# An Augmented Load-Balancing Algorithm for Task Scheduling in Cloud-Based Systems

Franck Seigneur Nininahazwe, Jian Shen, Micheal Ernest Taylor

School of Computer and Software, Nanjing University of Information Science and Technology, China
seigneurinuyasha777@yahoo.fr, s_shenjian@126.com, delen007@live.com

## Abstract

Task scheduling in the cloud offers many advantages to cloud providers and users, such as managing cloud computing performances and maximizing resource utilization. However, the load might not be balanced among the multiple data centers leading to some servers being overloaded while others are idle or barely working. This paper proposes an augmented load-balancing algorithm (ALA) inspired by particle location-based search system and the Artificial Bee Colony's (ABC) memory mechanism. The search system is modified by adding the best response time criterion, best path and a data center level-based distribution system to ensure an even load handling. In contrast with the ABC and Particle Swarm Optimization (PSO) algorithms, the (ALA) takes into account the number of virtual machines (VMs) per host and the response time of each data center when scheduling the given tasks. The proposed algorithm is evaluated against other well-known techniques with a different number of experiment using the designed system model proposed. The experiments results show that (ALA) distributed the load as equally as possible and kept the system balanced having an improved response time and processing time.

**Keywords:** Cloud computing, Artificial Bee Colony, Particle Swarm Optimization, Load-balancing, Data centers

## 1 Introduction

The National Institute of Standards and Technology (NIST) defines cloud computing as: "A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1].

The institute also defined 4 categories of cloud computing. For public cloud, the full computing infrastructures are situated in the buildings of the enterprises that are providing different cloud services. Companies or organizations with the same objective use community cloud jointly (student community, professional community). A private cloud is owned and controlled by one company. Moreover, Hybrid cloud uses both public and private clouds [1].

The cloud has four major services, Infrastructure as a service offers you the possibility to rent infrastructures such as servers maintained by a cloud provider. Software as a service is a subscription-based method that offers software over the internet. This service uses host software, manage it and handle any update or security patch. Platform as a service is a development service that provides testing and deliver mobile or web applications. Functions as a service includes one more layer to PaaS. It allows developers to upload working chunks of code that are intended to be set off by a specific action. This service does not use any of IaaS resources until the specified event starts, which reduces the utilization fees [1].

Cloud computing has been gaining some popularity over the last few years. Using it comes with many advantages such as not having to buy and maintain your own IT infrastructure, storage or networking but instead pay for only what you use [2-4].

Cloud computing also comes with some responsibilities, such as ensuring all user requirements and services are met. This makes load balancing a major task in cloud computing. It is a series of actions that allows the load to be distributed equally among many servers, nodes or VMs preventing other nodes from being overloaded, under loaded or idle [2-4]. A good load balancing technique ensures this. It helps high traffic websites and enterprises achieve high performances with lower costs and business steadiness. The absence of one could lead to system overload and an unsatisfied client.

Cloud load balancing benefits comes from the scalable and global aspect of the cloud itself. An Efficient load balancer can handle increased traffic and redistribute it to different servers or networks. High-performing client applications can work faster and produce better performances. If a server or node fails, the workload can be redistributed to other working servers or nodes efficiently. It can be used by big and

*Corresponding Author: Franck Seigneur Nininahazwe; E-mail: seigneurinuyasha777@yahoo.fr
DOI: 10.53106/160792642021122207001

small businesses to balance the load across several cloud resources [2-7].

Load balancing techniques are far from perfect. Some challenges need to be addressed, such as maintaining the system's stability while increasing its performance, decreasing the execution time of the given tasks, and improving the load balancing technique's response time. Efficient resource utilization and other challenges that are being addressed by researchers [1-7].

The solution to load balancing issues is a better task scheduling algorithm. There are many types of load-balancing techniques which are generally grouped based on system state [1] or process initiation [5]. The System state-based algorithms classification is discussed further later.

The focus of this work is to design an augmented load-balancing algorithm that combines the search system of the Particle Swarm Optimization with the memory mechanism of Artificial Bee Colony. The response time of each datacenter is added in the search system allowing optimized searching for the most suitable to process the request, compute the best path and schedule the tasks by considering how many available VMs per host at given. A well-balanced system is designed to enable equal distribution of tasks among multiple VMs and processes. Based on in-depth research, this work has not been done. Unlike most researchers in the field, this work improves the servicing time and the degree of imbalance of the system. It focuses on the response time, resources usage and the standard deviation of the servicing time.

## 2 Related Work

Allocation of tasks in an even manner is a huge problem in cloud computing. Being able to distribute tasks properly means resources are properly used. A good task scheduling algorithm should be involved to solve this issue of which many solutions have been proposed. The focus of this work is on state based load-balancing algorithms.

Authors in [8] propose a task scheduling technique inspired by ant colony concentrating on the infrastructure as a service. Simulations performed using CloudSim prove that the scheduler performs better than algorithms based on random assignments and genetics. However, the main issue is that it was compared with only those two. M. Junaid et al. [9] propose a hybrid model that classifies the files available in the cloud-based on their format. They used Support Vector Machine (SVM) to classify files according to their format such as video, images and audio in the cloud. The grouping is then fed into Ant Colony Optimization (ACO) using File Type Formatting (FTF) for greater load-balancing in the cloud.

Walaa Hashem et al. [10] and Dhinesh Babu et al. [11] propose an improved algorithm based on the honey bee. After the experiments, their algorithms perform better than well-known algorithms of the same category such as ABC and ACO. L. Shen et al. [12] suggest an optimized ABC inspired algorithm to boost the overall load-balancing execution and realize greater adaptivity. However, their algorithm was only compared with two other type of ABC algorithms.

Akash Dave et al. [13] proposed a technique based on PSO in order to balance the load on Xen servers. They used the original PSO and only made the comparison with a single algorithm. Jigna Acharya et al. [14] proposed another algorithm based on PSO but tried to reduce the amount of time it took to complete a set of tasks. They only tested the makespan of their algorithm and only against the First Come First Served (FCFS) algorithm.

Thanh Tung Khuat et al. [15] proposed a hybrid technique of ABC and PSO in order to solve effort estimation using agile methodologies in software projects. The results show that their method outperformed ABC and PSO. Abraham Kiran Joseph et al. [16] introduce a hybrid technique of ABC and PSO but for test case optimization. Noosheen Baktash et al. [17] worked on a hybrid ABC and PSO task scheduling technique for optimization in a dynamic environment which proved to perform better against algorithms like Multi Quantum Swarm Optimization (mQSO) and Resampling Particle Swarm Optimization (RPSO). Their focus was on offline errors. S.G. Domanal et al. [18] developed a hybrid bio-inspired technique that combines a modified PSO plus a modified Cat Swarm Optimization (CSO) for task allocation and resource management in cloud environments. A.F.S. Devaraj et al. [19] propose a hybrid algorithm of firefly and Improved Multi-Objective Particle Swarm Optimization (IMPSO). The firefly algorithm is used to reduce the search space while the IMPSO identifies the enhanced response. The results are very promising.

Chukiat Worasucheep [20] introduces a hybrid ABC task scheduling technique with differential evolution. The outcome shows that it operates better than the original ABC and the original Opposition based Differential Evolution algorithm (ODE). FAN Chengli et al. [21] suggest a hybrid ABC technique with memory mechanism and fluctuating neighborhood search which seems to perform quite well for the benchmark functions they considered.

Nikhit Pawed et al. [22] suggested a mixture of ACO and ABC with dynamic feedback for resource usage. Its performance were excellent compared to existing ACO and ABC. Although their algorithm performed well, it was compared with only ABC and ACO.

M.A. Shahid et al. [23] suggest a novel technique that employs fault tolerance for task scheduling in the cloud computing environment after concluding that most load balancing techniques do not consider fault

tolerance issues. Nevertheless, their proposed algorithm is only theoretical and they do not offer any experimentation.

S. Velliangiri et al. [24] suggest a hybrid electro search with a genetic algorithm (GA) to enhance the behavior of load balancing techniques by considering variables such as makespan and resource utilization. Their algorithm exceeds current algorithms such as GA and ACO.

