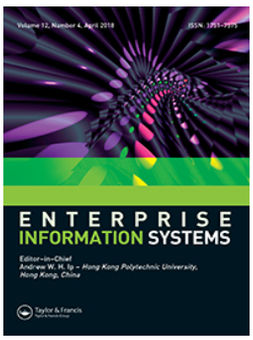


Enterprise Information Systems

ISSN: 1751-7575 (Print) 1751-7583 (Online) Journal homepage: http://www.tandfonline.com/loi/teis20

Fog computing job scheduling optimization based on bees swarm

Salim Bitam, Sherali Zeadally & Abdelhamid Mellouk

To cite this article: Salim Bitam, Sherali Zeadally & Abdelhamid Mellouk (2018) Fog computing job scheduling optimization based on bees swarm, Enterprise Information Systems, 12:4, 373-397, DOI: 10.1080/17517575.2017.1304579

To link to this article: https://doi.org/10.1080/17517575.2017.1304579

ENTERPRISE INFORMATION SYSTEMS, 2018
VOL. 12, NO. 4, 373–397
https://doi.org/10.1080/17517575.2017.1304579

Taylor & Francis
Taylor & Francis Group

Fog computing job scheduling optimization based on bees swarm

Salim Bitam[a], Sherali Zeadally[b] and Abdelhamid Mellouk[c]

[a]LESIA Laboratory, Department of Computer Science, University of Biskra, Biskra, Algeria; [b]School of Information Science, College of Communication and Information, University of Kentucky, Lexington, Kentucky, USA; [c]Image, Signal and Intelligent Systems Laboratory-LISSI, Department of Networks and Telecoms, IUT C/V, University of Paris-Est Creteil (UPEC), Vitry sur Seine, France

ABSTRACT

Fog computing is a new computing architecture, composed of a set of near-user edge devices called fog nodes, which collaborate together in order to perform computational services such as running applications, storing an important amount of data, and transmitting messages. Fog computing extends cloud computing by deploying digital resources at the premise of mobile users. In this new paradigm, management and operating functions, such as job scheduling aim at providing high-performance, cost-effective services requested by mobile users and executed by fog nodes. We propose a new bio-inspired optimization approach called Bees Life Algorithm (BLA) aimed at addressing the job scheduling problem in the fog computing environment. Our proposed approach is based on the optimized distribution of a set of tasks among all the fog computing nodes. The objective is to find an optimal tradeoff between CPU execution time and allocated memory required by fog computing services established by mobile users. Our empirical performance evaluation results demonstrate that the proposal outperforms the traditional particle swarm optimization and genetic algorithm in terms of CPU execution time and allocated memory.

ARTICLE HISTORY
Received 5 October 2016
Accepted 5 March 2017

KEYWORDS
Fog computing; edge computing; job scheduling; bees life algorithm; CPU execution time; allocated memory

1. Introduction

Recently, academia and industry have been developing efficient architectures to ensure the ubiquitous connectivity of smart devices. These smart devices are empowering mobile users by providing them an access to a range of high performance and cost-effective services in different types of environments including smart homes, smart cities, smart metering connected vehicles, large-scale wireless sensor networks, etc. In parallel, the wide deployment of these smart devices along with efficient communication and processing technologies has led to a new paradigm called the Internet of Things (IoT). However, several aspects such as limited computing, processing, and networking capabilities of IoT devices make them unsuitable for executing complex, processor or memory intensive applications. To address these constraints, many IoT solutions have been leveraging cloud computing technologies (Díaz, Martín, and Rubio 2016). Cloud computing dynamically provides resources to IoT applications by using scalable and virtualized resources from a pool of efficient computing, storage, and communication devices located in distributed data centers sharing by several end users.

CONTACT Salim Bitam s.bitam@univ-biskra.dz LESIA Laboratory, Department of Computer Science, University of Biskra, Biskra, Algeria

Today, several research efforts have been exploring the integration of IoT with cloud computing (Guerrero-ibanez, Zeadally, and Contreras-Castillo 2015)(Botta et al. 2014). Despite these integration efforts, there are still many unsolved IoT application issues and mobile user services regarding their requirements of low latency, mobility support, geo-distribution and location-based information. Since these services cannot be efficiently supported, we need an alternative technology to bring computational services to more smart devices that are located geographically near to the end user than to the cloud data centers. Such devices reside at the edge of the cloud network and they can sense the environment, store data, execute tasks, provide services and transmit information to the cloud cores (i.e. data centers) for further analysis. To be able to access these types of services, a new computational paradigm called the fog computing has recently been proposed. Fog computing is based on the idea of enabling computing directly at the edge of the cloud infrastructure rather than being transmitted to the core of the cloud (i.e. data centers) (The Network 2016).

Fog computing is defined as a distributed computing infrastructure that extends computational services offered by the cloud computing to the edge of the network, as presented in Figure 1. Considered as a complement infrastructure to the cloud, fog computing facilitates task processing, networking and data storage between cloud data centers and mobile users (Dastjerdi et al. 2016). Specifically, several applications and computing services that do not fit well with the cloud, could be performed by the fog like applications requiring reduced and predictable latency (e.g., video conferencing), geographically distributed applications (e.g., wireless sensor networks), intelligent transportation applications (e.g., smart connected vehicles, smart traffic lights) (Bonomi et al. 2014) (Deng et al. 2015) (Chen et al. 2015) (Anagnostopoulos et al. 2016).

To serve a fog computing user, an application could be executed both by cloud components such as smart gateways, routers and data center, as well as edge components of the cloud network called fog nodes. Fog nodes are often resource-constrained devices such as base stations, access points, routers, set-top-boxes that support computational and storage resources, transmission

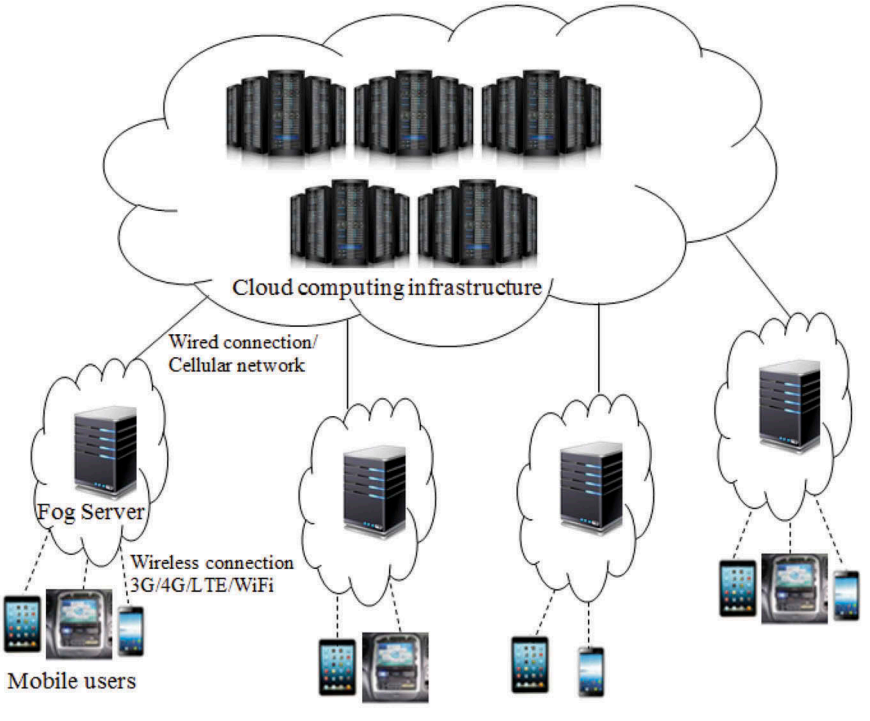


Figure 1. Fog computing infrastructure.

protocols, mobility, diverse types of interface. Fog nodes are used to meet the requirements of computational applications, which request reduced communication overheads, minimized latency, dense and large-scale distribution (Dastjerdi et al. 2016). One example of a fog device, IOx products manufactured by Cisco, helps developers to develop IoT applications for control systems, data aggregation, and cybersecurity. These IoT applications are hosted on a guest operating system that enables compiled code to be executed at the network edge (Cisco. IOx technical overview 2016).

To support fog services, new management and operating functions such as job scheduling were introduced. These new fog functions can be offered by a set of fog servers placed at the edge of networks in selected locations like offices of local government, shopping centers or service stations. Considered as a highly virtualized computing resource, each fog server is equipped with computational device, wireless communication unit, and data storage cards (Deng et al. 2016). A mobile user can communicate directly with fog servers through a single-hop wireless connection using a wireless interface namely 4G LTE devices, WiFi, Bluetooth, etc. Hence, pre-defined applications and pre-cached information are provided by fog servers independently of the cloud resources to benefit the mobile user. In addition, a direct connection between the fog servers and the cloud infrastructure is often established via wired or wireless connections (e.g. through cellular networks) to have access to more computing services/resources and application tools. With the fog computing infrastructure, requests from mobile users can be served by many virtual machines created in several fog servers. Therefore, the requested service (specified in a job request) could be decomposed into a set of service primitives (i.e. job tasks), which are executed on newly created virtual machines after an optimized scheduling within the machines has been determined. The mobile users are charged per CPU hour per virtual machine against a service cost. This situation requires applying an optimal approach (also called a broker approach (Shojafar et al. 2015)) that guarantees the distribution of the tasks across thousands fog devices to serve tens of thousands of mobile user queries per second; it is the job scheduling problem.

In this paper, we focus on the job scheduling problem for the fog computing environment to deal with the problem of the increasing demand for computational resources requested by mobile users to perform a large number of tasks efficiently. The job scheduling problem determines an optimal assignment of various jobs submitted to be executed on the lowest number of fog computing resources (e.g. less memory) in the shortest CPU execution time. As a result, the mobile user achieves faster execution time of his/her tasks at the lowest cost. Each job corresponds to a service request from a mobile user, which could be divided into a set of tasks. In this case, the purpose of a service provider (a fog server) is to allocate fog resources in order to satisfy service level agreement (SLA) signed with a client (i.e. mobile user), while minimizing the CPU execution time and allocated memory for the job (the requested service) (Aazam and Huh 2015). These two performance metrics (i.e. CPU execution time and allocated memory, which could be expressed by a service cost) are the most representative fog device capabilities in terms of CPU and memory resources. Since there is no deterministic polynomial algorithm to solve it, the job scheduling problem is considered as a NP-hard problem, starting with the decomposition of each job (service request) into a set of tasks that are scheduled onto the fog resources required while taking into consideration both the resource availability and the requested service criteria, such as the CPU execution time and the allocated memory.

The rest of the paper is organized as follows. The next section reviews different approaches proposed for job scheduling in fog computing and outlines the contributions of this work. In section 3, the problem statement is presented. Section 4 illustrates the proposed bees life algorithm (BLA). We present a performance evaluation of our proposed bees life algorithm in section 5. Finally, we conclude the paper in section 6 with some future research directions.

2. Related works on job scheduling in fog computing

In the literature, there are few studies on job scheduling in the fog computing. In (Deng et al. 2016), the power consumption and computation latency tradeoff problem, when allocating workloads in both fog and cloud computing systems was presented. The authors formulated the power consumption and delay functions as three sub-problems to be independently solved through existing optimization techniques. Considered as a convex problem, the first sub-problem is to search an optimal compromise between the power consumption and computation latency only in a fog computing sub-system. To solve this first sub-problem, a convex optimization technique was applied (He et al. 2014). The second sub-problem, which is an integer nonlinear programming problem, is devoted to finding a compromise between the power consumption and the computation delay in the cloud computing. This sub-problem was solved with a nonlinear integer programming method (Li and Sun 2006). Finally, the third sub-problem was applied to minimize the delay of data transmission from the fog node to the cloud server for a chosen traffic dispatch rate. This sub-problem was solved with the Hungarian method (Kuhn 2005). This study showed that fog computing enhances the performance of cloud computing by abandoning moderate computation resources, hence communication bandwidth and reduce transmission latency can be saved. However, this study performs the optimization process to reduce power consumption and computation delay by a centralized approach, which is less appropriate with a fog computing infrastructure due the fact that the performance bottleneck of the central node responsible for workload allocation can easily occur thereby degrading the overall system performance.

