1028: 012240.
21. Zhong Y, Zhao L, Liu Z, Xu Y, Li R (2010) Using a support vector machine method to predict the development indices of very high water cut oilfields. Petroleum Science 7: 379-384.
22. Garza-Fabre M, Pulido GT, Coello CAC (2009) Ranking methods for many-objective optimization in Mexican international conference on artificial intelligence Springer 633-645.
23. Alkayal E (2018) Optimizing resource allocation using multi- objective particle swarm optimization in cloud computing systems. PhD thesis, University of Southampton.
24. Mesbahi M, Rahmani AM (2016) Load balancing in cloud computing: a state of the art survey. Int. J. Mod. Educ. Comput. Sci, 8: 64.
25. Gupta H, Vahid Dastjerdi A, Ghosh SK, Buyya R (2017)

ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Software: Practice and Experience 47: 1275-1296.

26. Mahmud R, Buyya R (2019) Modelling and simulation of fog and edge computing environments using ifogsim toolkit. Fog and edge computing: Principles and paradigms 1-35.
27. Kong L, Mapetu JPB, Chen Z (2020) Heuristic load balancing based zero imbalance mechanism in cloud computing Journal of Grid Computing 18: 123-148.
28. Junaid M, Sohail A, Rais RNB, Ahmed A, Khalid O, et al. (2020) Modeling an optimized approach for load balancing in cloud," IEEE Access 8: 173208-173226.
29. Engelbrecht A (2012) Particle swarm optimization: Velocity initializa- tion," in 2012 IEEE congress on evolutionary computation. IEEE 1-8.