H. Yong et al. [25] introduce a chaotic algorithm with the artificial firefly technique and a task scheduling optimization strategy founded on that same algorithm. Their algorithm seems more suitable for solving large-scale task scheduling problems in cloud-fog network in comparison to other task scheduling algorithms.

The main contribution of this work will be optimum scheduling, load-balancing, servicing time and response time that leads to better resource utilization and a more stable system.

## 2.1 Background

In this section, the background of cloud computing technology related to this work is presented. However, there are several categories of load balancing. The focus is inclined to the category of this paper's augmented load-balancing algorithm.

### 2.1.1 Load-Balancing Algorithms Classification

As mentioned earlier, this work focuses primarily on system state-based load-balancing. The System state-based load-balancing techniques can be classified into two major groups, namely static and dynamic algorithms [14]. The most known ones are compared in Table 1.

**Table 1.** Main algorithms comparison

| Algorithms | Advantages | Disadvantages |
|---|---|---|
| Round Robin | Simple technique and the focus is on fairness.<br>Works in circular manner.<br>Fast when dealing with equal load distribution.<br>No starvation. | Not adaptive and scalable.<br>Some node maybe overloaded while others are idle.<br>Do not register the previous node state. |
| Min-Min | Fast and simple.<br>Made for small tasks. | Prioritize small tasks.<br>Larger tasks will wait in queue for a long time.<br>Does not consider the current node load. |
| Min-Max | Simple technique.<br>Smaller tasks are run concurrently. | Prioritize larger tasks.<br>Smaller tasks will wait longer.<br>Poor task scheduling. |
| ABC | Self-coordinating and inspired from nature.<br>Bigger system size will increase performance.<br>Made for heterogeneous environments. | Throughput does not increase with more resources. |
| Throttled Load-Balancing | Keeps a list of VMs and their state.<br>Satisfying performance. | Starts by scanning the entire list of VMs.<br>Current VM load does not matter. |
| ACO | Under loaded nodes are identified from the get go.<br>Not centralized. | Network overhead issues.<br>Delay when moving through the network. |
| PSO | Simple implementation.<br>Can be robust. | Memory.<br>Can converge prematurely. |

- **Static algorithms** are suitable for stable and homogenous environments. Static techniques are not adaptive. They do not check the state of the previous node while distributing tasks. Static load-balancing algorithms presents three main techniques: **round robin, min-min, min-max.**
- **Dynamic algorithms** are good for heterogeneous environment. They are flexible and tasks assignment is computed according to the actual state of the different nodes. It makes them more complex. They can be implemented in a distributed system or non-distributed ones. There are several dynamic algorithms out there. They are created and improved upon regularly such as: **ABC, ACO, PSO and throttled load-balancing.**

Hybrid algorithms that combine two or more algorithms from the same category or from the two different categories also exist [26]. They usually perform better due to the inheritance of the advantages of other involved algorithms.

### 2.1.2 Nature-Based Load-Balancing Techniques

In cloud computing, many of the techniques used for dynamic load-balancing are inspired from nature by studying the behavior of animals. Figure 1 presents some of the most known nature inspired task scheduling algorithms.

There are several nature-based techniques but this work focuses on two well-known swarm algorithms (ABC and PSO). These algorithms were chosen based on their working schemes and procedures that are functional and corresponds to how the (ALA) functions. The augmented load-balancing algorithm functions

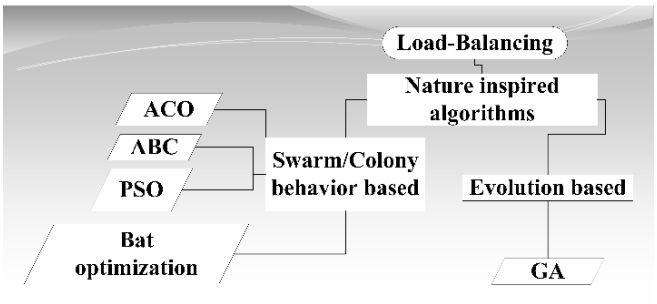



**Figure 1.** Nature inspired load-balancing techniques

better in terms of servicing time and improves load distribution among multiple VMs.

- **ABC (Artificial Bee Colony):** ABC is founded on the foraging system of honey bees' colony. It is composed of three distinct type of bees: scout bees, employed bees and onlooker bees [20]. In the initialization phase, the standard ABC algorithm randomly set up an original population of food sources. Let $B_i = \{b_i^1, b_i^2, ..., b_i^n\}$ represent the $i^{th}$ food source, at that point the location of food sources are computed. Each employed bee $X_i$ go to the different food sources in the same area, collect information and then return to the hive. That knowledge is shared with the onlooker bees by carrying out dances. Onlooker bees determine the food source relying upon that information and computing the odds [5]. Each food source $B_i$ is given a control criterion, which store the number of unsuccessful trials. If a food source cannot be enhanced within $trial_i$, $B_i$ will be forgotten and the employed bees transformed into scout bees [27]. At the same time, a new food source will be randomly produced [28]. The ABC algorithm has the advantages of doing exploitation and exploration and has a fast convergence and possesses few control parameters [29].
- **PSO (Particle Swarm Optimization):** PSO is a population-related optimization technique where the system is initiated along with a population of randomized particles. The algorithm then looks for the general best position by revising each particles best position [2]. Every particle in this algorithm symbolizes a bird matching a solution, and its fitness possesses a value measured by a fitness function [14]. Particles use velocity to help them navigate in the exploration zone [15]. Assume that the exploration area is *D*-dimensional. The location of the $i^{th}$ particle could be depicted by a *D*-dimensional vector $y_i = \{y_{i1}, ..., y_{iD}\}$, while the pace or velocity of the same particle could be defined as $V_i = \{v_{i1}, ..., v_{iD}\}$ [17]. Every particle looks for the highest relevant solution within the exploration area by revising its position along with its velocity records. The advantage of this algorithm is that it checks all the VMs to find the fittest. The fittest will be the one that wastes the least the memory [29].

#### 2.1.3 Problem Statement

With the large number of tasks that need to be scheduled in cloud environments, issues of deviation for VM response and task processing arise, impeding the effectiveness of cloud computing systems causing these systems to be overloaded, idle or under loaded. The inefficiency of these systems and their inability to respond and process client requests taunt its use. Several Load balancing techniques have been proposed in the literature to resolve these impending issues. These imbalances in response and servicing time of client request further ignites fluctuations issues in these systems amounting to huge differences between the average, minimum, maximum response and processing time. Furthermore, overloaded VMs cause incoming client requests to be queued for a longer period before being processed. Response time and processing time are significant issues of task scheduling in cloud computing that cannot be overlooked.

#### 2.1.4 Main Contribution

In this research, an augmented load-balancing algorithm is proposed for scheduling tasks in a cloud system. Hence, the contributions are as follows:

- Datacenters response time is used as the first factor to find the best datacenter. PSO particles are then used to find the best path from each region using how much time it would take to get there.
- ABC memory mechanism is used to memorize the best path to the chosen datacenter.
- The VMs/Host list is gotten from the concerned datacenter, which is used to schedule the incoming tasks evenly.
- The efficacy of the proposed algorithm is demonstrated as the response time and servicing time are enhanced with more balance and consistency in the system.

## 3 Proposed Augmented Load-Balancing Algorithm (ALA)

In this section, the goals of the proposed augmented algorithm and a detailed description of the algorithm are presented.

### 3.1 Methodology

This work approach task scheduling in a different manner. The tasks have to be scheduled in an even manner to keep the system balanced and improve the servicing time of different operations. Hence a combination of the PSO search system and ABC memory mechanism is used. The datacenters are

chosen as if they were a food source. To choose the best existing source, the response time of each datacenter is computed, the particles are then used to find the best route to get to that datacenter taking a lesser time. Once the path has been chosen the ABC memory mechanism is used while the route stays the same unless a datacenter with a better response time is found. At that point a list of VMs hosted by different physical machines in the datacenter is given. The list is used so that different tasks can be scheduled to different VMs and hosts evenly. The number of used VMs depends largely on the number of tasks received. The overview steps of the proposed algorithm are presented in Figure 2. Further details are given in the algorithm description.