In (Ningning et al. 2016), the authors proposed a fog computing load balancing mechanism of task allocation based graph partitioning, where fog computing tasks are assigned to a single or multiple virtual machines nodes according to the level of resources required by the task. The authors represent the physical nodes of the fog computing by a non-directional graph. These physical nodes come into a set of virtual machine nodes according to the available fog computing resources, where virtual machine nodes provide services to the users by graph partitioning. To achieve this, a minimum spanning tree is constructed from the entire graph; edges that did not provide enough resources are removed. The resulting graph represents the load balancing partition that is handled by fog computing. This effectiveness of this proposed mechanism has been demonstrated in terms of tasks' run time. However, a major drawback of this approach is that its performance is not optimal for dynamic fog load balancing because of the frequent graph repartitioning needed to cope with fog changes.

In (Cardellini et al. 2015), the authors evaluated the distributed quality of service (QoS)-aware scheduler for data stream processing (DSP) operating in a fog computing environment. In this work, the authors introduced new components, namely a worker monitor, a QoS monitor, and an adaptive scheduler. The worker monitor is responsible for obtaining the incoming and outgoing data rate for each executor defined as a computing component that executes a group of tasks on the fog node. This incoming and outgoing data rate is stored in a local database to be subsequently used by the adaptive scheduler. The QoS monitor estimates the QoS parameters (e.g. network latency) and is responsible for obtaining intra-node utilization and availability and inter-node information. This information is sent to the distributed adaptive scheduler, which implements the system's scheduling policy. The adaptive scheduler runs a single loop iteration periodically, checking for each candidate's task to be executed (called movable executor). If the executor will be effectively relocated, the adaptive scheduler executes the corresponding actions. To achieve this, the scheduler determines a worker node that will execute the candidate executor only if this node improves the application performance in terms of runtime capabilities. This algorithm showed that the distributed QoS-aware scheduler outperforms the centralized default one, thereby improving the application performance and enhancing the system with runtime adaptation capabilities. Nevertheless, complex fog topologies that involve many operators may cause some instability that can decrease the DSP application's availability.

In (Oueis, Strinati, and Barbarossa 2015), the authors addressed the issue of load balancing in fog computing in order to improve users' quality of experience (QoE). The authors of this research activity assume that all requests of different users requiring computation offloading are executed by local computation clusters of resources. In this proposal, a reduced complexity task scheduling algorithm for fog computing was introduced. Resources are allocated to serve a small cell (i.e. fog node) based on some specific scheduling rules. The first rule is to allocate local computational resources at each serving small cell for domestic users. Each small cell sorts users' offloading requests according to a particular parameter such as arrival time, latency constraint, and so on. This ordering also defines the scheduling rule (e.g. first in first out (FIFO) policy, earliest deadline first (EDF) policy) to be adopted for local resource allocation. By this way, different priorities can be given to users' requests depending on the sorting parameter. Despite the fact that this approach yields high users' satisfaction in terms of high latency gain and/or, modest power consumption, this proposal can suffer from a high complexity for large scale fog computing infrastructure because the algorithms used (e.g. EDF) often give good results for low dense computing infrastructures.

The goal of the work described in (Intharawijitr, Iida, and Koga 2016) is to decrease the computing and communication latency, which is often considered as a major drawback of cloud-based services that support message communication especially in 5G cellular networks (Chen, Ling, and Zhang 2011)(Fettweis 2014). The authors of (Intharawijitr, Iida, and Koga 2016) proposed a fog computing architecture to help a computing system (such as a 5G fog-based infrastructure) to achieve maximum efficiency by ensuring an optimal job scheduling. For fog job scheduling, three policies were considered in this study; the first one is the random policy in which one fog node is randomly selected from a uniform distribution to execute a job. The second policy is the lowest latency policy where the fog node providing the lowest total latency based on the current state of the system. Finally, the maximum available capacity policy selects the fog node with the maximum remaining resources among the candidate nodes. The simulation results of this work showed that the lowest latency policy provides significantly better performance because of the availability of resources. The authors concluded that all of these policies can be used to find the most suitable fog node for one job. However, the use of one particular policy might not be the optimal solution for the whole system. Table 1 summarizes the main ideas of the cited works, the improved criteria and limitations of past related works.

Table 1. Summary of the related works discussed.

Approach	Main ideas	Improved criteria	Limitations
(Deng et al. 2016)	– Convex representation – Use of integer nonlinear programming – Use of Hungarian method (He et al. 2014)	– Power consumption – Computation latency	– Considered a centralized fog computing infrastructure
(Ningning et al. 2016)	– Graph partitioning representation and a minimum spanning tree – The use of multiple virtual machines	– Tasks' run times	– High load balancing complexity
(Cardellini et al. 2015)	– Quality of service (QoS)-aware scheduler – Use of an adaptive scheduler	– Network latency	– Not suitable for complex fog computing topology
(Oueis, Strinati, and Barbarossa 2015)	– Quality of Experience	– Task latency – Power consumption	– High complexity for large-scale fog computing infrastructures
(Intharawijitr, Iida, and Koga 2016)	– 5G fog-based infrastructure	– Computing and communication latencies	– Simulated on a partial fog computing system

2.1. *Contributions of this work*

The main contributions of this work are summarized as follows:

- To deal with the job scheduling problem in fog computing, we propose a new bio-inspired optimization approach named bees life algorithm (BLA) to find an optimal allocation for a job's tasks among the available fog resources (i.e. fog nodes) so that we can achieve a tradeoff between the CPU execution time and the allocated memory. In this way, the response latency and bearable cost (i.e. allocated memory) can satisfy mobile users' requests.
- We evaluate the performance of the proposed novel optimization approach based on BLA and demonstrate its efficiency by comparing its performances with other approaches using the particle swarm optimization (PSO) and the genetic algorithm (GA).

3. Job scheduling in fog computing: problem statement

The job scheduling problem in fog computing aims at assigning a set of jobs tasks to fog nodes located at the edge of the network. More specifically, fog nodes are allocated to different tasks of submitted jobs representing the services requested of mobile users, so that both the CPU execution time and the allocated memory required by the overall tasks are minimized. In this subsection, we present our system model for the fog computing infrastructure as well as the formulation of job scheduling problem.

3.1. *System model*

In this paper, the fog computing system assumed consists of '*m*' stationary fog nodes located in the edge of the network, as shown in Figure 2. We assume that our system is composed of an administrator node responsible for job scheduling after receiving all submitted jobs and their parameters (such as the number of tasks for each job). Each job (i.e. a set of tasks) is established and submitted by a mobile user in the form of a service request to be run in the fog computing infrastructure.

To ensure job scheduling, we propose a new bio-inspired algorithm called bees life algorithm (BLA), performed by the administrator node in order to find the optimal order (scheduling), which

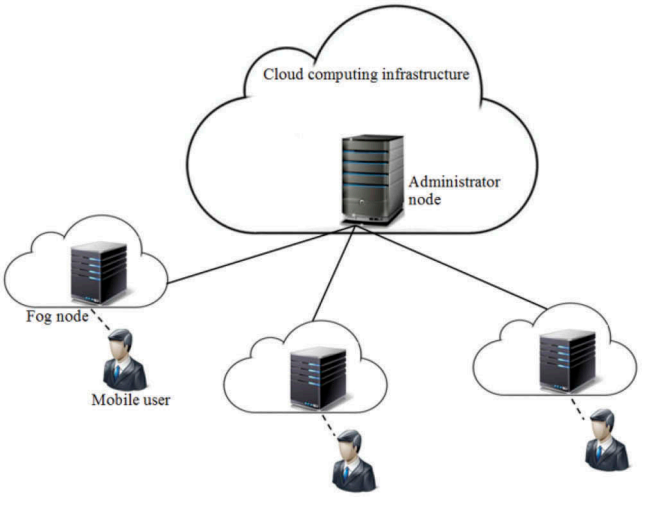


Figure 2. System model for fog job scheduling.

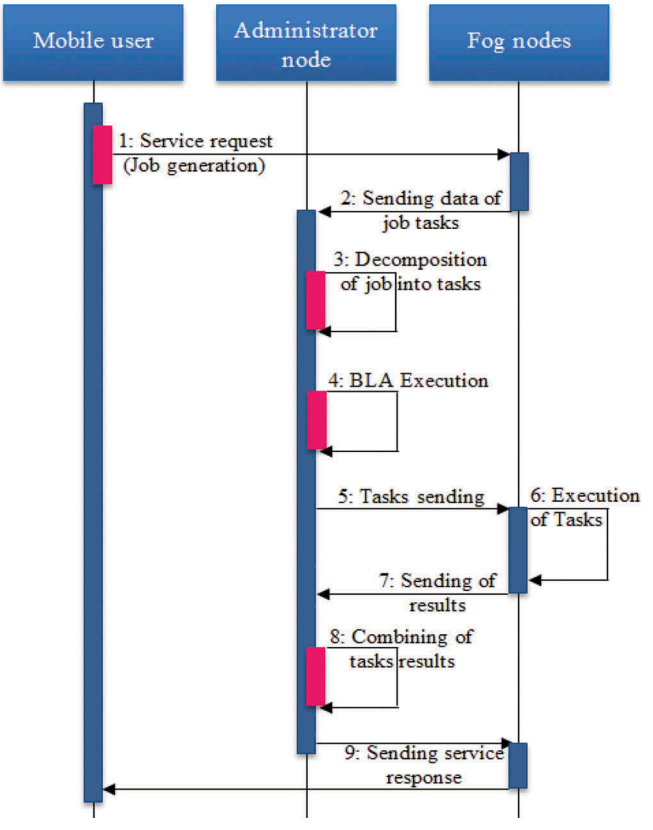


Figure 3. System model of fog job scheduling.

is further executed by the fog nodes. Thus, the fog nodes can together guarantee CPU and memory performance of the scheduled services. We describe the operation of our system model by summarizing the steps for running a scheduled service in Figure 3.

First, a mobile user sends a service request to a fog node located at the edge of the network of this computing infrastructure (step 1). Next, the fog node sends data and parameters of this request as a job to the administrator node often located far away from the user (step 2). The administrator node decomposes the job into a set of tasks (step 3) after which the BLA is executed to find the best job scheduling (step 4). Next, each fog node receives its assigned task (step 5). This task is executed at the level of this fog node (step 6). Each fog node sends its results (step 7) to the administrator node. The administrator node prepares the final result based on the partial received results (step 8) received from the fog nodes. The final result is then sent to the mobile user as a service response (step 9).

