International Journal of Progressive Sciences and Technologies (IJPSAT)
ISSN: 2509-0119.

http://ijpsat.ijsht-journals.org
Vol. 30 No. 1 December 2021, pp.205-219

Multi-Criteria Aware Task Scheduling Algorithm Based on PSO For Fog Computing

Miradontsoa Asafa Andrianarison, Andry Auguste Randriamitantsoa, Paul Auguste Randriamitantsoa

Laboratoire de Recherche Télécommunication, Automatique, Signal et Images
Ecole Doctorale en Sciences et Techniques de l'Ingénierie et de l'Innovation
Université d'Antananarivo, Madagascar

Abstract—In this paper, we propose a task scheduling method for applications made of independent tasks executed on a fog computing network. Fog computing is a computing network architecture extending the cloud computing model. It adds an intermediate computational layer between IoT devices and the cloud to solve the computational requirements of the current IoT era. This fog layer or fog network uses resources from network infrastructures such as base stations, routers, switches, or servers. These resources, also called fog nodes, have computational capabilities so that they can process tasks, and we consider that they have additional properties such as link characteristics or security threat levels. Task scheduling consists of the attribution of the tasks of applications coming from IoT devices to the resources of the fog network, according to a predefined objective. However, finding optimal task scheduling solutions is a np-hard problem, so that is the reason we use a heuristic approach in our research. The paper presents a task scheduling method for fog computing based on particle swarm optimization (PSO). This scheduling method considers several criteria defined as objectives. These criteria are the makespan, the cost of processing, the cost of using network links, and the security cost involved in the processing of the application. Then, we evaluate the performance of the provided scheduling method in a simulated fog network environment and evaluate the impact of a specific criterion.

Keywords—fog computing; task scheduling algorithm; makespan; multi-cost; particle swarm optimization (PSO).

I. INTRODUCTION

Today, large numbers of Internet of Things (Internet of Thing) devices are connected to the Internet. These objects use the Internet, also called the Cloud, to receive and transfer data and also to process calculations or to delegate complex and resource-intensive tasks. This is the basic principle of Cloud Computing. However, it is estimated that 50 billion devices are connected to the Internet in 2020 and this number will reach 1 trillion in 2025. The introduction of such a large number of connected objects requires a flexible and extensible architecture, in order to guarantee the qualities of services required by their applications [1]. Fog Computing is an extension of the cloud computing model used to address these demands and constraints of IoT [2]. It generally consists of providing computational (or storage) resources between the Cloud (the Internet) and end user devices. These devices send application execution requests as a set of tasks that will be processed. These tasks will then be processed by the resources of the fog network or by the Cloud. These resources have technical characteristics such as the computing power or the bandwidth of the link, and additional characteristics such as their level of vulnerability [3] or the costs generated by their use. In this article, we propose a method of distributing these tasks on the available resources of the fog network. This method is based on the particle swarm optimization (PSO) algorithm.

Corresponding Author: Miradontsoa Asafa Andrianarison

II. PROBLEM FORMULATION

In this section, we define the main characteristics of fog computing and we develop the constraints that led us to propose a task scheduling method for fog computing.

A. Fog computing architecture

The architecture of the fog computing system (Fig. 1) consists mainly of three levels of logical layers: the object layer, the fog layer, and the cloud layer. [2][4].

The objects layer encompasses all end devices connected to the Fog. It includes IoT devices such as sensors, actuators, mobile devices, and others. These objects have limited performance (limited power, memory or autonomy) and it is for this reason that heavy tasks or applications are processed in the fog or the cloud[5]. In this article, we refer to user equipment (UE) as the device belonging to this layer. It is these UEs which generate the tasks to be executed.

The fog layer is made up of intermediate network devices located between the Objects layer and the Cloud layer. It is made up of several computation nodes for storage and processing, and maintains communication between these two layers. In this paper, we call these nodes the fog nodes (FN). These FNs offer their computing resources to the object layer [6]. Thus, compared to the cloud, the FNs of the fog layer are closer to the user equipment, and this is one of the main advantages of implementing the fog layer of the fog computing model.

Typical cloud infrastructures reside in the cloud layer. The main functionality of this layer is to provide a long-term storage system and to offer computing and complex processing resources to network users. We call a cloud resource (CR) a cloud layer computing node.

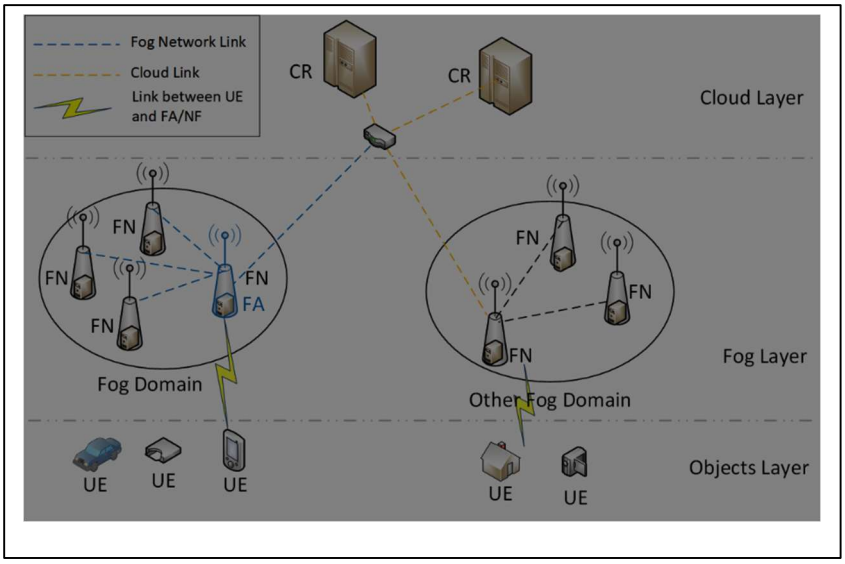

Fig. 1. Fog computing architecture.

B. Fog network

We call fog network, the network of available fog computing resources. In this article, we consider a fog network in star topology, so that the management of fog and cloud resources is centralized (Fig. 1).

A fog node, called fog administrator (FA) is responsible for the task scheduling of applications coming from the UE, namely the distribution of tasks between the different resources according to the characteristics of these resources and the requirements of the applications. We call resource a computing node. A resource can represent a fog node (FN) or a cloud resource node (CR). In fog networks, these resources are managed by FAs which can themselves be FNs. FN resources managed by the same FA belong to the same fog domain (FD).