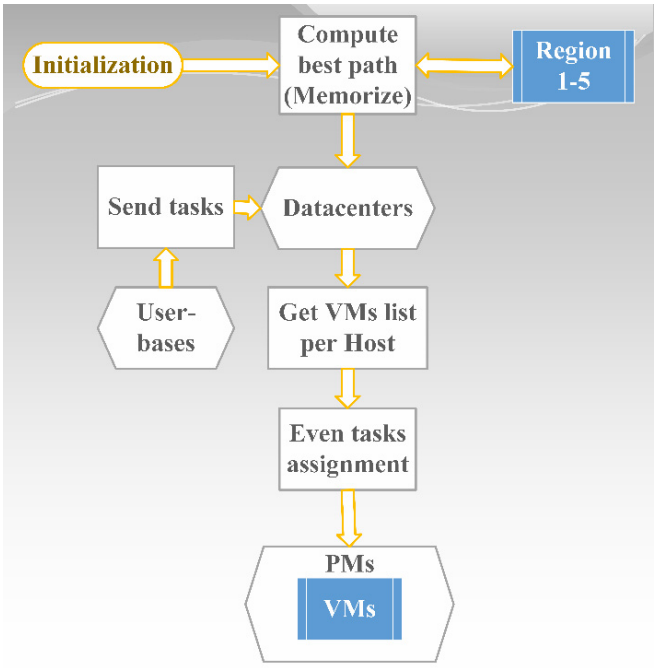



**Figure 2.** Algorithm overview steps

The methodology of this algorithm has been formulated in a manner to satisfy the following objectives:

- **Reduce response time:** the response time represents the time it takes for a user-base to send a request to a datacenter and receive a response. The response time should not change massively regardless of the location of the user-base.
- **Reduce processing time:** the processing time defines the time it takes for a VM to receive a task to the moment of its completion. The processing time of the same type of tasks should be similar.
- **Consistency:** consistency is defined as the system's ability to maintain almost the same computation time from light, medium to heavy load and not deviate but work the same way in different datacenters. To measure the system consistency the degree of imbalance and standard deviation that are well-known statistic measurements are used.

This research has some limitations. The users do not have different hardware or internet broadband. The tasks differ in size and not in composition. It is a scenario-based simulation and does not take into account the unpredictability of real life events.

### 3.1.1 Algorithm Description

The proposed augmented algorithm draws inspiration from the PSO search and ABC memory mechanisms and proposes a different method of scheduling tasks. The response time of each datacenter is added to enhance the search session. It is used as an indication of the destination for the search session. A well-defined position for each one and the local best position to get the best path to the selected datacenter is used. Every path taken will be memorized until a better one is found. The tasks are then distributed to the VMs in the datacenter in an even manner after the VMs list per host is received. A detailed pseudo-code of the algorithm is presented in Figure 3.

The algorithm has 4 phases as explained below.

- **First phase**

This is the initialization phase, every particle, VM, host, datacenter and user-base are initialized. All components of the system are preconfigured and settings does not change.

- **Second phase**

The response time of each data center is computed in order to determine the fastest depending on the different regions.

$$RT = TST_d - RRT_d \quad \textbf{(1)}$$

Where $RT$ represents the response time, RST is the request submission time, RRT is the request reaction time and $d$ represents the datacenter. The algorithm also checks the different attributes of the datacenters to measure their performances. If the response time achieved by a chosen datacenter deteriorate during the run of the algorithm, a better datacenter will be chosen by computing again the least response time for the concerned user-base or region.

- **Third phase**

Each employed bee is replaced by particles that will be in charge of finding the best path to a given datacenter. Their position and velocity are updated after each sequence to make sure that no better path has opened up. The following equations are used.

$$v_{i+1} = W * v_i + c_1 * rd1 * (Pb_i - x_i) + c_2 * rd2 * (Gb_i - x_i) \quad \textbf{(2)}$$

$$p_{i+1} = p_i + v_{i+1} \quad \textbf{(3)}$$

Where $v_i$ is used to obtain the velocity of $i^{th}$ particle, $p_i$ the position of that same particle and i is its index. The constant W is the inertia weight and is used to

**Input:** particles, VMs, datacenters, PMs, user bases and a set of tasks
**Output:** Scheduled tasks assigned to VMs

```
Begin
 Initialize particles (x1, x2... xn), datacenters (DC1-DC4), user bases (UB1-UB12), VMs
 (VM1, VM2... VMn)
 Compute DCs RT & attributes check
 For particles set
   Compute &
   Update particle positon
   Compute fitness value
   If ft  Pb > ft Pi  then
     Pb = Pi
     Swarm best Pi = Gb
     Local best Pi = lbest
     Set best path
     Memorize best path
   End If
   If curr_estimated_RT > least_estimated_RT then
       Least_RT = current_RT
     Switch DC
     Update best path
   End If
 End For
 For tasks set
     Get VMs/Host list
   If allocated_VM < VMs/Host list then
      Tasks sets  à  unallocated_VM/Host
          Assign tasks to VMs/Host evenly
   Else
     For VMid : allocation_Count
      Curr_count = allocation_Count
      If Curr_count < min_count then
        min_count = curr_count
       Tasks  à  VMid/Hostid
      End If
     End For
   End If
 End For
End
```

**Figure 3.** Algorithm pseudo-code

balance the local and global search capabilities. c1 and c2 are the coefficients of acceleration that each particle can take per iteration. rd1, rd2 represent two random figures that reside between (0, 1). Pb is used as the best position of the said particle while Gb is the best position of the entire swarm. The finest position in a neighborhood is the local best position (lbest) for a given region, which leads to the best datacenter available for that region. $x_i$ is the current particle position. The particles best position and local best position are updated if necessary using a fitness function (ft). The fittest particles indicate the best path to a given datacenter. The local best position that leads to the best path to a given datacenter is memorized using onlooker bees memory mechanism and can only change if a better path is found.

$$ft(x) = x_{tn} - x_{t0} \quad \textbf{(4)}$$

Where $x_{t0}$ represents the departure time of an x particle and $x_{tn}$ represents the arrival time.

$$if \ \ ft(Pb) < ft(lbest) \rightarrow lbest = Pb \quad \textbf{(5)}$$

$$if \ \ ft(p_i) < ft(Pb) \rightarrow Pb = p_i \quad \textbf{(6)}$$

Different user-bases from the same region or a single user-base might send tasks to different datacenters simultaneously if the two datacenters have quick response time and both paths are just about alike.

- **Fourth phase**

At the datacenter level, the list of VMs hosted by the physical machines (hosts) is produced allowing the number of incoming tasks to be evenly allocated. A quick operation is made to ensure an even distribution

of tasks.

$$tasks \ mod \ (VMs \mid Hosts) = 0 \quad \textbf{(7)}$$

If the result is not 0 the remaining task(s) are given to the first host with a VM with the least allocations. If there are other incoming tasks, they will also be allocated to VMs with less task's allocations than the others. If all available VMs are not allocated, then the new tasks are allocated to those. After completion of a task, the status of concerned VMs/Host is updated. A complete system check is done for more user requests. If there is, it then returns to the second phase and continues the process. If all the requests have been processed then it terminates.

The next section talks about the implementation and test of ALA compared to different other algorithms, the simulation setup, the libraries used and the different scenarios and why they were chosen. Finally, the results and their corresponding analysis are presented.

## 4 Performance Evaluation

### 4.1 Simulation Setup

To assess the performance of the proposed augmented algorithm (ALA), a simulation model is created using the CloudSim and CloudAnalyst libraries (CloudSim based library). The library is implemented in eclipse with JDK 8 on a computer with windows 10. The computer runs on a RAM of 16 GB with a six core processor Intel i7-8750H at a base clock speed of 2.20 GHz. To validate the performances of the algorithm, a simulation of 1 hour (1H), 1 day (1D) and 3 days (3D) of continuous cloud task scheduling is conducted. One hour is chosen as a minimum and it is noted that at that point lightweight load is handled. One day of continuous tasks scheduling represents a medium load. Three days were chosen to ascertain a change in the load handling after a couple of days when the load is heavy.

CloudSim is one of the best tools to implement the provided techniques and can be extended quite easily with programming knowledge. It is an expandable simulation library that enables simulation and modeling of Cloud computing applications and systems. A system model is designed based on the libraries of CloudSim to enable experimentation.