3.2. *Problem formulation*

In this subsection, we formulate the job scheduling problem in the fog computing environment. We define a job as a process that corresponds to a service request established by a fog user that could be a mobile user. Then, a set of *'n'* jobs could be submitted in order to be scheduled and executed by the fog computing infrastructure. These jobs are represented by the following job set:

$$Jobs = \{J_1, J_2, \ldots, J_i, \ldots, J_n\}$$

Each job *'i'* among '*n*' jobs (i.e. $1 \le i \le n$) can be partitioned into a set of *'r'* tasks. Each task '*k*' among '*r*' tasks (i.e. $1 \le k \le r$) is disseminated to one fog node *'j'* among '*m*' fog nodes (i.e. $1 \le j \le m$) in order to be executed, such as:

$$Job_iTasks = \left\{JTask_{i1}{}^{a}, JTask_{i2}{}^{b}, \ldots, JTask_{ik}{}^{j}, \ldots, JTask_{ir}{}^{m}\right\}$$

For example, tasks of job '*i*':

$$Job_iTasks = \left\{JTask_{i1}{}^{4}, JTask_{i2}{}^{6}, JTask_{i3}{}^{9}\right\}$$

are performed as follows:

1st task ($JTask_{i1}{}^{4}$) is executed in fog node 4, 2nd task ($JTask_{i2}{}^{6}$) is executed in fog node 6, and 3rd task ($JTask_{i3}{}^{9}$) is executed in fog node 9.

Consequently, each fog node FN_j can execute a set of disjoint subset of the decomposed jobs set (i.e. tasks). For its assigned jobs, FN_j ensures the execution of its tasks as follows:

$$FN_jTasks = \left\{JTask_{ax}{}^{j}, JTask_{by}{}^{j}, \ldots, JTask_{ik}{}^{j}, \ldots, JTask_{nr}{}^{j}\right\}$$

The union of these overall disjoint subsets is the complete set of jobs.

For example, after scheduling, the following tasks are assigned to the fog node FN_j:

$$FN_jTasks = \left\{JTask_{23}{}^{j}, JTask_{61}{}^{j}, JTask_{74}{}^{j}\right\}$$

Then, FN_j carries out 3rd task of job 2, 1st task of job 6, and 4th task of job 7.

On the one side, the total CPU execution time of all tasks ('*r*' tasks) assigned to FN_j would be:

$$CPU\_Execution\_Time\left(FN_jTasks\right) = \underset{\substack{1 \le k \le r \\ i \in jobs\ of\ selected\ tasks}}{\text{sum}} \left(JTask_{ik}^{j}.StartTime + JTask_{ik}^{j}.ExeTime\right)$$

where $JTask_{ik}^{j}.StartTime$ represents the starting time of task *'k'* of a job 'i' executed on FN_j and $JTask_{ik}^{j}.ExeTime$ is the CPU execution time of this task *'k'* at FN_j.

On the other side, the allocated memory to task *'k'* assigned to FN_j is calculated, as follows:

$$Memory\left(FN_jTasks\right) = \underset{\substack{1 \le k \le r \\ i \in jobs\ of\ selected\ tasks}}{\max} \left(JTask_{ik}^{j}.AllocatedMemory\right)$$

Therefore, the job scheduling problem in the fog computing could be formulated as searching of a set:

$$FNTasks = \{FN_1Tasks, FN_2Tasks, \ldots, FN_mTasks\}, \text{where :}$$

$$FN_jTasks = \left\{JTask_{ax}{}^{j}, JTask_{by}{}^{j}, \ldots, JTask_{ik}{}^{j}, \ldots, JTask_{nr}{}^{j}\right\}, \textit{as explained above.}$$

3.3. *Cost function*

A cost function is defined to evaluate the quality of the expected solution *(FNTasks)*. This cost function is a minimization function used to measure the optimality of the above two objectives: *CPU execution time and allocated memory size*, as follows:

$$Cost\_function(FNTasks) = Min\left[\sum_{m}^{j=1}\left(Cost\_function\left(JTask_{ik}^{j}, FN_j\right)\right)\right]$$

$$where, Cost\_function\ \left(JTask_{ik}{}^{j}, FN_j\right) = \\ w1.CPU\_Execution\_Time\left(FN_jTasks\right) + w2.Memory\left(FN_jTasks\right)$$

where, w_1 and w_2 are weight factors that are fixed to emphasize the importance of each of the two evaluated objectives (i.e. CPU execution time and allocated memory). In other words, the choice of the weights in such multi-objective optimization approaches reflects the preference of the decision maker (Collette and Siarry 2013).

4. Bees life algorithm for fog computing job scheduling

Several bee-based algorithms (Yuce et al. 2013)(Akbari, Mohammadi, and Ziarati 2010)(Ozturk, Hancer, and Karaboga 2015) have been proposed recently. In (Yuce et al. 2013) and (Akbari, Mohammadi, and Ziarati 2010) the authors focused on solving functional optimization, and in (Ozturk, Hancer, and Karaboga 2015) the authors addressed the clustering problem. In contrast, in this work, we focus on novel bees life algorithms that can solve the job scheduling challenge in the fog computing environment. In this section, we present this BLA proposal. First, we describe the life of bees in nature which motivates the development of BLA. Next, we explain BLA and our proposed job scheduling approach in fog computing is explained.

4.1. *Bees in nature*

Considered as domestic insects, the bees live in a colony that generally composted of three kinds of individuals: a queen representing the breeding female of the colony, few thousands of drones which are the males, and several thousands of workers forming sterile females responsible of collecting nectar and pollen as a source of energy to feed the colony individuals. Two major behaviors differentiate district bees from other insects, namely food source searching and bee marriage. To search food, bee workers fly and explore the environment and return to the hive if found to communicate and share this discovery with its nest mate using a dance language. After this, some workers are sent to collect food till the food quantity exhausted. The second behavior is the bees' reproduction (the marriage behavior) realized by the queen and the males. During the mating flight, the marriage is ensured by the queen that mates with several males (Bitam, Batouche, and Talbi 2010).

4.2. *Illustration of bees life algorithm*

Our proposed algorithm (i.e. BLA) is a biologically inspired optimization method that imitates the most important behaviors of Bees namely marriage (i.e. reproduction) and food foraging (Bitam, Mellouk, and Zeadally 2015).

As shown in Figure 4, BLA first step is to choose *N* bees (individuals) at random from the search space to form the initial population. Then, this initial population is evaluated by calling the cost function. After sorting this population, the fittest bee is considered as the queen, the drones are the following *'D'* fittest bees, whereas the remaining bees form the workers (their number is *'W'*). We mention that the values of *'N', 'D'* and *'W'* are fixed by the user as experimental values.

The next step is to execute a number of BLA iterations until the stopping criterion is met. Each iteration represents bees' life cycle (BLA iteration) consisted of the main two behaviors of bees' (i.e. reproduction and food foraging). During the reproduction stage, a mating-flight is performed where the queen mates with a set of drones by the crossover and mutation operators. Therefore, *'N'* broods are laid by the queen. All hive individuals (i.e. *'N'* original bees and *'N'* broods) are evaluated and sorted by calling the cost function; hence, the best bees will substitute the precedent queen, the following *'D'* fittest bees are selected to form the new drones and the

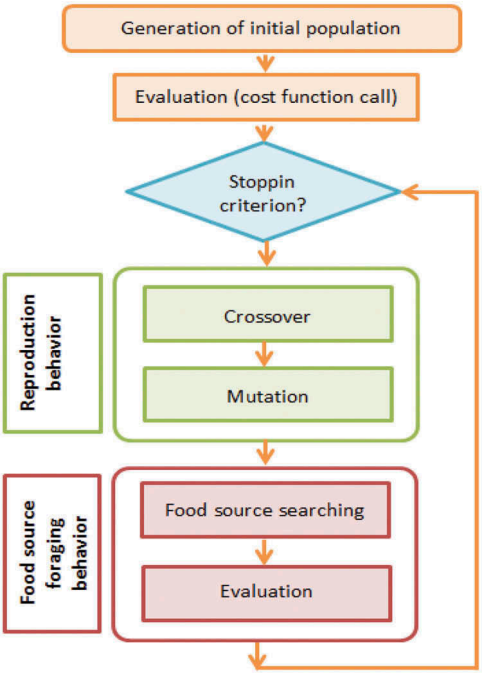


Figure 4. Bees Life Algorithm pseudo-code.

following *'W'* best bees are considered as the new workers. Now, the food foraging stage is executed by the workers. Each worker is responsible to find a food source (i.e. region of flowers) and then to recruit several other workers to collect the found food. We note that there are more workers (*'FBest'*) recruited for more promising regions; *'B'* regions in which there are more food compared with the remaining regions (only *'FOther'* workers are recruited, *FOther* □ *FBest*). Note that FBest and FOther values are chosen as experimental values. Afterward, the best worker of every region keeps alive in the next population. This choice of the best worker is done with the cost function call.

4.3. *Individual representation, initialization and stopping criterion*

4.3.1. *Individual representation*

Since fog job scheduling is considered as an optimization problem, we represent the search space by a directed graph. We assume that one job (consisting of a set of tasks) is represented by a graph starting with one root node, ending with one leaf node, and its tasks are represented by a set of intermediate nodes, where the root expresses the beginning of a job and the leaf the end of the job. Consequently, one path in the graph starting from the root to the leaf forms one solution (i.e. one individual), which is represented by a string. For instance, we have a job *'i'* that is consisted of three tasks ($JTask_{i1}$, $JTask_{i2}$, $JTask_{i3}$). There are multiple possibilities to schedule these job tasks among *'m'* fog nodes. For example, one solution configuration is that this job is scheduled as follows: $JTask^{5}_{i1}$ in FN_5, $JTask^{7}_{i2}$ in FN_7, and $JTask^{1}_{i3}$ in FN_1, as shown in Figure 5.

There are *'N'* individuals (solutions) for each population. Each individual indicates a set of fog nodes, as follows:

$$FNTasks = \{FN_1Tasks, FN_2Tasks, \ldots, FN_mTasks\} : \textit{set of fog nodes.}$$

Each fog node is responsible of executing its affected tasks as follows:

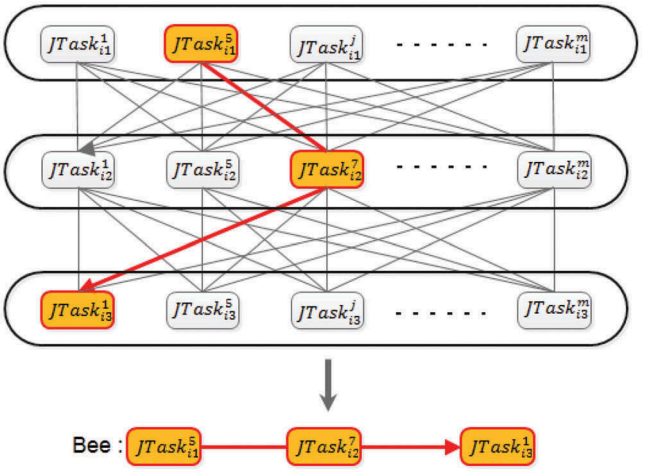


Figure 5. BLA individual representation.

$$FN_jTasks = \{JTask_{ax}{}^{j}, JTask_{by}{}^{j}, \ldots, JTask_{ik}{}^{j}, \ldots, JTask_{nr}{}^{j}\}$$

4.3.2. *BLA initialization*

The BLA initialization generates '*N*' individuals selected randomly to form the first population. For instance, the following individuals are selected:

$$FNTasks = \{FN_1Tasks, FN_2Tasks, \ldots, FN_mTasks\} =$$
$$\{<JTask_{ax}{}^{1}, JTask_{by}^{1}, \ldots, JTask_{cz}^{1}>, <JTask_{a'x'}^{2}, JTask_{b'y'}^{2}, \ldots, JTask_{c'z'}^{2}>, \ldots,$$
$$<JTask_{a''x''}^{m}, JTask_{b''y''}^{m}, \ldots, JTask_{c''z''}^{m}>\}$$

To evaluate each individual, the cost function (mentioned in subsection 3.3) is applied.