A fog network $Fnet$ presented in (1) is formed by all the NF or CR resources connected to an FA. Thus, this network is made up of m different resources $R_i (i \in \{1, \dots, m\})$, linked to an FA.

$$Fnet = \{R_1, R_2, \dots, R_m\} \quad (1)$$

In the fog layer, there can be several fog domains. We consider that if an FA of a fog domain wants to access the resources of another fog domain, it must pass its request through the cloud; thus, we can theoretically consider this other fog domain as a simple cloud resource.

A resource R_i has the characteristics presented by (2).

$$R_i = \left(P_i^{CPU}, B_i, t_i^{lat}, CM_i^{CPU}, CM_i^{Link}, CM_i^{Security}\right) \quad (2)$$

P_i^{CPU} represents the computational speed of its processor (Central processing unit or CPU), expressed in ips (instructions per second); B_i represents the bandwidth of the link between R_i and the FA, expressed in bits per second; and t_i^{lat} represents the latency on this link, expressed in seconds.

Regarding the cost models, CM_i^{CPU} represents the model of the cost of processing by the resource CPU according to the time of use, CM_i^{Link} represents the model of the cost of using the communication link according to the size of the data exchanged, and $CM_i^{Security}$ represents the cost of exposure to threats on the resource according to the time of use [3]. What we call cost may represent a monetary cost (in dollars) or some other system of evaluation of use, such as the energy consumed or the vulnerability level. Equation (6) represents, for example, the energy consumed E_i^{Link} as a function of the quantity of data transferred d on an R_i-FA transmission link, where P_i represents the power required to transfer on the link and t represents the time required to transfer the data d through the link. We can consider this expression as the cost model CM_i^{Link} of the link.

$$E_i^{Link} = P_i \times t = P_i \frac{d}{B_i} = f(d) \stackrel{\text{def}}{=} CM_i^{Link} \quad (3)$$

A task to be processed T_j represented by (4) has the following characteristics: p_j the number of instructions to be executed on the resource processor; d_j^{up} the data size before processing (uplink data); and d_j^{down} the data size after processing (downlink data). We assume that these characteristics are known in advance.

$$T_j = \left(p_j, d_j^{up}, d_j^{down}\right) \quad (4)$$

To simplify the calculations of the task scheduling model presented in this article, we consider that a resource can only process a single task, and that the FA only processes one application at a time. Therefore, we do not consider the previous workload of a resource. Nevertheless, considering the queue (and the workload) of a resource would be interesting for future improvements to the presented model.

The completion time $t_i^{CPU}(p_j)$ of a task of p_j instructions by the CPU of the resource R_i is obtained by (5):

$$t_i^{CPU}(p_j) = \frac{p_j}{P_i^{CPU}} \quad (5)$$

The time $t_i^{data}(d_j)$ required to transmit data of size d_j on the R_i-FA link is expressed by (6):

$$t_i^{data}(d_j) = \frac{d_j}{B_i} + t_i^{lat} \quad (6)$$

We consider that R_i is only processing one job at a time, and k tasks $\{T_1, T_2, \dots T_k\}$ are in the queue of R_i. The CPU processing time of the j-th task $T_j$by R_i from the moment it arrives at R_i is equal to (7):

$$t_i^{CPU}(T_j) = \sum_{h=1}^{j} \frac{p_h}{P_i^{CPU}} \quad (7)$$

We consider that k tasks $\{T_1, T_2, \dots T_k\}$ of an application A are sent at the same time by the FA to R_i for processing, this set is denoted $A_k = \{T_1, T_2, \dots T_k\}$; and in a resource, processing does not begin until all of the tasks A_k of have arrived at R_i, and R_i then

processes one task at a time. The makespan $t_i^{make}(A_k)$ to complete the k tasks, from the time A_k leaves the FA until the results return to the FA is equal to (8):

$$t_i^{make}(A_k) = 2t_i^{lat} + \sum_{j=\{1,\dots,k\}}\left(\frac{p_h}{P_i^{CPU}}\right) + \max_{j=\{1,\dots,k\}}\left(\frac{d_j^{down}}{B_i} + \sum_{h=1}^{j}\frac{p_h}{P_i^{CPU}}\right) \quad (8)$$

To express the three cost models CM_i^{CPU}, CM_i^{Link} and $CM_i^{Security}$ when performing the task T_j on the resource R_i, we use the expression $Cost_i(T_j)$ of (9) which is shown in Fig. 2. This model is based on the function $Cost_i^{line}(T_j)$ of (10) to express cost as a function of the use, and has a minimum cost $Cost_i^{min}$ and a maximum cost $Cost_i^{max}$.

$$CM_i = Cost_i(T_j) = \begin{cases} Cost_i^{min} & \text{if } Cost_i^{line}(T_j) < Cost_i^{min} \\ Cost_i^{line}(T_j) & \text{if } Cost_i^{min} \le Cost_i^{line}(T_j) < Cost_{i,j}^{max} \\ Cost_i^{max} & \text{if } Cost_i^{line}(T_j) \ge Cost_i^{max} \end{cases} \quad (9)$$

$$Cost_i^{line}(T_j) = CUU_i \times u_j + Cost_i^{base} \quad (10)$$

CUU_i means cost per unit of usage ; u_j represents the usage of the resource by T_j, and is expressed in unit of time (such as second) or in bits depending on the model ; and $Cost_i^{base}$ is the base cost of R_i regardless of use.

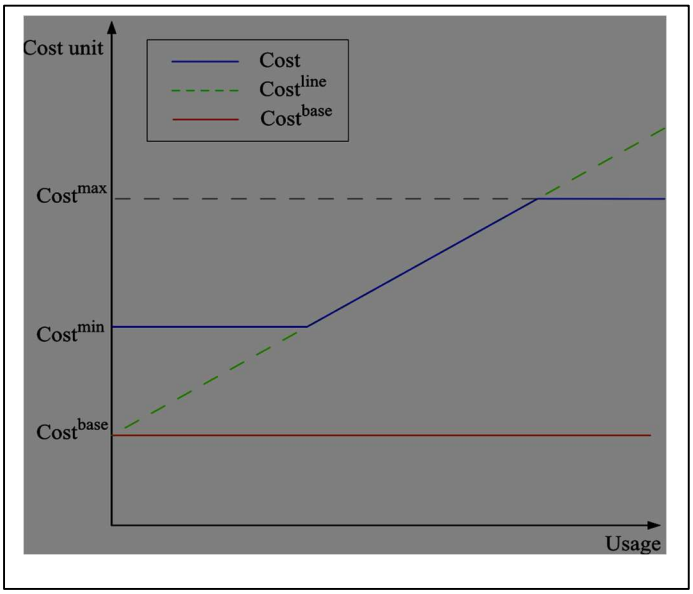

Fig. 2. Cost model representation.