This design is inspired by the work of authors in [8], [10], and [22] in cloud computing which was used to design this system. Figure 4 shows a detailed map of the simulation model. There are 12 user-bases (UB) and four datacenters (DC). UB1 and UB12 are in North America with DC1 (Region 0), UB11 and UB2 are in South America with DC2 (Region 1), UB10 and UB3 are in Europe (Region 2), Ub9 and UB4 are in Asia with DC3 (Region 3), UB8 and UB5 are in Africa with DC4 (Region 4), and finally UB6 and Ub7 are in Oceania (Region 5). Each datacenter has several physical machines (PMs) which support the different VMs.

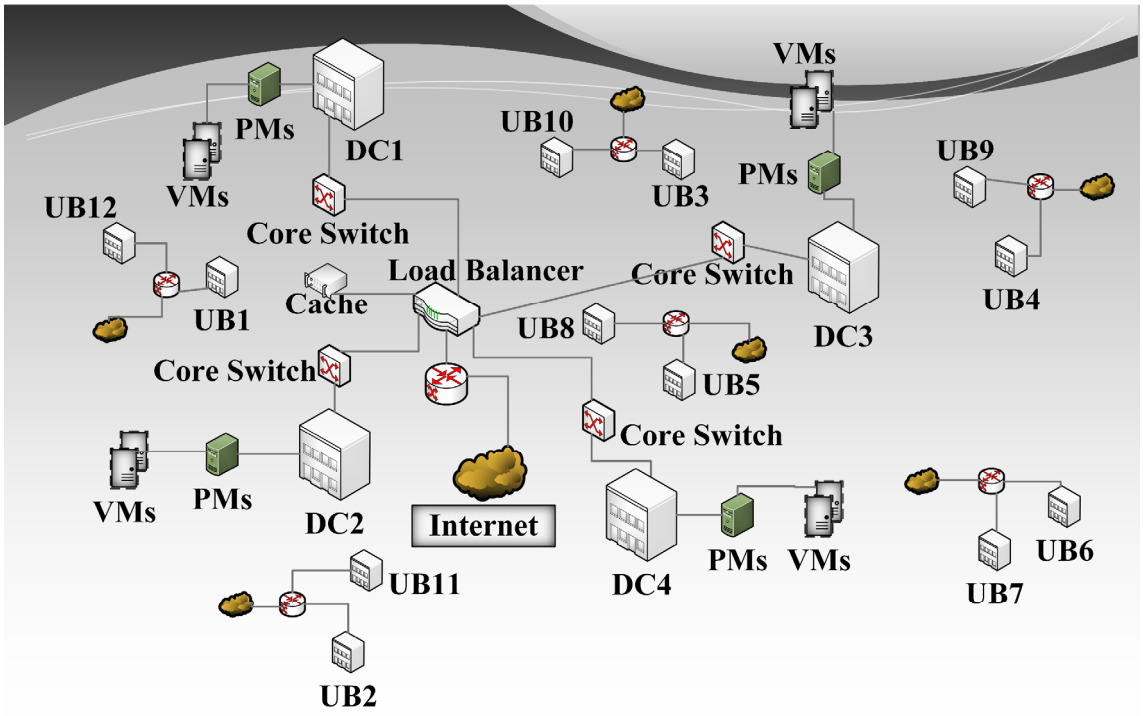



**Figure 4.** Simulation model

As in Figure 4, the system comprises three main parts: *the user-bases, the load balancer and the datacenters.*

- **The user-bases** are composed of several users to generate requests continuously to complete the simulation runtime. Their requests are sent to the internet via the nearest router. Each user-base has 10 simultaneous users.
- **The load balancer** receives the incoming requests from different user-bases via the router connecting the load balancer to the internet. It's responsible for distributing the afore-mentioned tasks among the several datacenters and also makes sure of system stability and balance. Load balancing algorithms with memory options such as the best PM, VM, and most requested task, will use the cache to store that information so they can be accessed quickly during the next iteration.
- **The datacenters** are in charge of processing every single task which comes through the core switch. The PMs are responsible of hosting the different VMs and will be used to process the different tasks are located there. Each VM in the system can handle a threshold of 10 simultaneous tasks.

Table 2 shows the datacenters different properties such as the Operating System (OS), Virtual Machine Manager (VMM), number of VMs (N° VMs), the bandwidth (BW), the number of physical machines (PMs) and so on. It also indicates that the four datacenters do not have the same capabilities and cannot handle the same number of tasks.

**Table 2.** Datacenters configuration

| Datacenters | DC1 | DC2 | DC3 | DC4 |
|---|---|---|---|---|
| Region ID | 0 | 1 | 3 | 4 |
| OS | Linux | Linux | Linux | Linux |
| VMM | Xen | Xen | Xen | Xen |
| BW (Mbit/s) | 3000 | 2000 | 4000 | 3500 |
| N° PMs | 15 | 5 | 17 | 12 |
| N° VMs | 30 | 10 | 35 | 25 |
| Memory (GB) | 32 | 24 | 32 | 16 |
| Storage (TB) | 6 | 4 | 5 | 6 |
| N° Processor/PM | 4 | 4 | 6 | 5 |
| Processors Speed (GHz) | 3 | 2.6 | 2.5 | 2.7 |
| VM Policy | Time Shared | Time Shared | Time Shared | Time Shared |

In Table 3, the user-bases general configurations were used. Each user-base has a different number of users' requests they can send. At the end of an hour of continuous task scheduling a total of 421 563 tasks are handled, after one day it is 15 220 871 tasks and finally after three days it is 35 373 935 tasks. The system will have to handle a continually increasing number of tasks and it will be the responsibility of the chosen balancing techniques to correctly schedule them.

In Table 3, Avg USR/HR represents the average number of users per hour, RQST/USR/HR represents the number of requests per user per hour and DT SZE/RQST represents data size per request.

**Table 3.** User-Bases configuration

| User-Base | Region | Avg USR/HR | RQST /USR/HR | DT SZ/RQST (KB) |
|---|---|---|---|---|
| UB1 | 0 | 200 | 360 | 1000 |
| UB2 | 1 | 100 | 260 | 500 |
| UB3 | 2 | 150 | 200 | 1000 |
| UB4 | 3 | 300 | 400 | 2000 |
| UB5 | 4 | 100 | 100 | 1500 |
| UB6 | 5 | 50 | 50 | 300 |
| UB7 | 5 | 70 | 350 | 400 |
| UB8 | 4 | 150 | 300 | 2000 |
| UB9 | 3 | 250 | 60 | 3000 |
| UB10 | 2 | 170 | 160 | 1800 |
| UB11 | 1 | 110 | 230 | 2300 |
| UB12 | 0 | 190 | 120 | 2200 |

## 4.2 Results and Discussion

To decide the efficiency of the proposed algorithm, analysis of several effects from the response time, the processing time to the resources usage, degree of imbalance and standard deviation are considered. A comparison of ALA against a set of algorithms namely ABC, PSO, TLB (threshold load balancer), ACO, and SJF (short job first) was made and results were analyzed.

### 4.2.1 Response Time Analysis

As mentioned before, the response time is when it takes for a user-base to send a request and the datacenter to respond. In the first experiment, different algorithms' response time was computed and results were given in milliseconds. Figure 5 illustrates the minimum (Min), maximum (Max), and average (Avg) overall response time after simulating one hour, one day and three days of continuous task scheduling.

In Figure 5, the response time of the different algorithms was compared. It can be seen that, the proposed augmented algorithm (ALA) performs better and has a low minimum, maximum and average response time. ACO and TLB average response time are closest to that of ALA but the ACO maximum time increase when it gets to one day and three days. This increase results from ACO having issues when moving through the network. If the load becomes increasingly heavy, it causes the response time to increase in some cases. The maximum response times for ABC and PSO are high throughout the experiments. Their average is low but does not defeat ALA's. The SJF algorithm did better with the maximum response time for one day and three days, but ALA maintained a better average response time. The minimum response time of the algorithms is low and the differences are small.