4.3.3. *BLA stopping criterion*

The stopping criterion represents a pre-determined value (*MaxIT*), fixed by the job scheduling administrator, as the maximum number of BLA iterations. *MaxIT* could correspond to the computing resource limit (i.e. performance of the fog node) or to a time period that a mobile user can support before obtaining the results. It is worth noting that a convergence value could be defined which is lower than *MaxIT*, expressing the case when there is no significant improvement in the values of the cost function, from one generation to the next. In this study, *MaxIT* is the maximum number of BLA iterations, which represents the fog node computing capacity. We used *MaxIT* runs for a set instances of the job scheduling problem in fog computing. These instances differ in the number of job tasks chosen to evaluate the effectiveness of BLA.

4.4. *Optimization operators of BLA*

BLA suggests two (02) genetic operators in its reproduction phase in order to increase diversity of individuals in the populations generated; these operators are: the crossover and mutation. However, in the food source searching stage, a greedy approach is applied as a local search algorithm to better improve solutions obtained in the reproduction phase. This greedy algorithm improves the solution obtained through an iterative process by applying local changes in the search space until an optimal solution is discovered when there is no improvement in the cost function value of this solution. In the following sections, we illustrate the use of these operators.

4.4.1. *Crossover*

Crossover operation is applied on two colony individuals called parents which are the queen and a drone. These parents are combined together to form two new individuals called offsprings. This process is performed by the queen and '*D*' drones in order to produce '*N*' offsprings. The queen and the '*D*' drones are selected among the best existing individuals in the population with a preference toward the cost function as explained in subsection 4.2. In this way, better offsprings are expected to appear in the next generation, and so on until an overall individual (solution) is obtained. We note that the crossover is applied with a probability of 'P_c', which is often a high value. There are several crossover operators that have been proposed in the literature. In this work, we select the two-point crossover operator (as shown in Figure 6). With a two-point crossover, two points are randomly selected for the string that represents each parent; in our case, the parent is represented by a graph path which is represented in turn by a string (as described in section 4.3.A). As a result, each parent is divided into three substrings.

The first part (starting from beginning to the first crossover point) and the third part (starting from the second crossover point to the end of the string representing the first parent (i.e. the queen) are copied to form the first and the third parts of the first offspring. The second part (starting from the first crossover point to the second crossover point) of the second parent (i.e. a drone) is copied to form the second part of this first offspring. This first offspring inherits string parts from its two parents. Similarly, the second offspring is formed by inheriting its first and third parts from the first and third parts of the second parent, while its second part is inherited from the second part of the first parent. We propose for job scheduling in fog computing a two-point crossover operator, as illustrated in Figure 6. In this figure, the first and second crossover points are selected at random after the first and the fourth represented tasks of the individual. In its first and third parts, the first offspring (P_i') is formed by the first and the third parts of the first parent (P_i), whereas the first offspring's middle part is the same as the middle part similar to the second parent (P_j).

4.4.2. *Mutation*

Mutation is a unary operator which introduces changes into the characteristics of the offsprings resulting from the crossover operation. These changes are very small and random according to the mutation probability of 'P_m', generally chosen as a small value. Therefore, the new offspring will not be different from the original one. In our proposal, we propose a mutation operator called task inversion. The task inversion mutation selects a task at random which is then replaced by another task also selected at random in the same fog node. This process is repeated for other fog nodes.

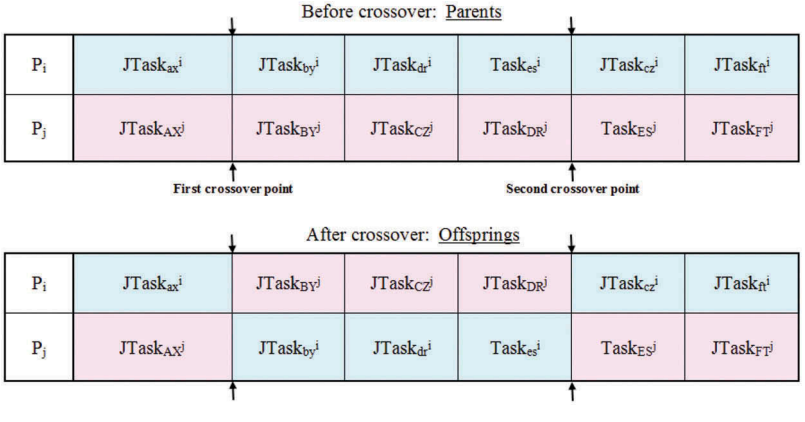


Figure 6. Two-point crossover.

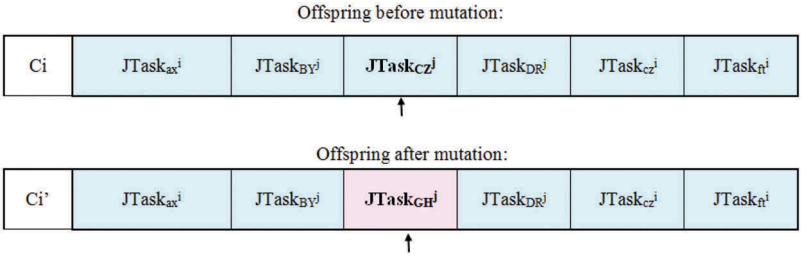


Figure 7. Task inversion mutation.

Figure 7 illustrates a task inversion mutation of the offspring C_i that will be C'_i after mutation. In this example, the task $JTask_{cz}^{\ i}$ is replaced by the task $JTask_{gh}^{\ i}$ designed to the same fog node '*i*'.

4.4.3. *Greedy local search approach*

In the foraging part of BLA, we use a greedy approach for the local search process in order to reach the optimal individual (solution) among different individuals (solutions) in the neighborhood of the original individual. In this work, one individual task (a job task) is randomly appointed to be substituted by another task from the nearest fog node. Figure 8 shows an example of the greedy local search approach used. In this example, the task $JTask_{by}^{\ i}$ is selected from the fog node '*i*' and is replaced by the task $JTask_{B'Y'}^{\ j}$ from the fog node '*j*'.

4.5. *BLA complexity analysis*

We analyze the computational complexity of BLA as follows. First, two *O(N)* time units are used to choose and to evaluate *N* bees which represent the population of individuals. This means that the initialization performs in $O(N) + O(N) \approx O(N)$.

BLA process can be executed in the worst case *ItMax* iterations, where *ItMax* is the number of iterations executed by BLA as explained in subsection 4.3.C). During one iteration, the number of offsprings generated is $(N \times p_c)$ with a crossover probability of p_c. $((N \times p_c) \times p_m)$ will be mutated among the offsprings where p_m is the mutation probability.

Next, *W* workers among *N* individuals (N is the population size) performs the food source foraging by using a greedy local search. *FBest* foragers are sent to *B* best regions and *FOthers* foragers are sent to the remaining regions *(W-B)* regions. Therefore, *FBest* foragers execute in *O*

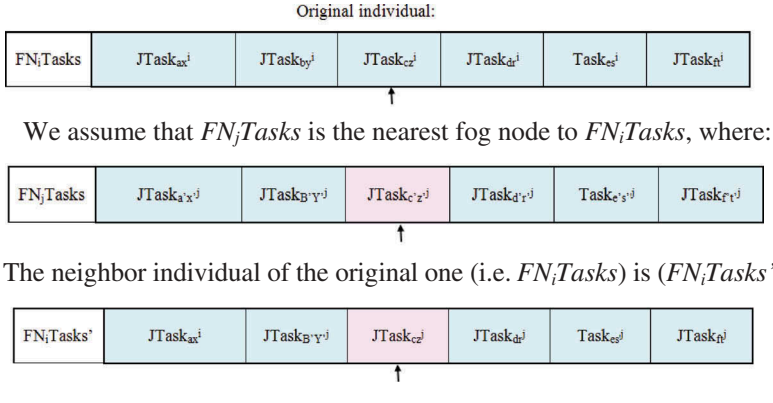


Figure 8. Greedy local search approach.
We assume that FN_jTasks is the nearest fog node to FN_iTasks, where: The neighbor individual of the original one (i.e. FN_iTasks) is (FN_iTasks')

(FBest × B). However, *FOther* foragers execute in *O(FOthers × (W-B))*. For each iteration, the time complexity can be calculated in this way:

$$O(N) + O((N \times p_c) \times p_m) + O(FBest \times B) + O(FOthers \times (W - B))$$

$$\approx O(N + ((N \times p_c) \times p_m) + (FBest \times B) + (FOther \times (W - B)).$$

BLA executes in the worst case:

$$O(ItMax \times (N + ((N \times p_c) \times p_m) + (FBest \times B) + (FOther \times (W - B))))$$

As conclusion, the complexity is linear.

5. Performance evaluation

In this section, we conducted a performance evaluation to assess the effectiveness of BLA to job scheduling in fog computing. To achieve this goal, we implemented the BLA framework in C++. This framework simulates four (04) fog computing infrastructures each consisting of 5, 10, 15 and 20 fog nodes. In addition, we also used one administrator node (a data center) for each fog computing infrastructure. We performed all experiments on a desktop PC with an Intel Pentium Dual-Core CPU T4500, a clock rate of 2.30 gigahertz, and 4 gigabytes of memory. We use the following two performance evaluation metrics to evaluate our proposed approach.

- *CPU execution time* (measured in second): this is defined as the time between the start and the completion of a given task executed on a fog node. In this study, we do not include the time taken for separating and combining tasks before and after their scheduling because this time is constant and does not affect the job scheduling process. A task is composed of a set of instructions. We assume that each instruction requires one clock cycle to be executed. Then, the CPU execution time is calculated as follows:

  CPU execution time = number of instructions of a task (i.e. clock cycles for a task) /clock rate

- *Allocated memory size* (measured in byte): this is defined as the total amount of memory (i.e. the main storage unit) of a fog node, devoted to the execution of a given task.

Based on the above metrics, we compare the results obtained by BLA with those obtained by the traditional genetic algorithm and particle swarm optimization.

Genetic algorithm (GA) is one of the most attractive population-based optimization methods, which is inspired from natural evolution and biological genetics to solve NP-hard problem. GA starts with a random initialization of a population consisting of number of individuals (i.e. potential solutions). The population is improved through a set of iterations (i.e. generations) using two genetic operators such as crossover and mutation. The crossover is applied with a high probability on pairs of parents selected from the population and produces a set of offsprings. Each offspring could be mutated with low probability. Consequently, the best individuals among parents and offsprings are chosen to form a new population and so on until the optimal individual (i.e. the optimal solution) is reached (Xu et al. 2014).

Particle swarm optimization (PSO) algorithm is one of the efficient optimization algorithms inspired from the behavior of a flock of birds or group of fishes when they move to a desired destination (Abdi, Motamedi, and Sharifian 2014). Similar to GA, PSO initializes a population of particles (i.e. potential solutions) where each particle is defined by its position representing the solution quality. Each particle is defined by two specific positions pbest and gbest. Pbest is defined as the best position (i.e. the optimal solution) that the particle has experienced since the process has started, whereas gbest represents the absolute best position found among all particles (optimal

solution found). To find the optimal solution several iterations are executed. After each iteration, the particle's position is updated based on its prior position as well as to the particle's velocity that represents a close solution in the search space of the problem. The following formula shows how a particle's position (X_{k+1}) is updated according to the velocity V_{k+1} in the next iteration (k + 1):

$$X_{k+1} = X_k + V_{k+1}$$

The particle's velocity is also updated based on its prior velocity, pbest, gbest and five parameters (i.e. w, c_1, c_2, r_1, and r_2). 'w' is a weight chosen for rapid particle movement, 'c_1' and 'c_2' are constant acceleration coefficients, where 'r_1' and 'r_2' are numbers with uniform distributions in the range of [0, 1] (Salim, Batouche, and Talbi 2010). The velocity formula is:

$$V_{k+1} = w \times V_k + c_1 r_1 (pbest_k - X_k) + c_2 r_2 (gbest - X_k)$$

5.1. *Experimental settings*

In our experimental study, we conducted a set of simulation tests according to different fog computing infrastructures. Fog nodes are heterogeneous in terms of their processing capability and storage capacity. In order to evaluate the effectiveness of BLA in scheduling the tasks of jobs, we assume that every fog node has its own processing capability represented by the capacity of the CPU clock rate of the fog node (measured in gigahertz -GHz-), as well as its own storage capacity represented by the available memory (measured in gigabytes -GBytes-). Table 2 displays the CPU clock rate and available memory size of each one of the 20 simulated fog nodes made available to jobs to be executed. We note that for the cases 5, 10, 15 and 20 fog nodes, first 5^{th}, 10^{th}, 15^{th} and 20^{th} columns of Table 2 are considered as fog nodes data respectively.