Therefore, from the models CM_i^{CPU}, CM_i^{Link} and $CM_i^{Security}$, a set of tasks $A_k = \{T_1, T_2, \dots T_k\}$ processed on a resource R_i will have the costs $Cost_i^{CPU}(A_k)$, $Cost_i^{Link}(A_k)$ and $Cost_i^{Security}(A_k)$ respectively, for usages $u_i^{CPU}(A_k)$ (the overall CPU computing time of tasks of A_k on R_i), $u_i^{Link}(A_k)$ (all the data transmitted via the communication link R_i-FA), and $u_i^{Security}(A_k)$ (the entire usage time of R_i) that we defined in (11), (12) and (13):

$$u_i^{CPU}(A_k) = \sum_{h=1}^{k}\frac{p_h}{P_i^{CPU}} \quad (11)$$

$$u_i^{Link}(A_k) = \sum_{h=1}^{k}\frac{d_h^{up} + d_h^{down}}{B_i} \quad (12)$$

$$u_i^{Security}(A_k) = \sum_{h=1}^{k}\frac{p_h}{P_i^{CPU}} \quad (13)$$

C. Task scheduling

In general, the resources of the fog network (including the cloud resource) have different characteristics; i.e. they have different computing powers, different costs of use, different levels of security, or different communication link specifications.

On the other hand, applications, made up of tasks with different characteristics that will be executed on the Fog, also have different needs. Therefore, finding an ideal solution to place the different tasks equally on the different resources considering these different characteristics of the resources and the application is necessary to obtain an optimal result.

This method of scheduling and distributing tasks is called "task scheduling". It consists in the distribution of the various tasks arriving in a distributed system on the various resources participating in the computing according to defined criteria and objectives. This is one of the major problems in distributed systems such as fog computing. In this paper, we consider that task scheduling process is carried out by the FA.

Regarding the applications and the tasks that compose them, they are generally classified into two: Applications formed by interdependent tasks, i.e. which communicate with each other (generally modeled in the form of directed acyclic graph), and applications formed by tasks independent of each other called bag-of-tasks.

In this article, we consider applications made up of bag-of-tasks. An application is thus formed from a set of n parallel tasks as in (14):

$$A = \{T_1, T_2, \dots, T_n\} \tag{14}$$

A scheduling solution S_p ($p \in \{1, \dots, l\}$ for l number of scheduling solutions considered) consists of the allocation of T_j tasks from A to the m resources R_i ($i \in \{1, \dots, m\}$) of the fog or the cloud. In example (15), $T_j^{R_i}$ means that task T_j is assigned to the resource R_i:

$$S_p = \left\{T_1^{R_2}, T_2^{R_m}, \dots, T_j^{R_i}, \dots T_n^{R_a}\right\} \tag{15}$$

We denote by $A_i = \{T_a, T_b, \dots, T_k\}$ the set of tasks assigned to the resource R_i by the scheduling solution S_p. Then $t_i^{make}(A_i)$ of (8) represents the time necessary to perform all the tasks of A_i on R_i. Thus, the makespan time $t^{make}(S_p)$ required to complete all the tasks of an application A using scheduling S_p, from the moment when the FA sends the tasks from the application to the resources until the results coming from the resources are received is then obtained by (16):

$$t^{make}(S_p) = \max_{i \in \{1,\dots,m\}} t_i^{make}(A_i) \tag{16}$$

The usage costs, $Cost^{CPU}(S_p)$, $Cost^{Link}(S_p)$, and $Cost^{Security}(S_p)$ of the resources are expressed by (17), (18) and (19) using the CM_i^{CP} , CM_i^{Link}, and $CM_i^{Security}$ cost models for each resource.

$$Cost^{CPU}(S_p) = \sum_{i \in \{1,\dots,m\}} Cost_i^{CPU}(A_i) \tag{17}$$

$$Cost^{Link}(S_p) = \sum_{i \in \{1,\dots,m\}} Cost_i^{Link}(A_i) \tag{18}$$

$$Cost^{Security}(S_p) = \sum_{i \in \{1,\dots,m\}} Cost_i^{Security}(A_i) \tag{19}$$

The objective of task scheduling depends on the needs defined by the application. For example, a task scheduling mechanism can have as objective the shortest makespan of tasks regardless of the cost that this will require (whatever other costs it will result in); another task scheduling objective may require the reverse (minimal cost, whatever makespan time); and another may combine both.

Therefore, we assume that several criteria can be considered in task scheduling, and that for each criterion c^k, an importance weight w^k is assigned. In this way, a criterion score σ_p^k is assigned for the scheduling solution S_p. This allows us to assign a total score σ_p to each solution as in table I. A high score means a good solution, so if table I represents all of the scheduling solutions, then the one with the best score will be the solution to the task scheduling problem of the application A.

Table I. SCHEDULING DECISION TABLE

	Criterias				
	t^{make}	$Cost^{CPU}$	$Cost^{Link}$	$Cost^{Security}$	
Solutions	w^{make}	w^{CPU}	w^{Link}	$w^{Security}$	**Score**
S_1	σ_1^{make}	σ_1^{CPU}	σ_1^{Link}	$\sigma_1^{Security}$	σ_1
…	…	…	…	…	…
S_p	σ_p^{make}	σ_p^{CPU}	σ_p^{Link}	$\sigma_p^{Security}$	σ_p
…	…	…	…	…	…
S_l	σ_l^{make}	σ_l^{CPU}	σ_l^{Link}	$\sigma_l^{Security}$	σ_l