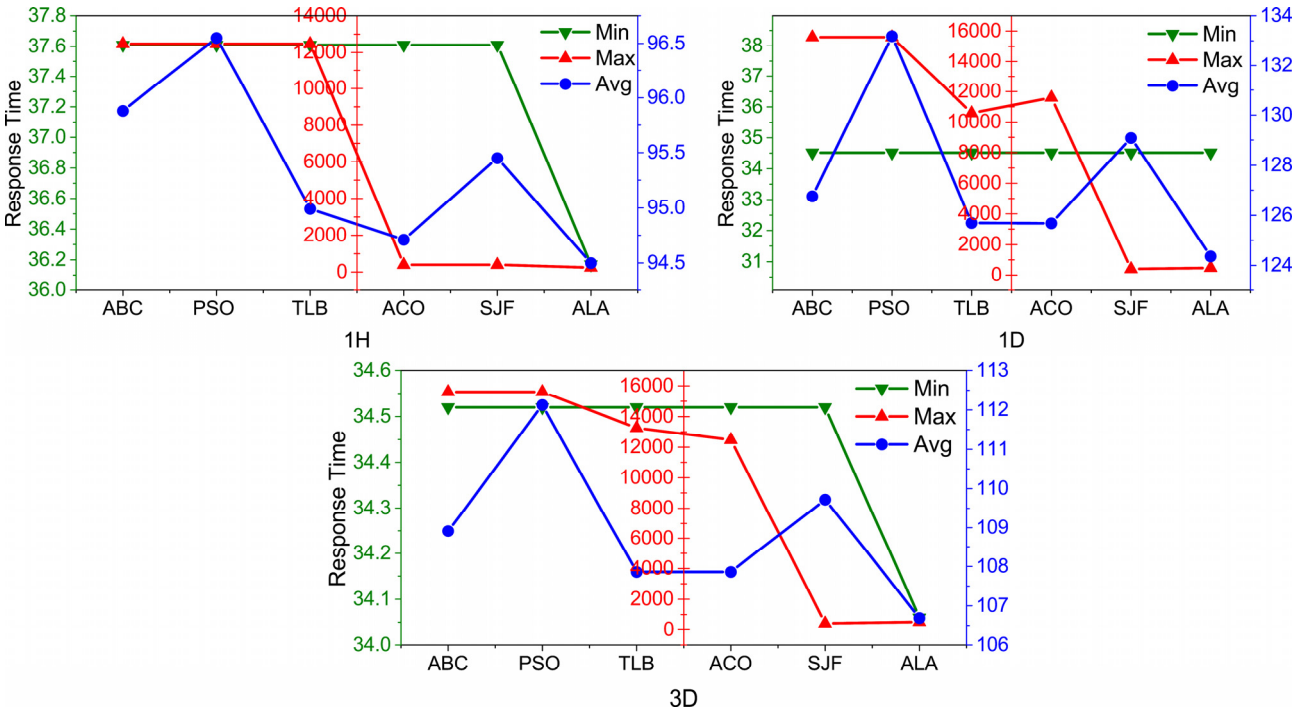



**Figure 5.** Minimum maximum and average overall response time (ms) after (a) 1 Hour (b) 1 Day and (c) 3 Days of continuous task scheduling

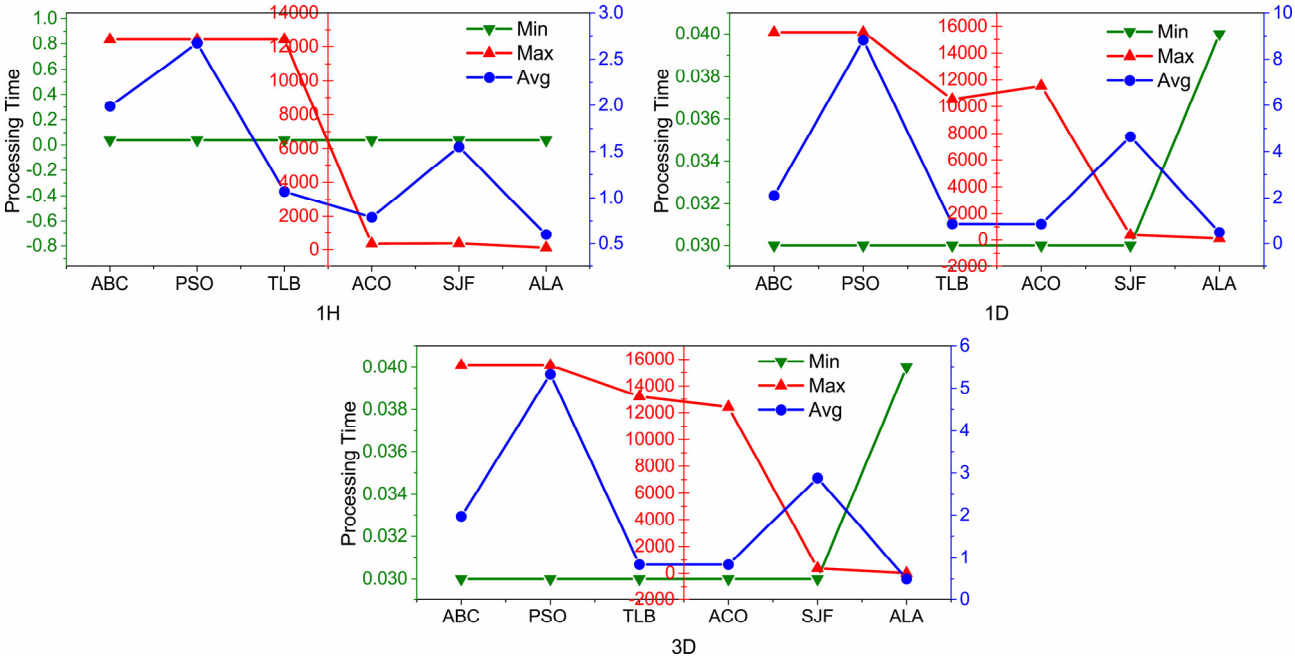



**Figure 6.** Minimum maximum and average overall processing time (ms) after (a) 1 Hour (b) 1 Day and (c) 3 Days of continuous task scheduling

### 4.2.2 Processing Time Analysis

In this section, the processing time in milliseconds of the algorithms is the time it takes for each task to be completed. In Figure 6, the minimum, maximum and average (Avg) servicing time of the system and then in Figure 7 the average processing time achieved by the four datacenters are presented.

In Figure 6, it can be concluded that ALA performs better than the others. However, the minimum processing time is higher than the rest for one day and three days of experiments. It results from the reduction in the minimum for the other algorithms while ALA minimum stays the same. The average processing time of the proposed algorithm is better than the ACO, TLB and other algorithms compared, as shown in Figure 7. The ACO maximum processing time increases after one day due to network overhead that the algorithm suffers from. The proposed algorithm and SJF managed a low maximum time but the proposed

algorithm performed much better. The proposed algorithm (ALA) maintained a low minimum, maximum and average processing time compared to the rest and kept those times consistent throughout the experiments.

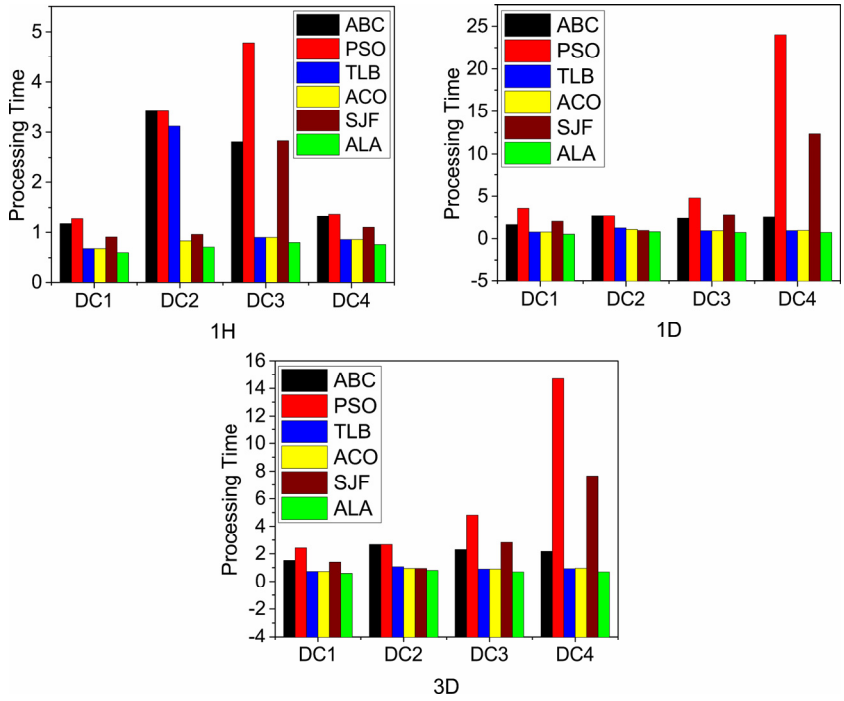



**Figure 7.** Average processing time (ms) of each datacenter after (a) 1 Hour (b) 1 Day and (c) 3 Days of continuous task scheduling

### 4.2.3 System Consistency Analysis

**· Resource Utilization**

Throughout the experiments, the algorithms were expected to behave differently. Therefore, in Figure 8 and Figure 9 an analysis and comparison of how many VMs have been used after each experiment by the algorithms are conducted. The number of tasks that was assigned to each datacenter is also analyzed.