For example, fog node number 1 offers 1.25 GHz as CPU clock rate needed to execute one task instruction, and 1.0 GBytes of memory available and shared among submitted jobs.

In this study, we consider 5 jobs submitted to be performed among these 5, 10, 15 or 20 fog nodes. Each job consists of 5 tasks. These 5 tasks are consisted of a set of instructions (Ins) and require a memory space (Mem) as illustrated in Table 3.

For instance, job_1 is composed of task 1, 2, 3, 4 and 5 of 2 x 10^9, 2 x 10^9, 3 x 10^9, 2 x 10^9 and 3 x 10^9 instructions, respectively. Also, job_0 requires 0.20, 0.10, 0.10, 0.30, and 0.10 GBytes of its tasks 1, 2, 3, 4 and 5, respectively.

It is worth pointing out that we proposed this benchmark (i.e. Table 2 and Table 3) because there are no benchmarks published in the literature in this research area. These values of the fog nodes' processing capabilities and storage capacities were selected from architectures and computing systems of current fog nodes such as Personal Digital Assistants (PDAs), smart phones, onboard computers, etc. Additionally, the task sizes chosen are the average ones used by tasks in the literature (Cao et al. 2013). BLA, GA, and PSO parameters are depicted in Table 4.

In this study, we have set the weights w_1 and w_2 equal at 1 and 10, respectively, in order to obtain a uniform representation of the solutions on the tradeoff CPU execution time and allocated memory of fog nodes.

Table 2. CPU clock rate and available memory size of fog nodes.

FN_j	1	2	3	4	5	6	7	8	9	10
CPU clock rate (GHz)	1.25	1.00	0.83	1.00	0.83	1.25	0.90	0.77	1.11	1.00
Available Memory size (GBytes)	1.0	0.9	1.4	1.5	1.4	1.0	1.2	0.9	1.0	1.2
FN_j	11	12	13	14	15	16	17	18	19	20
CPU clock rate (GHz)	0.77	1.11	1.00	0.90	1.25	0.83	0.83	1.00	1.25	1.00
Available Memory (GBytes)	1.2	1.1	1.0	0.8	1.2	1.4	1.2	1.5	1.0	0.8

Table 3. Parameters of each task of a job [Ins: number of instructions of $task_i$; Mem: memory required (GBytes)].

	$Task_1$		$Task_2$		$Task_3$		$Task_4$		$Task_5$	
	Ins	Mem	Ins	Mem	Ins	Mem	Ins	Mem	Ins	Mem
Job_1	2×10^9	0.20	2×10^9	0.10	3×10^9	0.10	2×10^9	0.30	3×10^9	0.10
Job_2	3×10^9	0.10	1×10^9	0.30	2×10^9	0.20	2×10^9	0.20	3×10^9	0.10
Job_3	2×10^9	0.20	3×10^9	0.30	1×10^9	0.20	1×10^9	0.10	1×10^9	0.20
Job_4	1×10^9	0.30	2×10^9	0.10	1×10^9	0.20	3×10^9	0.20	1×10^9	0.20
Job_5	1×10^9	0.20	1×10^9	0.20	3×10^9	0.10	3×10^9	0.10	2×10^9	0.30

Table 4. BLA, GA, and PSO parameters.

	Population size (N)							Weight factors		
	Number of queens	Number of drones	Number of workers	Crossover ratio			Mutation ratio	w_2	w_2	Number of iterations (MaxIt)
BLA	1	7	12	90%			1%	1	10	200
GA		20								
PSO		20		w	c_1	c_2	r_1	r_2		
				1	0.5	0.5	0.5	0.5		

5.2. *Experimental results*

Figures 9, 10, and 11 show the results of BLA, GA and PSO executions of 5 jobs each consisting of 5 tasks. For each case (5, 10, 15 and 20 fog nodes in the fog computing infrastructure), Figure 9 presents the CPU execution time returned after executing all job tasks by different fog nodes. Figure 9 shows that BLA gives the reduced execution time for various simulation tests. For example, BLA executes all jobs in 52.42 seconds with 20 fog nodes compared to GA and PSO, which took 53.05 and 53.34 seconds respectively.

Figure 10 shows the allocated memory by all job tasks after scheduling with BLA, GA and PSO. In this case, BLA allocates the minimum size of memory. For instance, in the case of 20 fog nodes, BLA consumes only 3.0 GBytes compared to GA and PSO which allocate 3.4 and 3.1 GBytes respectively.

Figure 11 shows the cost function that represents the multi-objective job scheduling problem, in which BLA outperforms GA and PSO for all tests.

Table 5 presents an example of tasks repartitioning for the case of 20 fog nodes, in which different tasks of each job are scheduled and assigned to a fog node. In this table, tasks repartitions obtained by BLA, GA and PSO are presented according to CPU execution time, allocated memory, and cost function.

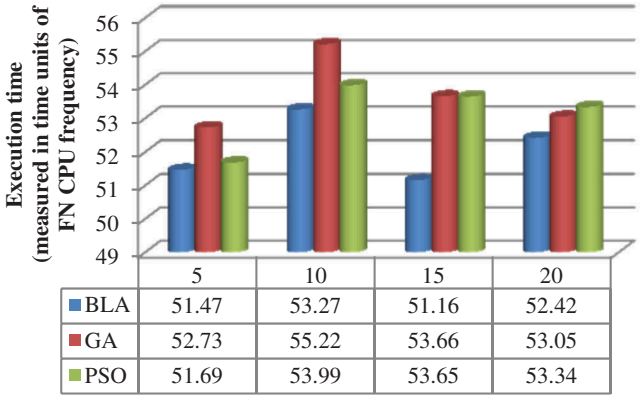
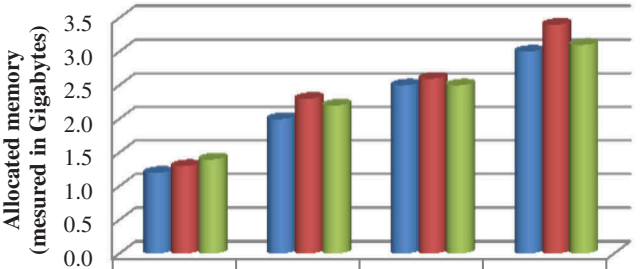
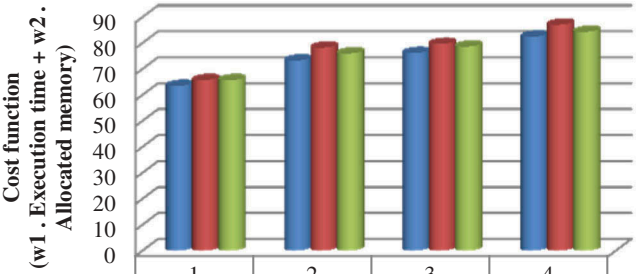


Figure 9. Job execution time after BLA, GA and PSO scheduling.



	5	10	15	20
BLA	1.2	2.0	2.5	3.0
GA	1.3	2.3	2.6	3.4
PSO	1.4	2.2	2.5	3.1

Figure 10. Allocated memory by jobs after BLA, GA and PSO scheduling.



	1	2	3	4
BLA	63.47	73.27	76.16	82.42
GA	65.73	78.22	79.66	87.05
PSO	65.69	75.99	78.65	84.34

Figure 11. Cost function of BLA, GA and PSO fog job scheduling problem.

For example, BLA assigns task 1 of job 5 ($JTask_{51}^{1}$) to the fog node 1 ($FN_1 Tasks$). This task is composed by 1 x 10^9 instructions and required 0.80 seconds (i.e. 1 x 10^9 /1.25 GHz) to be executed in $FN_1 Tasks$. Moreover, this task uses 0.2 GBytes of memory. By this way, all the other fog nodes execute their assigned tasks. By applying formulas to calculate total CPU execution time and total consumed memory cited in subsection 3.2 and the cost function cited in subsection 3.3, we conclude that BLA can perform all these tasks in 51.47 seconds by consuming only 3.0 GBytes. In addition, the minimized cost function gives score of 82.42, in this case of scheduling these jobs in 20 fog nodes using BLA. However, GA and PSO can run all job tasks in 53.05 and 53.34 seconds respectively with 3.4 and 3.1 Gbytes memory consumed by GA and PSO respectively. As a result, the cost functions achieved by GA and PSO are 87.05 and 84.34 respectively.

The rest of the results (cases of job scheduling in 5, 10 and 15 fog nodes for BLA, GA, and PSO) is presented in the appendix.

5.3. *Discussion*

Figures 9, 10 and 11 show that BLA yield the best scheduling cost as compared with the scheduling cost obtained with GA or PSO. It represents the best trade-off between the execution time and the allocated memory of the overall tasks of all jobs. For example, Table 4 shows that (J_1,…, J_5) are performed by the 20 fog nodes (FN_1,…FN_{20}), where BLA executes all these jobs with a reduced

Table 5. BLA, GA and PSO results for 20 fog nodes.