We consider the followig criteria: the makespan $t^{make}(S_p)$, the costs of the processing of the tasks by the resources $Cost^{CPU}(S_p)$, the costs of using the communication links $Cost^{Link}(S_p)$, and the costs of exposure to threats $Cost^{Security}(S_p)$; with their respective weights of importance w^{make}, w^{CPU}, w^{Link}, and $w^{Security}$. Regarding the time to perform a scheduling S_p, we assign a score $\sigma^{make}(S_p)$ expressed in (20). This way, if the makespan $t^{make}(S_p)$ of a solution is low, then it will have a high score. We apply the same principle for σ^{CPU}, σ^{Link}, and $\sigma^{Security}$ scores, for other criteria in (21), (22) and (23):

$$\sigma_p^{make} = \sigma^{make}(S_p) = \frac{1}{t^{make}(S_p)} \quad (20)$$

$$\sigma_p^{CPU} = \sigma^{CPU}(S_p) = \frac{1}{t^{CPU}(S_p)} \quad (21)$$

$$\sigma_p^{Link} = \sigma^{Link}(S_p) = \frac{1}{Cost^{Link}(S_p)} \quad (22)$$

$$\sigma_p^{Link} = \sigma^{Security}(S_p) = \frac{1}{Cost^{Security}(S_p)} \quad (23)$$

As the σ^k scores of the criteria defined in (20) (21), (22) and (23) have different units and different dimensions (for example: σ^{mak} is expressed in s^{-1}, and σ^{CPU} is expressed in $\$^{-1}$), we use the weighted product model [7] of expression (24) to establish the global score $\sigma(S_p)$ of each scheduling solution S_p:

$$\sigma_p = \sigma(S_p) = \prod_{kcriteria} [\sigma^k(S_p)]^{w^k}$$

$$\sigma(S_p) = \frac{1}{[t^{make}(S_p)]^{w^{make}} \times [Cost^{CPU}(S_p)]^{w^{CPU}} \times [Cost^{Link}(S_p)]^{w^{Link}} \times [Cost^{Security}(S_p)]^{w^{Security}}} \quad (24)$$

Consequently, finding the best task scheduling solution S_{best} is equivalent to determining the scheduling which has the maximum score σ_{best} in (25) among the l scheduling solutions considered:

$$\sigma_{best} = \sigma(S_{best}) = \max_{p=\{1,\dots,l\}} \sigma(S_p) \quad (25)$$

The task scheduling problem defined in (25) is of type np-hard, so we propose a heuristic approach in this article to solve it. For this we use a resolution method based on the Particle Swarm Optimization algorithm that we develop in the following section III.

III. MULTI-CRITERIA TASK SCHEDULING BASED ON PSO ALGORITHM

In this section, we present the scheduling algorithm that we use to map the tasks of an application to the available resources of the fog. For this, we use a modified version of the PSO method to find acceptable solution for the multi-criteria expression (24) and (25). First, we explain the PSO algorithm.

A. PSO Algorithm principle

PSO was first described by Kennedy and Ebernet in 1995, and a binary version (Binary PSO) was presented by these researchers in 1997 [8][9]. This heuristic algorithm mimics the interaction and communication between the elements of a grouping (swarm) of animals such as birds or insects, moving to achieve a common goal, such as finding a source of food.

In PSO, a population of q particles (swarm) is defined first. A particle $i(i \in \{1, \dots, q\})$ represents a candidate solution. Over time, or more precisely the iteration steps k of the algorithm, the positions of the particles x_i^k are in motion according to their velocity of displacement v_i^k in (26) and the rule of displacement in (27). These particles try to follow the leader *gbest* of the population (which have the best solution), and they improve their own best results *pbest*. If a particle performs better than the leader, it becomes the new *gbest* leader of the population. The operation is repeated until a certain limit *kmax* of the number of iterations is reached or a certain acceptable condition is reached.

In equations (26) and (27), the positions x_i^k and the velocities v_i^k are in D dimensions; v_i^{k+1} is the velocity of the particle i, updated for the iteration $k+1$; x_i^{k+1} is the position of the particle i; updated for iteration $k+1$; ω^k is called inertia of iteration k (value between $[\omega_{min}, \omega_{max}]$); c_1 and c_2 are constants of accelerations; and $rand_1$ and $rand_2$ are random numbers between 0 and 1:

$$v_i^{k+1} = \omega^k v_i^k + c_1 rand_1 \times \left(pbest_i - x_i^k\right) + c_2 rand_2 \times \left(gbest - x_i^k\right) \quad (26)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (27)$$

In its binary version BPSO, positions x_i^k and velocities v_i^k are represented by D dimension matrices, and their elements take the value 0 or 1. So instead of expression (27), expression (28) is used for binary conversion of the position. Here, $x_i^{k+1}(j)$ represents the value of the j-th element of the position x_i^{k+1}, and $rand_i$ is a random value between 0 and 1, specific to each particle:

$$x_i^{k+1}(j) = \begin{cases} 1 & if \ \frac{1}{\exp(-v_i^k+1)} > rand_i \\ 0 & overwise \end{cases} \quad (28)$$

To determine whether the position x_i^k of a particle i will be a *pbest* or *gbest* during iterations, the PSO algorithm uses a function, called "fitness function", to assign a score to each particle. Pseudo-algorithm 1 in Fig. 3 is then used.

At the end of the realization of the algorithm, the particle with the position *gbest* will be chosen as the best solution.

Algorithm 1: *pbest* and *gbest* assignment at each iteration:

if *fitness_function*(x_i^k) > *fitness_function*$(pbest_i)$:

    $pbest_i = x_i^k$

if *fitness_function*$(pbest_i)$> *fitness_function*$(gbest)$

    $gbest = pbest_i$

Fig. 3. Pseudo-algorithm for pbest and gbest.

B. PSO for task scheduling

In the literature, several task scheduling algorithms based on the PSO have been proposed for distributed systems such as cloud computing, grid computing, or fog computing.

Some of these algorithms are based on modified versions of the BPSO [10][11]. Other approaches use the continuous version of the PSO algorithm to be applied in sequential type problem classes using a rule called Smallest Position Value (SPV) for task

placement [12][13]. These PSO-based algorithms perform well compared to other heuristic algorithms. However, we have found that most of these algorithms use the weighted sum model as an optimization method when evaluating solutions, whether with the PSO or other scheduling algorithms such as in [12][13][14].

In this contribution, we propose a task scheduling algorithm based on the PSO combined with a decision method from the weighted product model of (25).

This allows us to consider various optimization criteria with different dimensions, and this property can thus be exploited in fog computing.

As an implementation of the PSO, our algorithm uses the continuous version of the PSO, but then converts the positions of the particles to discrete values.

Consider m fog resources and n tasks to assign to these resources. A solution S_p is represented by the expression (29). This expression is transcribed in the form of a vector of dimension n. The n elements of the vector represent the tasks and the value g of a j-th element represents the resource R_g which will process the task, as in example (29):

$$S_p = \left\{T_1^{R_a}, T_2^{R_m}, \ldots, T_j^{R_g}, \ldots T_n^{R_f}\right\} \stackrel{\text{def}}{=} [a, m, \ldots, g, \ldots, f] \quad (29)$$

In our approach, we consider that this solution S_p is equivalent to the position x_i of the particle i of the PSO. x_i is represented by a vector of size n. A particle move with a v_i velocity. v_i is also represented by a vector of size n.