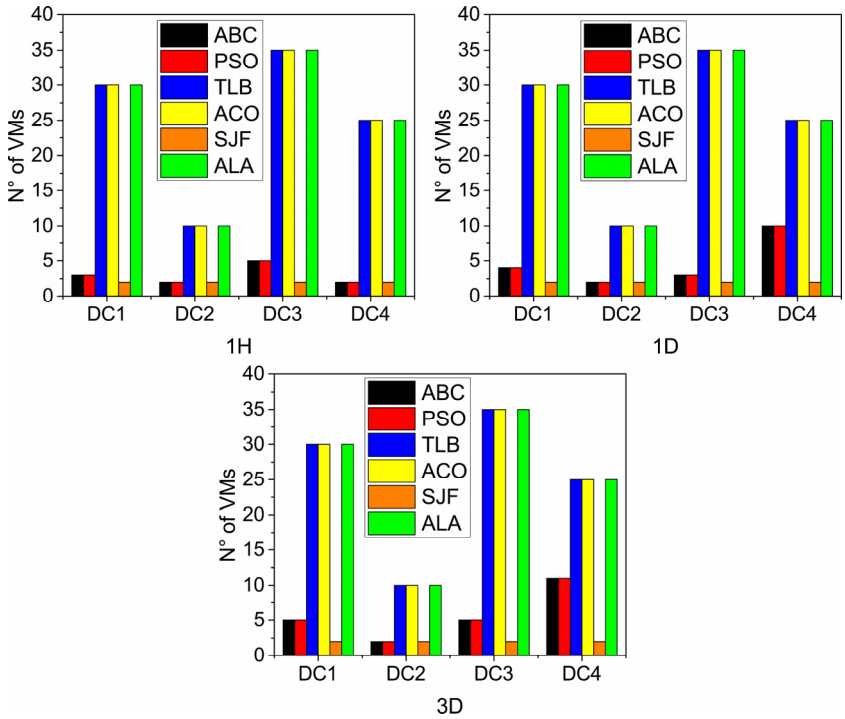



**Figure 8.** Number of VMs used at each datacenter after (a) 1 Hour (b) 1 Day and (c) 3 Days of constant task scheduling

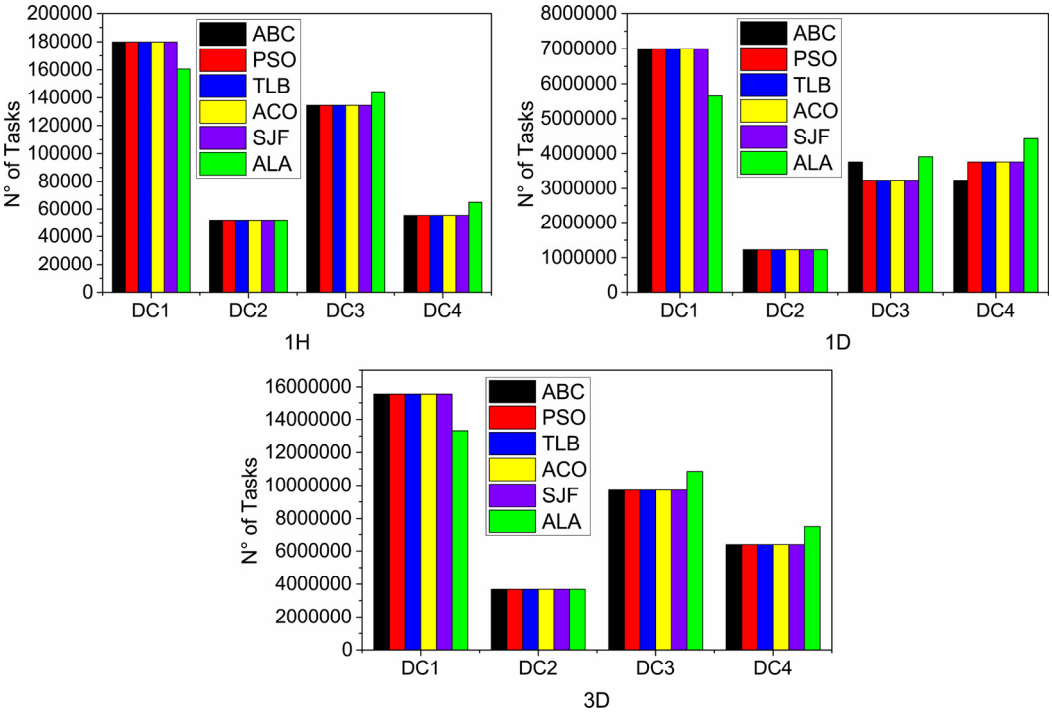



**Figure 9.** Number of tasks handled by each datacenter after
(a) 1 Hour (b) 1 Day and (c) 3 Days of constant task scheduling

It is clear in Figure 8 that not all algorithms work the same way and use the same number of VMs. For ABC, though it chooses a food source (VM) and only changes it if it deteriorates, it leads to the usage of a limited number of VMs. A new VM will be used only if the one at work is busy or has a problem. The same issues occur with the PSO algorithm. Although it checks every VM, it only allocates tasks to the number of VMs which are considered as being the solution. Since allocating tasks evenly among the VMs is the goal, the proposed algorithm will use all the VMs at its disposition to enhance the processing time and keep the maximum processing time low. The ACO algorithm uses also all the available VMs since the ants will park the whole network allocating and re-allocating tasks but is slowed down by the backward and forward movements of the ants. For the SJF algorithm, since it deals with short jobs first, it means it may not have the opportunity to use all of the available VMs. In the end, only TLB, ACO and the proposed algorithm (ALA) managed to use all the available VMs.

In Figure 9, it can be seen that none of the algorithms distribute the load equally to the different datacenters and that would be an unrealistic goal since they are dispatched to different regions and have different configurations. However, it can also be seen that ALA reduces the load on the DC1 and share it with DC3 and DC4 the two next most capable DCs. Where a task goes not only depend on the performance of the DC, the region UB it's in but also the response time of said DC. Considering all those factors it can be said that our algorithm distributes the load in a more efficient way.

- **Degree of Imbalance**

Figure 10 shows the Degree of Imbalance (DI) for the different techniques. The degree of imbalance computes the unevenness amongst the multiple VMs in a given datacenter. The smaller the measurement of the DI the better because it indicates that the load of the system is better balanced and that the scheduler is more efficient.

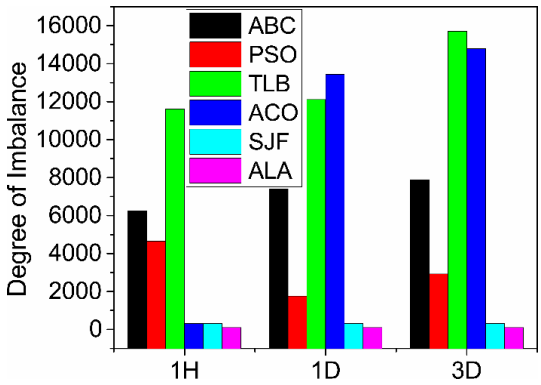



**Figure 10.** Overall degree of imbalance after 1 Hour 1 Day and 3 Days of constant task scheduling

$$DI = \{(pt_{max} - pt_{min}) / pt_{avg}\} \quad \textbf{(8)}$$

Where $pt_{max}$, $pt_{min}$, and $pt_{avg}$ are the maximum, minimum and average processing time of VMs in datacenter respectively.