Algorithm	Task scheduling (job tasks assigned to a fog node $FN_i Tasks$)	Number of task instructions assigned to $FN_i Tasks$	CPU execution time of a fog node (task instructions /clock rate)	Allocated memory (gigabytes)	Total
BLA	$FN_1 Tasks = JTask_{51}^{1}$	$1x10^9$	0.80	0.2	
	$FN_2 Tasks = JTask_{32}^{2}, JTask_{45}^{2}$	$3x10^9$, $1x10^9$	4.00	0.3, 0.2	
	$FN_3 Tasks = JTask_{53}^{3}, JTask_{55}^{3}$	$3x10^9$, $2x10^9$	6.02	0.1, 0.3	
	$FN_4 Tasks = /$	/	/	/	
	$FN_5 Tasks = /$	/	/	/	
	$FN_6 Tasks = JTask_{52}^{6}$	$1x10^9$	0.80	0.2	
	$FN_7 Tasks = /$	/	/	/	
	$FN_8 Tasks = JTask_{44}^{8}$	$3x10^9$	3.89	0.2	
	$FN_9 Tasks = /$	/	/	/	
	$FN_{10} Tasks = JTask_{24}^{10}, JTask_{31}^{10}, JTask_{35}^{10}, JTask_{41}^{10}$	$2x10^9$, $2x10^9$, $1x10^9$, $1x10^9$	6.00	0.2, 0.2, 0.2, 0.3	
	$FN_{11} Tasks = JTask_{22}^{11}, JTask_{42}^{11}$	$1x10^9$, $2x10^9$	3.89	0.3, 0.1	
	$FN_{12} Tasks = JTask_{13}^{12}, JTask_{54}^{12}$	$3x10^9$, $3x10^9$	5.40	0.1, 0.1	
	$FN_{13} Tasks = /$	/	/	/	
	$FN_{14} Tasks = /$	/	/	/	
	$FN_{15} Tasks = JTask_{25}^{15}$	$3x10^9$	2.40	0.1	
	$FN_{16} Tasks = JTask_{14}^{16}, JTask_{15}^{16}, JTask_{23}^{16}$	$2x10^9$, $3x10^9$, $2x10^9$	8.43	0.3, 0.1, 0.2	
	$FN_{17} Tasks = JTask_{12}^{17}, JTask_{21}^{17}$	$2x10^9$, $3x10^9$	6.02	0.1, 0.1	
	$FN_{18} Tasks = JTask_{33}^{18}, JTask_{34}^{18}$	$1x10^9$, $1x10^9$	2.00	0.2, 0.1	
	$FN_{19} Tasks = JTask_{43}^{19}$	$1x10^9$	0.80	0.2	
	$FN_{20} Tasks = JTask_{11}^{20}$	$2 x10^9$	2.00	0.2	
	Brut Total:	49 x 10^9 instructions	52.42 seconds	3.0 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)		52.42	30	**= 82.42**
GA	$FN_1 Tasks = JTask_{54}^{1}$	$3x10^9$	2.40	0.1	
	$FN_2 Tasks = JTask_{24}^{2}$	$2x10^9$	2.00	0.2	
	$FN_3 Tasks = /$	/	/	/	
	$FN_4 Tasks = JTask_{21}^{4}$	$3x10^9$	3.00	0.1	
	$FN_5 Tasks = JTask_{11}^{5}, JTask_{15}^{5}$	$2x10^9$, $3x10^9$	6.02	0.2, 0.1	
	$FN_6 Tasks = /$	/	/	/	
	$FN_7 Tasks = /$	/	/	/	
	$FN_8 Tasks = JTask_{31}^{8}, JTask_{43}^{8}$	$2x10^9$, $1x10^9$	3.89	0.2, 0.2	
	$FN_9 Tasks = JTask_{23}^{9}, JTask_{41}^{9}$	$2x10^9$, $1x10^9$	2.70	0.2, 0.3	
	$FN_{10} Tasks = /$	/	/	/	
	$FN_{11} Tasks = JTask_{33}^{11}, JTask_{53}^{11}$	$1x10^9$, $3x10^9$	5.19	0.2, 0.1	
	$FN_{12} Tasks = JTask_{55}^{12}$	$2x10^9$	1.80	0.3	
	$FN_{13} Tasks = JTask_{44}^{13}, JTask_{45}^{13}$	$3x10^9$, $1x10^9$	4.00	0.2, 0.2	
	$FN_{14} Tasks = /$	/	/	/	

(*Continued*)

Table 5. (Continued).

Algorithm	Task scheduling (job tasks assigned to a fog node FN_iTasks)	Number of task instructions assigned to FN_iTasks	CPU execution time of a fog node (task instructions /clock rate)	Allocated memory (gigabytes)	Total
	$FN_{15}Tasks = JTask_{50}^{15}$	$1x10^9$	0.80	0.2	
	$FN_{16}Tasks = JTask_{25}^{16}, JTask_{34}^{16}, JTask_{35}^{16}, JTask_{42}^{16}$	$3x10^9$, $1x10^9$, $1x10^9$, $2x10^9$	8.43	0.1, 0.1, 0.2, 0.1	
	$FN_{17}Tasks = JTask_{12}^{17}, JTask_{32}^{17}$	$2x10^9$, $3x10^9$	6.02	0.1, 0.3	
	$FN_{18}Tasks = JTask_{52}^{18}$	$1x10^9$	1.00	0.2	
	$FN_{19}Tasks = JTask_{22}^{19}$	$1x10^9$	0.80	0.3	
	$FN_{20}Tasks = JTask_{13}^{20}, JTask_{14}^{20}$	3 $x10^9$,2 $x10^9$	5.00	0.1, 0.3	
	Brut Total:	49 x 10^9 instructions	53.05 seconds	3.4 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)		53.05	34	**= 87.05**
PSO	$FN_1Tasks = JTask_{35}^{1}$	$1x10^9$	0.80	0.2	
	$FN_2Tasks = JTask_{44}^{2}$	$3x10^9$	3.00	0.2	
	$FN_3Tasks = JTask_{41}^{3}, JTask_{21}^{3}, JTask_{45}^{3}$	$1x10^9$, $3x10^9$, $1x10^9$	6.02	0.3, 0.1, 0.2	
	$FN_4Tasks = JTask_{55}^{4}$	$2x10^9$	2.00	0.3	
	$FN_5Tasks = JTask_{14}^{5}, JTask_{32}^{5}$	$2x10^9$, $3x10^9$	6.02	0.2, 0.3	
	$FN_6Tasks = /$	/	/	/	
	$FN_7Tasks = JTask_{12}^{7}, JTask_{42}^{7}$	$2x10^9$, $2x10^9$	4.44	0.1, 0.1	
	$FN_8Tasks = JTask_{34}^{8}$	$1x10^9$	1.29	0.1	
	$FN_9Tasks = JTask_{33}^{9}, JTask_{52}^{9}, JTask_{24}^{9}, JTask_{44}^{9}$	$1x10^9$, $1x10^9$, $2x10^9$, $3x10^9$	6.30	0.2, 0.3, 0.2, 0.2	
	$FN_{10}Tasks = JTask_{53}^{10}, JTask_{31}^{10}$	$3x10^9$, $2x10^9$	5.00	0.1, 0.2	
	$FN_{11}Tasks = JTask_{11}^{11}$	$2x10^9$	2.85	0.2	
	$FN_{12}Tasks = JTask_{22}^{12}$	$1x10^9$	0.90	0.3	
	$FN_{13}Tasks = JTask_{25}^{13}$	$3x10^9$	3.00	0.1	
	$FN_{14}Tasks = JTask_{51}^{14}$	$1x10^9$	0.80	0.2	
	$FN_{15}Tasks = /$	/	/	/	
	$FN_{16}Tasks = JTask_{23}^{16}, JTask_{13}^{16}$	$2x10^9$, $3x10^9$	6.02	0.2, 0.1	
	$FN_{17}Tasks = JTask_{54}^{17}$	$3x10^9$	3.16	0.1	
	$FN_{18}Tasks = JTask_{15}^{18}$	$3x10^9$	1.00	0.1	
	$FN_{19}Tasks = /$	/	/	/	
	$FN_{20}Tasks = /$	/	/	/	
	Brut Total:	49 x 10^9 instructions	53.34 seconds	3.1 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)		53.34	31	**= 84.34**

execution time along with a minimum of memory space devoted to these jobs. These results demonstrate the BLA effectiveness in solving the job scheduling problem in the fog computing environment. This superior performance is due to the diversity of this proposal which is globally optimized, ensured by the crossover operation. In addition, the proposed bee life algorithm guarantees escaping from the local optima escape when executing the mutation operation. In addition, the greedy approach takes advantage of the search history realized by the local search, which helps to achieve a rapid convergence toward the optimal solution.

6. Conclusion and future work

Fog computing is expected to continue to attract the attention of researchers in academia and industry given its tremendous potential in which computing resources and services are distributed in efficient fog nodes reside at the edge of the cloud computing network.

In this work, we focused on the job scheduling problem in the fog computing environment to ensure the efficient execution of tasks and satisfy the service requests of mobile users. To address the job scheduling challenge for fog computing, we have proposed a new optimization method called Bees Life Algorithm inspired by nature life of bees. Our proposal is a population-based approach based on collaborative behaviors of the individuals of the population. BLA closely imitates the two main behaviors of bees namely the marriage (the reproduction) and the food source searching.

To address the job scheduling problem in fog computing, we used two performance evaluation metrics in this study namely, the CPU execution time and the total amount of memory (allocated memory) needed by all tasks expected to be executed in the fog computing infrastructure.

In order to evaluate the reliability and the efficiency of our proposed bees life algorithm, we performed a set of simulation tests on an implementation of BLA and compared the results obtained against those obtained with the conventional GA and PSO. The results we obtained with BLA demonstrate its efficiency and performance in terms of the execution time and the allocated memory.

In the future, we will investigate the implementation of a dynamic job scheduling approach that considers the arrival of new requests while the other requests are being executed for the fog computing environment. We also plan to investigate another dynamic aspect of job scheduling when the fog servers are mobile. Additionally, we will consider optimization of the network bandwidth because this metric could improve the overall end-to-end processing performance of the underlying fog computing infrastructure.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Aazam, M., and E. N. Huh. 2015. "Fog Computing Micro Datacenter Based Dynamic Resource Estimation and Pricing Model for Iot." *IEEE 29th International Conference on Advanced Information Networking and Applications (AINA)*, 687–694.

Abdi, S., S. A. Motamedi, and S. Sharifian, 2014. "Task Scheduling Using Modified PSO Algorithm in Cloud Computing Environment." *International Conference on Machine Learning, Electrical and Mechanical Engineering* 8–9.

Akbari, R., A. Mohammadi, and K. Ziarati. 2010. "A Novel Bee Swarm Optimization Algorithm for Numerical Function Optimization." *Communications in Nonlinear Science and Numerical Simulation* 15 (10): 3142–3155. doi:10.1016/j.cnsns.2009.11.003.

Anagnostopoulos, I., S. Zeadally, and E. Exposito. 2016. "Handling Big Data: Research Challenges and Future Directions." *The Journal of Supercomputing* 72 (4): 1494–1516.

Bitam, S., M. Batouche, and E-G. Talbi. 2010. "A Survey on Bee Colony Algorithms." *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on. IEEE*, 2010.

Bitam, S., A. Mellouk, and S. Zeadally. 2015. "Bio-Inspired Routing Algorithms Survey for Vehicular Ad Hoc Networks." *IEEE Communications Surveys & Tutorials* 17 (2): 843–867. doi:10.1109/COMST.2014.2371828.