1. Particle velocity

To limit the domain of values of the elements of the velocity v_i between $[-v_{max}, v_{max}]$, we modify v_i^{k+1} to $v_i^{k+1\prime}$ by (30):

$$v_i^{k+1\prime} = \text{sgn}(v_i^{k+1}) \times \left[\left|v_i^{k+1}\right| \bmod v_{max}\right] \quad (30)$$

where $\text{sgn}(v_i^{k+1})$ is the signum function to extracts the sign of v_i^{k+1}, and mod is the modulo operation.

2. Particle position

Regarding the position of the particles, on (27), we get continuous values when updating the elements of x_i^{k+1}. To get discrete values for the element x_i^{k+1}, instead of (27) we use the expression (31) to update the position of the particles:

$$x_i^{k+1} = 1 + \left[\text{round}\left(x_i^{k+1} + v_i^{k+1\prime}\right)\right] \bmod v_{max} \quad (31)$$

where round is the rounded value.

This expression (31) also allows us to limit the domain of the values of the elements of x_i^{k+1} between $[1, m]$.

3. Fitness function

We use the model $\sigma(S_p)$ of (24) as the fitness function of our PSO algorithm. This allows us to establish several criteria of different dimensions.

The pseudo-algorithm of our multi-criteria task-scheduling algorithm based on PSO is presented in Algorithm 2 (Fig. 4).

Algorithm 2: Multi-criteria task-scheduling based on PSO:

1. Define a set of *n* parallel tasks of an application $A = \{T_1, T_2, \dots, T_n\}$, to be distributed over a set of resources in the fog network $Fnet = \{R_1, R_2, \dots, R_m\}$.
2. Initialize the importance weights of the criteria w^{make}, w^{CPU}, w^{Link} and $w^{Security}$ of the task scheduling.

   Initialize the parameters $\omega_{min}, \omega_{max}, c_1, c_2$ and *kmax* of the PSO. Initialize $k = 0$.

   Initialize a population of q particles of positions x_i^0 ($x_i^0 \in [1, m], i \in \{1, \dots, q\}$) and velocity v_i^0($v_i^0 \in [-v_{max}, v_{max}], i \in \{1, \dots, q\}$). The size of the vectors x_i^k and v_i^k is equal to *n*.
3. Repeat:

   Define ω^k with $\omega^k = \omega_{max} - k\frac{\omega_{max}-\omega_{min}}{kmax}$

   For each particle $i = \{1, \dots, q\}$ do:

   a. Determine $pbest_i$ and $gbest$ using the fitness function of (24), i.e.:

   if *fitness_function*(x_i^k) > *fitness_function*($pbest_i$):

   $pbest_i = x_i^k$

   if *fitness_function*($pbest_i$)> *fitness_function*(*gbest*)

   $gbest = pbest_i$

   b. Determine the velocity v_i^{k+1} from (26) then $v_i^{k+1\prime}$ from (30)

   c. Determine the position x_i^{k+1} from (31)

   Go to next iteration *k=k+1*

   Until *k=kmax* or a stop condition is reached
4. *gbest* is chosen as the optimal solution for the task scheduling

Fig.4. Pseudo-algorithm of the multi-criteria task-scheduling based on PSO.

IV. IMPLEMENTATIONS AND RESULTS OF EXPERIMENTS

In this section, we simulate task scheduling scenarios for the applications of table II in the fog network composed of the resources described in table III. We evaluate the results of our task scheduling method according to the optimization criteria imposed in table IV.

Table II. APPLICATIONS USED DURING SIMULATION

Application	Characteristic	Simulate	Configuration
App1	Composed of tasks with large amounts of instructions and light data sizes.	AI powered application such as voice assistants, Data collection application for data analysis purposes.	Instructions: 100000Mi Input data size: 1Mb Output data size: 1Mb Number of tasks: 20
App2	Composed of tasks with large amounts of instructions and large data sizes.	Storage and processing of medical images,	Instructions: 100000Mi

		Streaming video game application	Input data size: 100Mb Output data size: 100Mb Number of tasks: 20
App3	Composed of various tasks with different amounts of instructions and different sizes of data.	Application for devices made up of several different sensors (car, smart medical device, smart-home)	Instructions: Mixed Input data size: Mixed Output data size : Mixed Number of tasks : 20

We consider that the applications coming from the UEs arrive at the FA, and it is up to the FA to perform the task scheduling. Our simulations focus on evaluating performance, from the moment an application from a UE arrives at the AF, until the result of the processing comes back to the AF after processing. Therefore, we do not consider the EU-FA and FA-EU parts.

We implemented our simulation and our task scheduling method in Python using mainly the Numpy mathematical tool and the Simpy event simulator.

Table III. FOG NETWORK RESOURCES

Resource R_i	**P_i^{CPU} (Mips)**	**B_i (Mpbs)**	**t_i^{lat} (s)**	**CM_i^{CPU}**	**CM_i^{Link}**	**$CM_i^{Security}$**
FN1: Fog node (located within FA)	100	Inf	0	$CUU = 10$ $Cost^{base} = 0$	$CUU = 0.1$ $Cost^{base} = 0$	$CUU = 5$ $Cost^{base} = 0$
FN2: Fog node	1000	100	0.01	$CUU = 20$ $Cost^{base} = 0$	$CUU = 10$ $Cost^{base} = 0$	$CUU = 10$ $Cost^{base} = 0$
FN3: A less secure fog node	1000	100	0.01	$CUU = 20$ $Cost^{base} = 0$	$CUU = 10$ $Cost^{base} = 0$	$CUU = 100$ $Cost^{base} = 0$
CR: A cloud resource	200	50	0.2	$CUU = 500$ $Cost^{base} = 200$	$CUU = 10$ $Cost^{base} = 0$	$CUU = 200$ $Cost^{base} = 100$