Due to the process of scheduling incoming tasks evenly among the available VMs and choosing the best datacenter for each region, ALA produces a low degree of imbalance not only in the overall system but also with each data center and does so with the low,

medium and a large number of tasks, like it can be seen in Figure 10 and Figure 11. The ABC, PSO and TLB algorithms produce a fluctuating degree of imbalance, as shown in Figure 11. The difference is most likely due to the amount of load that is being handle and the fact that they do not use all the available VMs, leading to some overloading and an increase in processing time. For ACO, it is about the same issue but the network overhead it is known for does not help. For TLB the fact that it has high maximum processing time means it will not have a good degree of imbalance. The SJF algorithm managed the second best degree of imbalance throughout the experiments. Some datacenters will still be able to handle any load depending on their performances but the efficacy of each algorithm will stay very important.

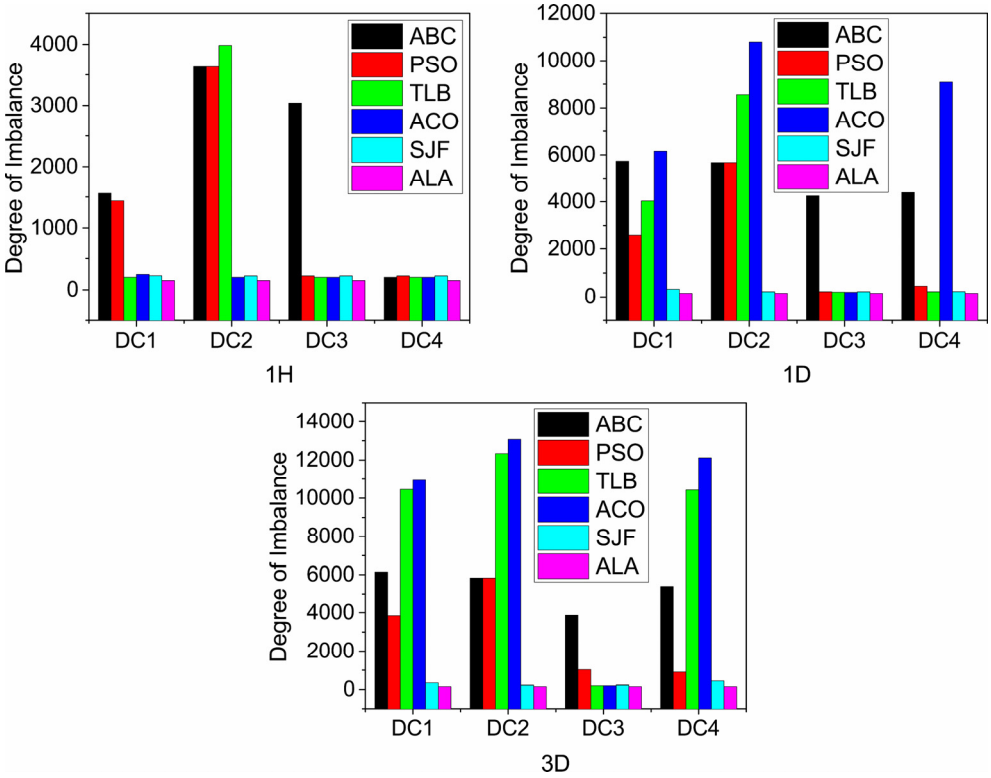



**Figure 11.** Degree of imbalance of each datacenter after (a) 1 Hour (b) 1 Day and (c) 3 Days of constant task scheduling

- **Standard Deviation**

In Figure 12, a computation of standard deviation is presented. The standard deviation is used to compute the degree of variation or scattering of an array of values. A small standard deviation demonstrates that the figures are inclined to gravitate around the value of the mean of a given set. But on the other hand, a big standard deviation demonstrates that the figures are stretched out upon a broader range.

$$\sigma = \sqrt{\Sigma(x_i - \mu)2^2 / N} \quad \textbf{(9)}$$

Where N represents the number of datacenters, $x_i$ each datacenter value, and $\mu$ the mean value.

The standard deviation of ALA is not only low but also better than the other algorithms. ACO manages the second best values which proves that although its maximum processing time increases, most values are close to the mean. It is the same thing for the TLB algorithm although it is slightly higher than the ACO one. ABC although not that low still performs well. On the other hand, PSO has a bad standard deviation and the cause of that is most certainly the fact that the algorithm can sometimes fall into false solutions leading to an increase in processing time and the scattering of the values. SJF has the second worst standard deviation, most likely due to how it treats different size tasks. It can then be concluded that ALA achieves better performances in this given situation as seen in Figure 12.

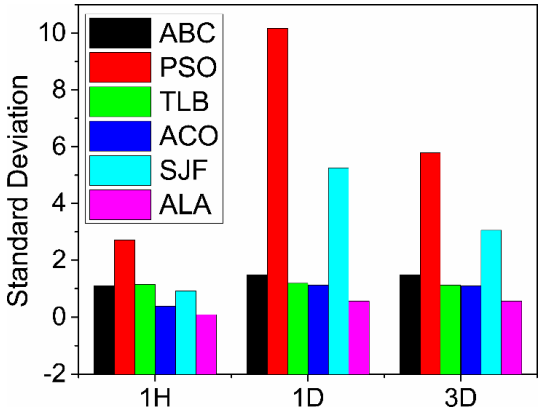



**Figure 12.** Standard deviation after 1 Hour 1 Day and 3 Days of constant task scheduling

### 4.2.4 Discussion

The augmented load balancing algorithm (ALA) for task scheduling inspired by the PSO search system and ABC memory mechanism has proven efficient. It added the response time in the search criteria to find the most suitable datacenter for each region and the shortest path to get there improving the requests response time. Generating a list of VMs per host from the chosen datacenter and scheduling tasks evenly enhanced the processing time and system consistency. In the experiments above many factors from the response time to the system consistency using a different number of tasks sent constantly for 1 hour, one day and three days have been evaluated. However, it was not perfect especially with the maximum response time. The SJF algorithm did better with the maximum response time for one day and three days. However, ALA performed better than the other techniques in the different experiments presented. Therefore, based on the experiments, it can be inferred that ALA takes up the challenge regarding the evaluated areas.

## 5 Conclusion

With the quick rise of cloud computing, several algorithms have been suggested to tackle the scheduling of the continually rising number of tasks. In this paper, an augmented load balancing algorithm (ALA) has been developed for task scheduling and resource optimization purposes. The algorithm is inspired by two major swarm algorithms namely ABC and PSO. It takes advantage of some of their strong points and adds a response time criterion in search of a suitable datacenter for each region and the best path to get there. It also schedules tasks evenly among the available VMs per host making sure that no VM is busy while others are providing little to no work at all. Experiments are conducted for one hour, one day and three days to assess the performance of the algorithm and are performed on a specially designed simulation model. The final results clearly show that not only the maximum response time is reduced to get closer to the average time, but also that the different tasks are processed quicker. The consistency of the algorithm is also tested by evaluating its resource utilization, how many tasks are scheduled towards each datacenter and also how balanced the load is. The final important point is the standard deviation that demonstrates that the different processing time have an inclination to gravitate around the value of the mean which means that it only slightly deviate from it. The augmented load balancing algorithm (ALA) not only produce a better scheduling scheme but also a more stable one which is balanced all around.

## Acknowledgements

We would like to thank Dengzhi Liu, Kondwani Michael Kamoto, Francis Mawuli Nakoty and Athanase Nkuzimana; all postgraduate students at Nanjing University of Information Science and Technology for all their useful insights and help during the different stages of this work.

## References

[1] M. Mesbahi, A. M. Rahmani, Load Balancing in Cloud Computing: A State of the Art Survey, *International Journal of Modern Education and Computer Science*, Vol. 8, No. 3, pp. 64-78, March, 2016.

[2] M. Hamdan, E. Hassan, A. Abdelaziz, A. Elhigazi, B. Mohammed, S. Khan, A. V. Vasilakos, M. N. Marsono, A comprehensive survey of load balancing techniques in software-defined network, *Journal of Network and Computer Applications*, Vol. 174, Article No. 102856, January, 2021.