Bonomi, F., R. Milito, P. Natarajan, and J. Zhu. 2014. "Fog Computing: A Platform for Internet of Things and Analytics." In *Big Data and Internet of Things: A Roadmap for Smart Environments*, Springer International Publishing, Switzerland, 169–186.
Botta, A., W. De Donato, V. Persico, and A. Pescapé. 2014. "On the Integration of Cloud Computing and Internet of Things." *IEEE International Conference on Future Internet of Things and Cloud (FiCloud)*, 23–30.
Cao, J., K. Hwang, K. Li, and A. Y. Zomaya. 2013. "Optimal Multiserver Configuration for Profit Maximization in Cloud Computing." *IEEE Transactions on Parallel and Distributed Systems* 24 (6): 1087–1096. doi:10.1109/TPDS.2012.203.
Cardellini, V., V. Grassi, F. L. Presti, and M. Nardelli. 2015. "On Qos-Aware Scheduling of Data Stream Applications over Fog Computing Infrastructures." *IEEE Symposium on Computers and Communication (ISCC)*, 271–276.
Chen, J., Q. Yu, B. Chai, Y. Sun, Y. Fan, and X. S. Shen. 2015. "Dynamic Channel Assignment for Wireless Sensor Networks: A Regret Matching Based Approach." *IEEE Transactions on Parallel and Distributed Systems* 26 (1): 95–106. doi:10.1109/TPDS.2014.2307868.
Chen, M., C. Ling, and W. Zhang. 2011. "Analysis of Augmented Reality Application Based on Cloud Computing." *4th International Congress on Image and Signal Processing (CISP)* 2: 569–572.
Cisco. IOx technical overview, Accessed October 02 2016. http://goo.gl/n2mfiw
Collette, Y., and P. Siarry. 2013. *Multiobjective Optimization: Principles and Case Studies*. Springer-verlag Berlin Heidelberg, Germany.
Dastjerdi, A. V., H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya. 2016. "Fog Computing: Principles, Architectures, and Applications." arXiv preprint arXiv: 1601.02752
Deng, R., R. Lu, C. Lai, T. H. Luan, and H. Liang. 2016. "Optimal Workload Allocation in Fog-Cloud Computing, Towards Balanced Delay and Power Consumption." *IEEE Internet of Things Journal*. doi:10.1109/JIOT.2016.2565516.
Deng, R., Z. Yang, M. Y. Chow, and J. Chen. 2015. "A Survey on Demand Response in Smart Grids: Mathematical Models and Approaches." *IEEE Transactions on Industrial Informatics* 11 (3): 570–582. doi:10.1109/TII.2015.2414719.
Díaz, M., C. Martín, and B. Rubio. 2016. "State-Of-The-Art, Challenges, and Open Issues in the Integration of Internet of Things and Cloud Computing." *Journal of Network and Computer Applications* 67: 99–117. doi:10.1016/j.jnca.2016.01.010.
Guerrero-ibanez, J. A., S. Zeadally, and J. Contreras-Castillo. 2015. "Integration Challenges of Intelligent Transportation Systems with Connected Vehicle, Cloud Computing, and Internet of Things Technologies." *IEEE Wireless Communications* 22 (6): 122–128. doi:10.1109/MWC.2015.7368833.
Fettweis, G. P. 2014. "The Tactile Internet: Applications and Challenges." *IEEE Vehicular Technology Magazine* 9 (1): 64–70. doi:10.1109/MVT.2013.2295069.
He, J., P. Cheng, L. Shi, J. Chen, and Y. Sun. 2014. "Time Synchronization in Wsns: A Maximum-Value-Based Consensus Approach." *IEEE Transactions on Automatic Control* 59 (3): 660–675. doi:10.1109/TAC.2013.2286893.
Intharawijitr, K., K. Iida, and H. Koga. 2016. "Analysis of Fog Model considering Computing and Communication Latency in 5G Cellular Networks." *IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)* 1–4.
Kuhn, H. W. 2005. "The Hungarian Method for the Assignment Problem." *Naval Research Logistics (NRL)* 52 (1): 7–21. doi:10.1002/(ISSN)1520-6750.
Li, D., and X. Sun. 2006. *Nonlinear Integer Programming*. USA: Springer-Verlag.
Ningning, S., G. Chao, A. Xingshuo, and Z. Qiang. 2016. "Fog Computing Dynamic Load Balancing Mechanism Based on Graph Repartitioning." *China Communications* 13 (3): 1,56–164. doi:10.1109/CC.2016.7445510.
Oueis, J., E. C. Strinati, and S. Barbarossa. 2015. "The Fog Balancing: Load Distribution for Small Cell Cloud Computing." *IEEE 81st Vehicular Technology Conference (VTC Spring)* 1–6.
Ozturk, C., E. Hancer, and D. Karaboga. 2015. "Dynamic Clustering with Improved Binary Artificial Bee Colony Algorithm." *Applied Soft Computing* 28: 69–80. doi:10.1016/j.asoc.2014.11.040.
Shojafar, M., S. Javanmardi, S. Abolfazli, and N. Cordeschi. 2015. "FUGE: A Joint Meta-Heuristic Approach to Cloud Job Scheduling Algorithm Using Fuzzy Theory and A Genetic Method." *Cluster Computing* 18 (2): 829–844. doi:10.1007/s10586-014-0420-x.
The Network. Cisco Delivers Vision of Fog Computing to Accelerate Value from Billions of Connected Devices. [Online]. Accessed October 02 2016. http://newsroom.cisco.com/press-release-content?articleId=1334100. M.,
Xu, Y., K. Li, J. Hu, and K. Li. 2014. "A Genetic Algorithm for Task Scheduling on Heterogeneous Computing Systems Using Multiple Priority Queues." *Information Sciences* 270: 255–287. doi:10.1016/j.ins.2014.02.122.
Yuce, B., M. S. Packianather, E. Mastrocinque, D. T. Pham, and A. Lambiase. 2013. "Honey Bees Inspired Optimization Method: The Bees Algorithm." *Insects* 4 (4): 646–662. doi:10.3390/insects4040646.

Appendix

In this appendix, the detailed results of job scheduling of the cases 5, 10 and 15 fog nodes are presented in tables 6, 7 and 8, respectively.

Table 6. BLA, GA and PSO results for 5 fog nodes.

Algorithm	Task scheduling (job tasks assigned to a fog node FN_iTasks)	Number of task instructions assigned to FN_iTasks	CPU execution time of a fog node (task instructions /clock rate)	Allocated memory (gigabytes)	Total
BLA	$FN_1Tasks = JTask^1_{24}, JTask^1_{44}$	$2x10^9$, $3x10^9$	4.00	0.2, 0.2	
	$FN_2Tasks = JTask^2_{14}, JTask^2_{15}, JTask^2_{22}, JTask^2_{23}, JTask^2_{32}, JTask^2_{33}, JTask^2_{52}$	$2x10^9$, $3x10^9$, $1x10^9$, $2x10^9$, $3x10^9$, $1x10^9$, $1x10^9$	13.00	0.3, 0.1, 0.3, 0.2, 0.3, 0.2, 0.2	
	$FN_3Tasks = JTask^3_{11}, JTask^3_{25}, JTask^3_{35}$	$2x10^9$, $3x10^9$, $1x10^9$	7.22	0.2, 0.1, 0.2	
	$FN_4Tasks = JTask^4_{12}, JTask^4_{21}, JTask^4_{34}, JTask^4_{41}, JTask^4_{42}, JTask^4_{43}, JTask^4_{45}, JTask^4_{51}, JTask^4_{55}$	$2x10^9$, $3x10^9$, $1x10^9$, $1x10^9$, $2x10^9$, $1x10^9$, $1x10^9$, $1x10^9$, $2x10^9$	14.00	0.1, 0.1, 0.1, 0.3, 0.1, 0.2, 0.2, 0.2, 0.3	
	$FN_5Tasks = JTask^5_{13}, JTask^5_{31}, JTask^5_{53}, JTask^5_{54}$	$3x10^9$, $2x10^9$, $3x10^9$, $3x10^9$	13.25	0.1, 0.2, 0.1, 0.1	
	Brut Total:	49 x 10^9 instructions	51.47 seconds	1.2 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)		51.47	12	**= 63.47**
GA	$FN_1Tasks = JTask^1_{13}, JTask^1_{14}, JTask^1_{34}, JTask^1_{43}, JTask^1_{53}$	$3x10^9$, $2x10^9$, $1x10^9$, $1x10^9$, $3x10^9$	8.00	0.1, 0.3, 0.1, 0.2, 0.1	
	$FN_2Tasks = JTask^2_{11}, JTask^2_{31}, JTask^2_{41}, JTask^2_{51}, JTask^2_{52}$	$2x10^9$, $2x10^9$, $1x10^9$, $1x10^9$, $1x10^9$	7.00	0.2, 0.2, 0.3, 0.2, 0.2	
	$FN_3Tasks = JTask^3_{24}, JTask^3_{25}, JTask^3_{33}, JTask^3_{35}, JTask^3_{44}, JTask^3_{45}$	$2x10^9$, $3x10^9$, $1x10^9$, $1x10^9$, $3x10^9$, $1x10^9$	13.25	0.2, 0.1, 0.2, 0.2, 0.2, 0.2	
	$FN_4Tasks = JTask^4_{12}, JTask^4_{23}$	$2x10^9$, $2x10^9$	4.00	0.1, 0.2	
	$FN_5Tasks = JTask^5_{15}, JTask^5_{21}, JTask^5_{22}, JTask^5_{32}, JTask^5_{42}, JTask^5_{54}, JTask^5_{55}$	$3x10^9$, $3x10^9$, $1x10^9$, $3x10^9$, $2x10^9$, $3x10^9$, $2x10^9$	20.48	0.1, 0.1, 0.3, 0.3, 0.1, 0.1, 0.3	
	Brut Total:	49 x 10^9 instructions	52.73 seconds	1.3 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)		52.73	13	**= 65.73**
PSO	$FN_1Tasks = JTask^1_{52}, JTask^1_{41}, JTask^1_{35}, JTask^1_{13}$	$1x10^9$, $1x10^9$, $1x10^9$, $3x10^9$	4.80	0.2, 0.3, 0.2, 0.1	
	$FN_2Tasks = JTask^2_{43}, JTask^2_{44}, JTask^2_{31}, JTask^2_{24}, JTask^2_{54}$	$1x10^9$, $3x10^9$, $2x10^9$, $2x10^9$, $3x10^9$	11.00	0.2, 0.2, 0.2, 0.2, 0.1	
	$FN_3Tasks = JTask^3_{45}, JTask^3_{14}, JTask^3_{22}, JTask^3_{11}, JTask^3_{15}, JTask^3_{33}$	$1x10^9$, $2x10^9$, $1x10^9$, $2x10^9$, $3x10^9$, $1x10^9$,	13.25	0.2, 0.3, 0.3, 0.2, 0.1, 0.2	
	$FN_4Tasks = JTask^4_{21}, JTask^4_{12}, JTask^4_{55}, JTask^4_{53}, JTask^4_{34}, JTask^4_{23}$	$3x10^9$, $2x10^9$, $2x10^9$, $3x10^9$, $1x10^9$, $2x10^9$	4.00	0.1, 0.1, 0.3, 0.1, 0.1, 0.2	
	$FN_5Tasks = JTask^5_{51}, JTask^5_{25}, JTask^5_{32}, JTask^5_{42}$	$1x10^9$, $3x10^9$, $3x10^9$, $2x10^9$	20.48	0.2, 0.1, 0.3, 0.1	
	Brut Total:	49 x 10^9 instructions	51.69 seconds	1.4 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)		51.69	14	**= 65.69**

Table 7. BLA, GA and PSO results for 10 fog nodes.