Table IV. TASK SCHEDULING ALGORITHM OPTIMIZATION CRITERIA

Name	Criteria weights	Details
TS Make.	$w^{make} = 1, w^{CPU} = 0, w^{Link} = 0, w^{Security} = 0.$	Optimized for makespan only
TS Proc.	$w^{make} = 0, w^{CPU} = 1, w^{Link} = 0, w^{Security} = 0.$	Optimized for processing cost only
TS Link.	$w^{make} = 0, w^{CPU} = 0, w^{Link} = 1, w^{Security} = 0.$	Optimized for link cost only
TS Secu.	$w^{make} = 0, w^{CPU} = 0, w^{Link} = 0, w^{Security} = 1.$	Optimized for security cost only
TS Make. & Proc	$w^{make} = 1, w^{CPU} = 1, w^{Link} = 0, w^{Security} = 0.$	Optimized for makespan and processing cost
TS Make. & Link	$w^{make} = 1, w^{CPU} = 0, w^{Link} = 1, w^{Security} = 0.$	Optimized for makespan and link cost
TS Make. & Secu	$w^{make} = 1, w^{CPU} = 0, w^{Link} = 0, w^{Security} = 1.$	Optimized for makespan and security cost
TS Make. & All Cost	$w^{make} = 1, w^{CPU} = 1, w^{Link} = 1, w^{Security} = 1.$	Optimized for makespan and all cost

A. Task scheduling optimized for a single criterion

First, we evaluate our heuristic task scheduling method by considering only one criterion at a time ("TS Make" or "TS Link" from table IV). We compare it to cloud-only, FA-only, or round-robin scheduling methods.

The cloud-only scheduling method assigns all the tasks to the CR; the FA-only method assigns all the tasks to the FA (FN1); and the round-robin method distributes alternately the tasks between the resources in an almost equal distribution.

Table V. APP 2 TASKS DISTRIBUTION AFTER SCHEDULING

Method	FA(FN1)	FN2	FN3	CR
TS Make.	0	7	7	6
TS Link.	20	0	0	0
Round Robin	5	5	5	5
FA only	20	0	0	0
Cloud only	0	0	0	20

Table V presents the result of the distribution of the tasks of App2 by the "TS Make" and "TS Link" algorithm, as well as that of the static methods round-robin, FA-only and cloud-only.

The results in Fig. 5 and Fig. 6 show that using an optimized task scheduling method for a defined criterion is necessary to obtain optimal performance.

In Fig. 5, the task scheduling algorithm optimized for the makespan "TS Make" performs better than the cloud-only even if the processing power of the cloud resource is twice as high as the two fog nodes FN2 and FN3. We can explain this result by the fact that

the characteristics (bandwidth and latency time) of the communication links FA-CR, FA-FN2 and FA-FN3 are different, and this affects the overall makespan if the tasks go through the FA-CR link. The effect of these link characteristics can also be seen by the slight difference between the makespans of App1 and App2 because they have different data sizes even though they have the same number of instructions.

Our results from table V showed that "TS Make" distributed tasks between FN2, FN3, and CR to obtain an optimal solution. Regarding the FA alone, the fact that the FA has low computing power penalizes this static scheduling method (even if the bandwidth of the FA-FN link is assumed to be almost infinite because it is on the same node). About the round-robin method, despite its equal distribution of tasks between resources, this method does not provide better results than the fair distribution algorithm "TS Make".

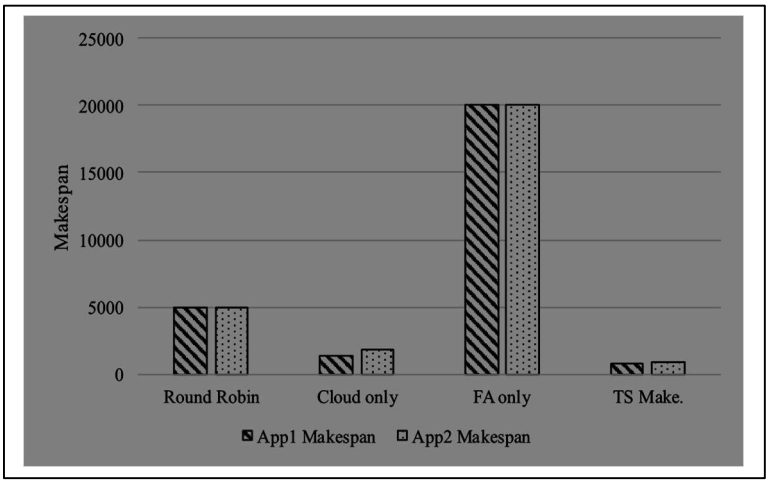

Fig. 5. App1 and App2 makespans comparison.

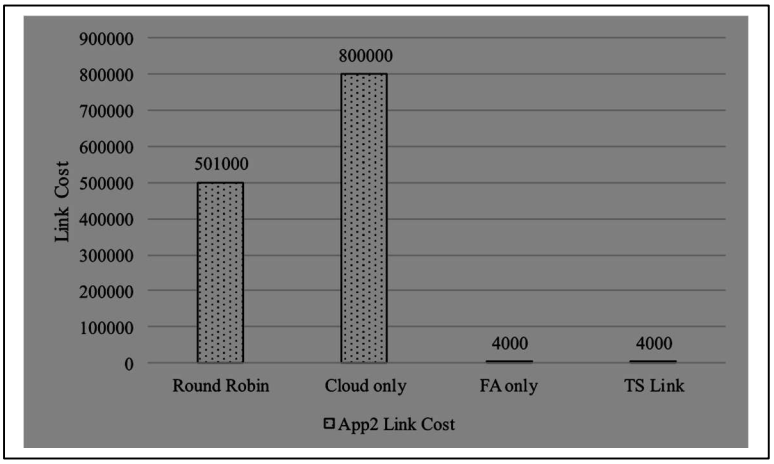

Fig. 6. Link costs comparison between different scheduling methods of App2.

Fig. 6 shows the costs of using links with different scheduling methods: round-robin, cloud-only, FA-only, and "TS Link" (task scheduling method optimized for link costs). Due to the characteristics of the links FA-FN2, FA-FN3, and particularly FA-CR, the App2, which uses a lot of data, will generate high costs with the round-robin and the cloud-only methods. The "TS Link" scheduling places all tasks on the FA, which is equivalent to the FA-only method. Even if this results in a very low link cost (4000 here), according to the results of Fig. 5, placing all the tasks on the FA will result in a very high makespan. It is for this reason that we propose the multi-criteria task scheduling method.

B. Task scheduling optimized for several criteria

We evaluate in this part the effect of using more than one optimization criterion in our heuristic scheduling method. The configurations of the criteria importance weights are those in Table IV.