[3] M. Ala'anzy, M. Othman, Load Balancing and Server Consolidation in Cloud Computing Environments: A Meta-Study, *IEEE Access*, Vol. 7, pp. 141868-141887, September, 2019.

[4] S. K. Mishra, B. Sahoo, P. P. Parida, Load balancing in cloud computing: A big picture, *Journal of King Saud University – Computer and Information Science*, Vol. 32, No. 2, pp. 149-158, February, 2020.

[5] R. S. Sajjan, R. Y. Biradar, Load Balancing and its Algorithms in Cloud Computing: A Survey, *International Journal of Computer Sciences and Engineering*, Vol. 5, No. 1, pp. 95-100, January, 2017.

[6] G. Muthusamy, S. R. Chandran, Cluster-based Task Scheduling Using K-Means Clustering for Load Balancing in Cloud Datacenters, *Journal of Internet Technology*, Vol. 22, No. 1, pp. 121-130, January, 2021.

[7] A. A. Nasr, N. A. El-Bahnasawy, G. Attiya, A. El-Sayed, Cloudlet Scheduling Based Load Balancing on Virtual Machines in Cloud Computing Environment, *Journal of Internet Technology*, Vol. 20, No. 5, pp. 1371-1378, September, 2019.

[8] E. Pacini, C. Mateos, C. García Garino, Balancing throughput and response time in online scientific Clouds via Ant Colony Optimization, *Advances in Engineering Software*, Vol. 84, pp. 31-47, June, 2015.

[9] M. Junaid, A. Sohail, A. Ahmed, A. Baz, I. A. Khan, H. Alhakami, A Hybrid Model for Load Balancing in Cloud Using File Type Formatting, *IEEE Access*, Vol. 8, pp. 118135-118155, June, 2020.

[10] W. Hashem, H. Nashaat, R. Rizk, Honey bee based load balancing in cloud computing, *KSII Transactions on Internet and Information Systems*, Vol. 11, No. 12, pp. 5694-5711, December, 2017.

[11] L. D. D. Babu, P. V. Krishna, Honey bee behavior inspired load balancing of tasks in cloud computing environments, *Applied Soft Computing*, Vol. 13, No. 5, pp. 2292-2303, May,

2013.

[12] L. Shen, J. Li, Y. Wu, Z. Tang, Y. Wang, Optimization of Artificial Bee Colony Algorithm Based Load Balancing in Smart Grid Cloud, *2019 IEEE PES Innovative Smart Grid Technologies-Asia (ISGT 2019)*, Chengdu, China, 2019, pp. 1131-1134.

[13] A. Dave, B. Patel, G. Bhatt, Y. Vora, Load balancing in cloud computing using particle swarm optimization on Xen Server, *2017 Nirma University International Conference on Engineering (NUiCONE 2017)*, Gujarat, India, 2017, pp. 1-6.

[14] J. Acharya, M. Mehta, B. Saini, Particle swarm optimization based load balancing in cloud computing, *International Conference on Communication and Electronics Systems (ICCES 2016)*, Coimbatore, India, 2016, pp. 1-4.

[15] T. T. Khuat, M. H. Le, A Novel Hybrid ABC-PSO Algorithm for Effort Estimation of Software Projects Using Agile Methodologies, *Journal of Intelligent Systems*, Vol. 27, No. 3, pp. 489-506, 2018.

[16] A. K. Joseph, G. Radhamani, A Hybrid Model of Particle Swarm Optimization ( PSO ) and Artificial Bee Colony (ABC) Algorithm for Test Case Optimization, https://www.semanticscholar.org/paper/A-Hybrid-Model-of-Particle-Swarm-Optimization-(-PSO-Joseph-Radhamani/116f77d76ede5f3e4c3bde92e9aa22356cd70aa1, 2013.

[17] N. Baktash, M. R. Meybodi, A New Hybrid Model of PSO and ABC Algorithms for Optimization in Dynamic Environment, *International Journal of Computer Theory and Engineering*, Vol. 4, No. 3, pp. 362-364, June, 2012.

[18] S. G. Domanal, R. M. R. Guddeti, R. Buyya, A Hybrid Bio-Inspired Algorithm for Scheduling and Resource Management in Cloud Environment, *IEEE Transactions on Services Computing*, Vol. 13, No. 1, pp. 3-15, January-February, 2020.

[19] A. F. S. Devaraj, M. Elhoseny, S. Dhanasekaran, E. L. Lydia, K. Shankar, Hybridization of firefly and Improved Multi-Objective Particle Swarm Optimization algorithm for energy efficient load balancing in Cloud Computing environments, *Journal of Parallel and Distributed Computing*, Vol. 142, pp. 36-45, August, 2020.

[20] C. Worasucheep, A Hybrid Artificial Bee Colony with Differential Evolution, *International Journal of Machine Learning and Computing*, Vol. 5, No. 3, pp. 179-186, June, 2015.

[21] C. Fan, Q. Fu, G. Long, Q. Xing, Hybrid artificial bee colony algorithm with variable neighborhood search and memory mechanism, *Journal of Systems Engineering and Electronics*, Vol. 29, No. 2, pp. 405-414, April, 2018.

[22] N. Pawar, U. K. Lilhore, N. Agrawal, A Hybrid ACHBDF Load Balancing Method for Optimum Resource Utilization In Cloud Computing, *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, Vol. 2, No. 6, pp. 367-373, November-December, 2017.

[23] M. A. Shahid, N. Islam, M. M. Alam, M. M. Su'ud, S. Musa, A Comprehensive Study of Load Balancing Approaches in the Cloud Computing Environment and a Novel Fault Tolerance Approach, *IEEE Access*, Vol. 8, pp. 130500-130526, July, 2020.

[24] S. Velliangiri, P. Karthikeyan, V. M. Arul Xavier, D. Baswaraj, Hybrid electro search with genetic algorithm for task scheduling in cloud computing, *Ain Shams Engineering Journal*, Vol. 12, No. 1, pp. 631-639, March, 2021.

[25] H. Yong, Load balancing strategy for medical big data based on low delay cloud network, *Journal of Engineering*, Vol. 2020, No. 9, pp. 799-804, September, 2020.

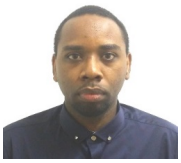
[26] C. Chukwuneke, H. Inyiama, S. Amaefule, M. Onyesolu, D. C. Asogwa, Review of Hybrid Load Balancing Algorithms in Cloud Computing Environment, *International Journal of Trend in Research and Development*, Vol. 6, No. 6, pp. 31-37, December, 2019.

[27] A. Ullah, N. M. Nawi, J. Uddin, S. Baseer, A. H. Rashed, Artificial bee colony algorithm used for load balancing in cloud computing: Review, *International Journal of Artificial Intelligence*, Vol. 8, No. 2, pp. 156-167, June, 2019.

[28] F. S. Abu-Mouti, M. E. El-Hawary, Overview of Artificial Bee Colony (ABC) algorithm and its applications, *2012 IEEE International Systems Conference (SysCon 2012)*, Vancouver, Canada, 2012, pp. 590-595.

[29] M. Rana, S. Bilgaiyan, U. Kar, A study on load balancing in cloud computing environment using evolutionary and swarm based algorithms, *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT 2014)*, Kanyakumari District, India, 2014, pp. 245-250.

## Biographies



**Franck Seigneur Nininahazwe** received the M.Sc. degree from the Nanjing University of Information Science and Technology, Nanjing, China, in 2018. He is currently working towards the Ph.D. degree with the School of Computer and Software, Nanjing University of Information Science and Technology. His research focuses on task scheduling and load-balancing in cloud computing systems.

**Jian Shen** received the M.E. and Ph.D. degrees in computer science from Chosun University, Gwangju, South Korea, in 2009 and 2012, respectively. Since late 2012, he has been a Professor with the Nanjing University of Information Science and Technology, Nanjing, China. His research interests include public cryptography, cloud computing and security, and information security systems.

**Micheal Ernest Taylor** received his M.Sc. degree from the Nanjing University of Information Science and Technology, Nanjing, China, in 2018. He is currently working towards his Ph.D. with the School of Computer and Software, Nanjing University of Information Science and Technology. His research focuses on resource management and load-balancing in cloud computing systems.