Algorithm	Task scheduling (job tasks assigned to a fog node $FN_i Tasks$)	Number of task instructions assigned to $FN_i Tasks$	CPU execution time of a fog node (task instructions /clock rate)	Allocated memory (gigabytes)	Total
BLA	$FN_1 Tasks = /$	/	/	/	
	$FN_2 Tasks = JTask^2_{22}, JTask^2_{34}$	$1x10^9$, $1x10^9$	2.00	0.3, 0.1	
	$FN_3 Tasks = JTask^3_{14}, JTask^3_{25}, JTask^3_{54}$	$2x10^9$, $3x10^9$, $3x10^9$	9.63	0.3, 0.1, 0.1	
	$FN_4 Tasks = JTask^4_{31}, JTask^4_{35}, JTask^4_{42}, JTask^4_{51}, JTask^4_{52}$	$2x10^9$, $1x10^9$, $2x10^9$, $1x10^9$, $1x10^9$	7.00	0.2, 0.2, 0.3, 0.2, 0.2	
	$FN_5 Tasks = JTask^5_{21}$	$3x10^9$	3.61	0.1	
	$FN_6 Tasks = JTask^6_{15}, JTask^6_{44}$	$3x10^9$, $3x10^9$	4.80	0.1, 0.2	
	$FN_7 Tasks = JTask^7_{11}, JTask^7_{33}, JTask^7_{41}, JTask^7_{45}$	$2x10^9$, $1x10^9$, $1x10^9$, $1x10^9$	5.55	0.2, 0.2, 0.3, 0.2	
	$FN_8 Tasks = JTask^8_{32}, JTask^8_{43}, JTask^8_{53}, JTask^8_{55}$	$3x10^9$, $1x10^9$, $3x10^9$, $2x10^9$	11.68	0.3, 0.2, 0.1, 0.3	
	$FN_9 Tasks = /$	/	/	/	
	$FN_{10} Task = JTask^{10}_{12}, JTask^{10}_{13}, JTask^{10}_{23}, JTask^{10}_{24}$	$2x10^9$, $3x10^9$, $2x10^9$, $2x10^9$	9.00	0.1, 0.1, 0.2, 0.2	
	Brut Total:	49 x 10^9 instructions	53.27 seconds	2.0 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)		53.27	20	**= 73.27**
GA	$FN_1 Tasks = JTask^1_{14}, JTask^1_{23}, JTask^1_{43}, JTask^1_{52}$	$2x10^9$, $2x10^9$, $1x10^9$, $1x10^9$	4.80	0.3, 0.2, 0.2, 0.2	
	$FN_2 Tasks = JTask^2_{55}$	$2x10^9$	2.00	0.3	
	$FN_3 Tasks = JTask^3_{34}, JTask^3_{54}$	$1x10^9$, $3x10^9$	4.81	0.2, 0.1	
	$FN_4 Tasks = JTask^4_{12}, JTask^4_{41}, JTask^4_{53}$	$2x10^9$, $1x10^9$, $3x10^9$	6.00	0.1, 0.3, 0.1	
	$FN_5 Tasks = JTask^5_{11}, JTask^5_{42}$	$2x10^9$, $2x10^9$	4.81	0.2, 0.1	
	$FN_6 Tasks = JTask^6_{51}$	$1x10^9$	0.80	0.2	
	$FN_7 Tasks = JTask^7_{25}, JTask^7_{31}, JTask^7_{44}$	$3x10^9$, $2x10^9$, $3x10^9$	8.88	0.1, 0.2, 0.2	
	$FN_8 Tasks = JTask^8_{13}, JTask^8_{22}, JTask^8_{24}, JTask^8_{45}$	$3x10^9$, $1x10^9$, $2x10^9$, $1x10^9$	9.09	0.1, 0.3, 0.2, 0.2	
	$FN_9 Tasks = /$	/	/	/	
	$FN_{10} Task = JTask^{10}_{15}, JTask^{10}_{21}, JTask^{10}_{32}, JTask^{10}_{33}, JTask^{10}_{35}$	$3x10^9$, $3x10^9$, $3x10^9$, $1x10^9$, $1x10^9$	11.00	0.1, 0.1, 0.3, 0.2, 0.2	
	Brut Total:	49 x 10^9 instructions	55.22 seconds	2.3 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)	36.9	55.22	23	**= 78.22**
PSO	$FN_1 Tasks = JTask^1_{23}, JTask^1_{21}$	$2x10^9$, $3x10^9$	4.00	0.2, 0.1	
	$FN_2 Tasks = /$	/	/	/	
	$FN_3 Tasks = JTask^3_{12}, JTask^3_{55}$	$2x10^9$, $2x10^9$	4.81	0.1, 0.3	
	$FN_4 Tasks = JTask^4_{45}, JTask^4_{11}$	$1x10^9$, $2x10^9$	3.00	0.2, 0.2	
	$FN_5 Tasks = JTask^5_{34}$	$1x10^9$	1.20	0.1	
	$FN_6 Tasks = JTask^6_{31}, JTask^6_{41}, JTask^6_{51}$	$2x10^9$, $1x10^9$, $1x10^9$	3.20	0.2, 0.3, 0.2	
	$FN_7 Tasks = JTask^7_{32}, JTask^7_{44}, JTask^7_{24}$	$3x10^9$, $3x10^9$, $2x10^9$	8.88	0.3, 0.2, 0.2	
	$FN_8 Tasks = JTask^8_{52}, JTask^8_{15}, JTask^8_{14}, JTask^8_{43}, JTask^8_{25}, JTask^8_{42}, JTask^8_{13}, JTask^8_{53}$	$1x10^9$, $3x10^9$, $2x10^9$, $1x10^9$, $3x10^9$, $2x10^9$, $3x10^9$, $3x10^9$	23.37	0.2, 0.1, 0.3, 0.2, 0.1, 0.1, 0.1, 0.1	
	$FN_9 Tasks = JTask^9_{54}, JTask^9_{33}, JTask^9_{35}$	$3x10^9$, $1x10^9$, $1x10^9$	5.50	0.1, 0.2, 0.2	
	$FN_{10} Task = JTask^{10}_{22}$	$1x10^9$	1.00	0.3	
	Brut Total:	49 x 10^9 instructions	53.99 seconds	2.2 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)		53.99	22	**= 75.99**

Table 8. BLA, GA and PSO results for 15 fog nodes.

Algorithm	Task scheduling (job tasks assigned to a fog node FN_iTasks)	Number of task instructions assigned to FN_iTasks	CPU execution time of a fog node (task instructions /clock rate)	Allocated memory (gigabytes)	Total
BLA	$FN_1Tasks = JTask^1_{14}, JTask^1_{25}, JTask^1_{32}, JTask^1_{41}$	$2x10^9$, $3x10^9$, $3x10^9$, $1x10^9$	7.20	0.3, 0.1, 0.3, 0.3	
	$FN_2Tasks = JTask^2_{15}, JTask^2_{22}, JTask^2_{51}, JTask^2_{55}$	$3x10^9$, $1x10^9$, $1x10^9$, $2x10^9$	7.00	0.1, 0.3, 0.2, 0.3	
	$FN_3Tasks = JTask^3_{31}$	$2x10^9$	2.40	0.2	
	$FN_4Tasks = JTask^4_{24}, JTask^4_{34}, JTask^4_{54}$	$2x10^9$, $1x10^9$, $3x10^9$	6.00	0.2, 0.1, 0.1	
	$FN_5Tasks = JTask^5_{11}$	$2x10^9$	2.40	0.2	
	$FN_6Tasks = JTask^6_{35}$	$1x10^9$	0.80	0.2	
	$FN_7Tasks = JTask^7_{21}, JTask^7_{33}$	$3x10^9$, $1x10^9$	4.44	0.1, 0.2	
	$FN_8Tasks = JTask^8_{12}, JTask^8_{43}, JTask^8_{51}$	$2x10^9$, $1x10^9$, $1x10^9$	5.19	0.1, 0.2, 0.2	
	$FN_9Tasks = JTask^9_{45}, JTask^9_{53}$	$1x10^9$, $3x10^9$	3.60	0.2, 0.1	
	$FN_{10}Task = JTask^{10}_{42}$	$2x10^9$	2.00	0.1	
	$FN_{11}Tasks = /$	/	/	/	
	$FN_{12}Tasks = /$	/	/	/	
	$FN_{13}Tasks = JTask^{13}_{23}$	$2x10^9$	2.00	0.2	
	$FN_{14}Tasks = JTask^{14}_{13}$	$3x10^9$	3.33	0.1	
	$FN_{15}Tasks = JTask^{15}_{44}$	$3x10^9$	2.40	0.1	
	Brut Total:	49 x 10^9 instructions	51.16 seconds	2.5 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)		51.16	25	**= 76.16**
GA	$FN_1Tasks = JTask^1_{24}, JTask^1_{53}$	$2x10^9$, $3x10^9$	4.00	0.2, 0.1	
	$FN_2Tasks = JTask^2_{35}$	$1x10^9$	1.00	0.2	
	$FN_3Tasks = JTask^3_{43}, JTask^3_{55}$	$1x10^9$, $2x10^9$	3.61	0.2, 0.3	
	$FN_4Tasks = /$	/	/	/	
	$FN_5Tasks = JTask^5_{12}, JTask^5_{33}$	$2x10^9$, $1x10^9$	3.61	0.1, 0.2	
	$FN_6Tasks = /$	/	/	/	
	$FN_7Tasks = JTask^7_{25}$	$3x10^9$	3.33	0.1	
	$FN_8Tasks = JTask^8_{14}, JTask^8_{31}, JTask^8_{34}, JTask^8_{54}$	$2x10^9$, $2x10^9$, $1x10^9$, $3x10^9$	10.38	0.3, 0.2, 0.1, 0.1	
	$FN_9Tasks = JTask^9_{13}, JTask^9_{32}, JTask^9_{52}$	$3x10^9$, $3x10^9$, $1x10^9$	6.30	0.1, 0.3, 0.2	
	$FN_{10}Tasks = /$	/	/	/	
	$FN_{11}Task = JTask^{11}_{11}, JTask^{11}_{42}, JTask^{11}_{44}, JTask^{11}_{51}$	$2x10^9$, $2x10^9$, $3x10^9$, $1x10^9$	10.38	0.2, 0.1, 0.2, 0.2	
	$FN_{12}Tasks = JTask^{12}_{15}, JTask^{12}_{23}$	$3x10^9$, $2x10^9$	4.50	0.1, 0.2	
	$FN_{13}Tasks = JTask^{13}_{41}$	$1x10^9$	1.00	0.3	
	$FN_{14}Tasks = JTask^{14}_{21}, JTask^{14}_{22}, JTask^{14}_{45}$	$3x10^9$, $1x10^9$, $1x10^9$	5.55	0.1, 0.3, 0.2	
	$FN_{15}Tasks = /$	/	/	/	
	Brut Total:	49 x 10^9 instructions	53.66 seconds	2.6 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)	36.9	53.66	26	**= 79.66**

(*Continued*)

Table 8. (Continued).

Algorithm	Task scheduling (job tasks assigned to a fog node FN_iTasks)	Number of task instructions assigned to FN_iTasks	CPU execution time of a fog node (task instructions /clock rate)	Allocated memory (gigabytes)	Total
PSO	$FN_1Tasks = JTask^1_{45}$	$1x10^9$	0.80	0.2	
	$FN_2Tasks = /$	/	/	/	
	$FN_3Tasks = /$	/	/	/	
	$FN_4Tasks = /$	/	/	/	
	$FN_5Tasks = JTask^5_{44}, JTask^5_{24}$	$3x10^9$, $2x10^9$	6.02	0.2, 0.2	
	$FN_6Tasks = JTask^6_{25}$	$3x10^9$	2.40	0.1	
	$FN_7Tasks = /$	/	/	/	
	$FN_8Tasks = JTask^8_{14}, JTask^8_{52}, JTask^8_{11}, JTask^8_{15}, JTask^8_{34}, JTask^8_{42}$	$2x10^9$, $1x10^9$, $2x10^9$, $3x10^9$, $1x10^9$, $2x10^9$	14.28	0.3, 0.1, 0.2, 0.1, 0.1, 0.1	
	$FN_9Tasks = JTask^9_{32}, JTask^9_{33}$	$3x10^9$, $1x10^9$	3.60	0.3, 0.2	
	$FN_{10}Task = JTask^{10}_{21}$	$3x10^9$	3.00	0.1	
	$FN_{11}Tasks = JTask^{11}_{31}, JTask^{11}_{23}, JTask^{11}_{54}$	$2x10^9$, $2x10^9$, $3x10^9$	10.00	0.2, 0.2, 0.1	
	$FN_{12}Tasks = JTask^{12}_{22}, JTask^{12}_{35}$	$1x10^9$, $1x10^9$	1.80	0.3, 0.2	
	$FN_{13}Tasks = JTask^{13}_{43}, JTask^{13}_{13}, JTask^{13}_{41}$	$1x10^9$, $3x10^9$, $1x10^9$	5.00	0.2, 0.1, 0.3	
	$FN_{14}Tasks = JTask^{14}_{55}, JTask^{14}_{12}$	$2x10^9$, $2x10^9$	4.44	0.3, 0.1	
	$FN_{15}Tasks = JTask^{15}_{51}, JTask^{15}_{53}$	$1x10^9$, $3x10^9$	3.20	0.2, 0. 1	
	Brut Total:	49 x 10^9 instructions	53.65 seconds	2.5 GBytes	
	Weighted Total: (W1 x ET), (W2 x M)		53.65	25	**= 78.65**