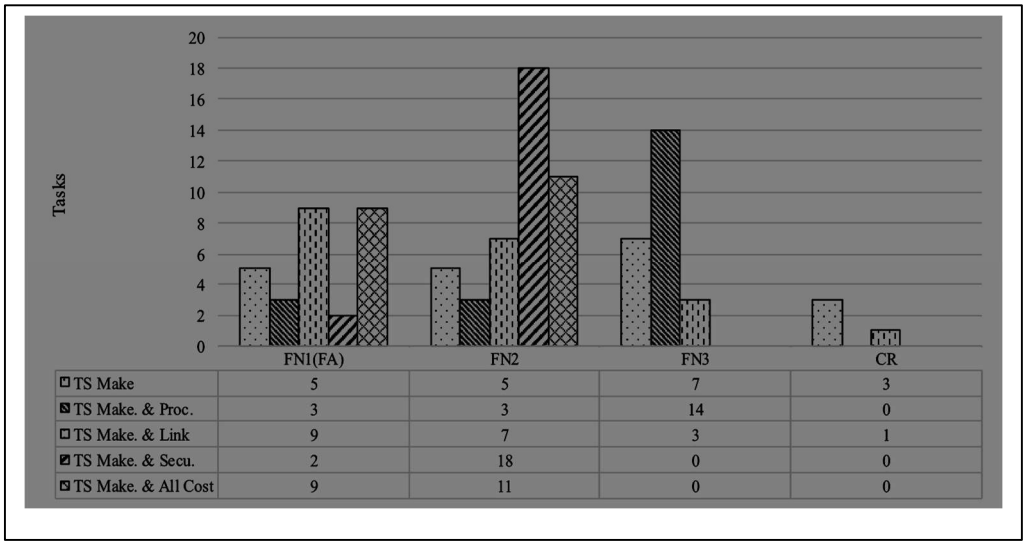

Fig. 7. App3 tasks distribution after scheduling.

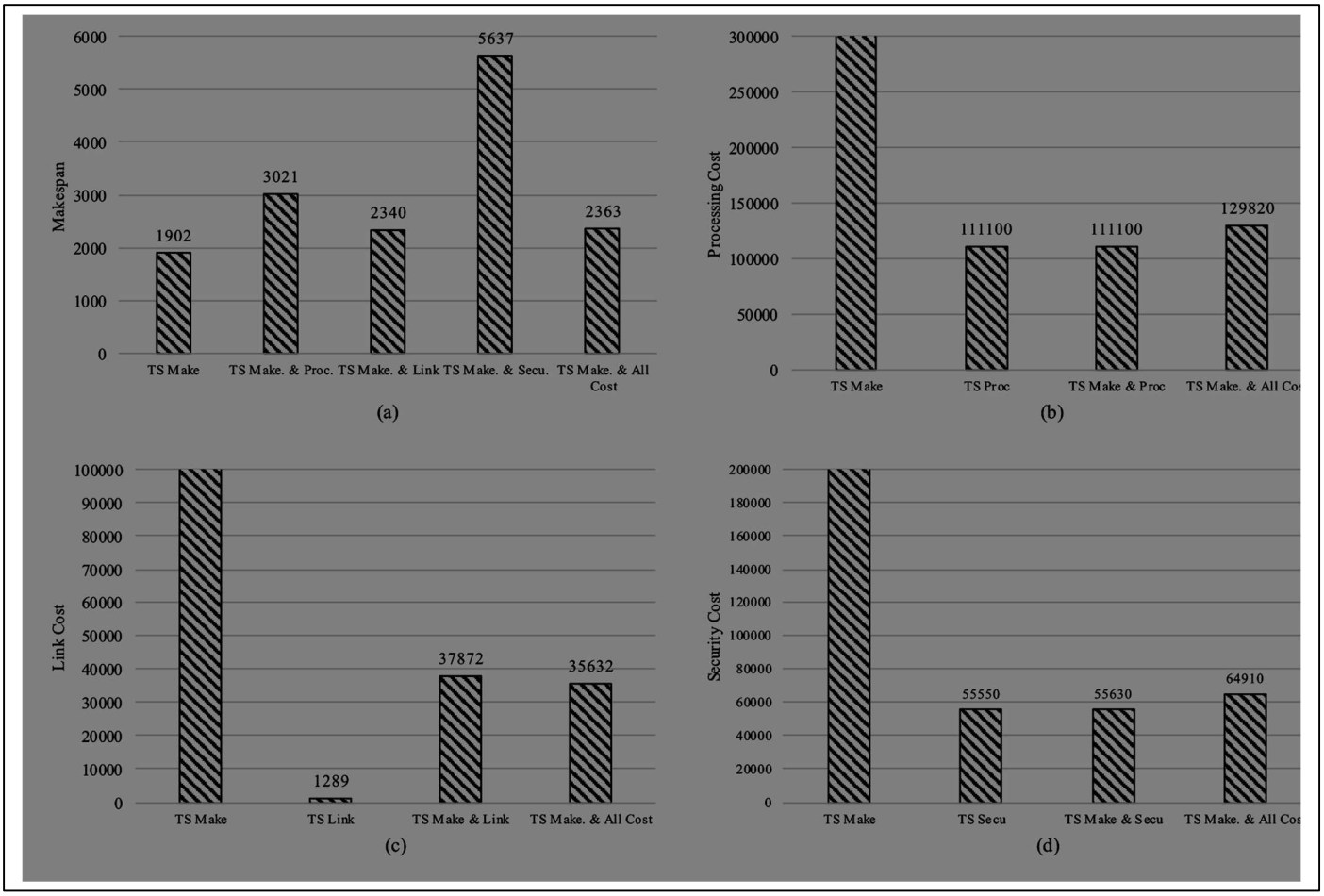

Fig. 8. App3 performance comparison of several multi-criteria task scheduling methods.

Fig. 7 shows the result of the distribution of the tasks of the App3 between the different resources. Most of the optimized task scheduling methods here do not use the CR although it has huge processing power. That is because of its high processing costs, its high threat costs, and the characteristic of the FA-CR communication link (low bandwidth and high latency). On the other hand, these algorithms send most jobs to FN2 and FN3 due to their tolerable costs, high processing power, high bandwidth, and low latency. As

we classify FN3 as less secure than FN2, the “TS Make & Secu” places most of the tasks in FN2. This effect is also seen with the “TS Make & All Cost” method. For the methods considering the cost of the links (“TS Make & Link” and “TS Make & All Cost”), as the FA-FN1 link has a very low cost, these methods distribute many tasks to FN1 despite the fact that FN1 has weak processing power.

Fig. 8 shows the performance of the defined multi-criteria task-scheduling methods for App 3. We observe from (b) (c) and (d) that if a task scheduling method is optimized for only a single criterion such as “TS Make”, it can lead to poor performance with other costs. But we see here that combining makespan with another criterion (such as “TS Make & Proc”, “TS Make & Link” or “TS Make & Secu”) allows us to get acceptable makespan and costs (relative to the criterion). We particularly note the ability of the "TS Make & All Cost" method, which considers the makespan and all the three cost models as criteria, to find an acceptable consensus result between these different criteria.

C. Varying weight of a criterion

Here we evaluate the effect of the weight of a criterion on a multi-criteria task scheduling method. For this, we consider the “TS Make & All Cost” method, and we vary w^{make} while keeping the values of w^{CPU}, w^{Link} and $w^{Security}$ fixed. We use the characteristics of the App1 but with 10, 20, and 40 tasks.

The results presented in Fig. 9 shows that for a criterion to have more impact compared to the other criteria, its weight must be greater than or equal to the sum of the weights of the other criteria. On the other hand, if the weight of a criterion is too high (twice the sum of the weights of the other criteria according to the presented experiment) it could mask the other criteria, and the task scheduling method would behave like a method with single criterion. Thus, we consider that these observations reveal need to be taken into account when fine-tuning the presented multi-criteria task scheduling method.

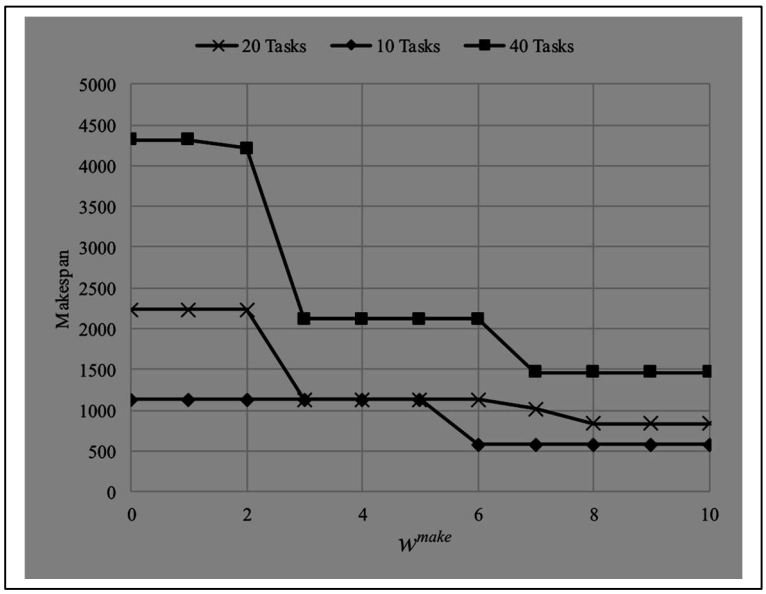

Fig. 9. Variation of makespan as a function of w^{make} using “TS Make & All costs” for App1.

V. Conclusion

In this article, we have presented a method of task scheduling for the fog computing environment. This method based on the PSO algorithm makes it possible to define several performance objectives with different dimensions at the same time, such as the time of realization, the monetary cost due to the uses of the nodes, the energy consumed by the links (presented here under the expression cost of links), and security threats (presented here under the expression security cost). For this, our method takes into account the different characteristics of the network resources, such as the computing power of the resources, the speeds and the latency times of the links, their usage cost models, and their vulnerability models. The results of our experiments show the benefits of using our multi-criteria TS over single-criterion TS methods or static methods. As future work, in this paper, we mentioned that we did not consider the previous queue or workload of a resource coming from other applications. Therefore, it is interesting to consider such parameters

in order to obtain a model that is closer to real scenario. We can also consider other heuristic resolution methods to solve our formulated problem in (25).

REFERENCES

[1] S. Ray, and Y. Jin, A. Raychowshury, "The changing computing paradigm with internet of things: a tutorial introduction," IEEE Design & Test, vol. 33, no. 2, April 2016.

[2] "OpenFog Reference Architecture for fog computing," OpenFog Consortium, February 2017.

[3] B.A. and Martin et al., "OpenFog security requirements and approaches," 2017 IEEE Fog World Congress (FWC), pp. 1-6, 2017. (doi:101109/FWC.2017.8368537)

[4] P. Hu, S. Dhelim, and H. Ning, T. Qiu, "Survey on fog computing: architecture, key technologies, applications and open issues," Journal of Network and Computer Applications, vol. 98, pp. 27-42, 2017. (doi:10.11016/j.jnca.2017.09.002)

[5] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero and M. Nemirovsky, "Key ingredients in an IoT recipe: fog computing, cloud computing, and more fog computing," 2014 IEEE 16th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), pp. 325-329. 2014. (doi:10.1109/CAMAD.2014.7033259)

[6] L.M. Vaquero, and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing". ACM SIGCOMM Comput. Communication Review, vol. 44, no. 5, pp. 27-32, October 2014. (doi:10.1145/2677046.2677052)

[7] E. Triantaphyllou, Multi-Criteria Decision Making Methods: A Comparative Study, Springer-Science+Business Media, 2000, pp.8-9.

[8] J. Kennedy, and R.C. Ebernhart, "A new optimizer using particle swarm theory," International Symposium on Micromachine and Human Science, 1995.

[9] R. C. Eberhart, and J. Kennedy, "A discrete binary version of the particle swarm algorithm," IEEE International Conference on Systems, 1997.

[10] H. Izakian, "A discrete particle swarm optimization approach for grid job scheduling," International Journal of Innovative Computing, Information and Control, 2010.

[11] B.M. Nguyen, H.T. Binh, and B.D. So, "Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud–fog computing environment," Applied Science, vol. 9, no. 9, 2019.

[12] R.M. Alguliyeva, Y. N. Imamverdiyev, and F.J. Abdullayeva, "PSO-based load balancing method in cloud computing," Automatic Control and Computer Sciences, vol. 53, pp. 45-55, 2019.

[13] M.F. Tasgetiren, M.Sevkil, Y.C. Liang, and G. Gencylmaz, "Particle swarm optimization algorithm for Single machine total weighted tardiness problem," Procedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753),vol. 2, pp. 1412-1419, 2004.

[14] L. Liu, Chang, X. Guo, S. Mao and T. Ristaniemi, "Multi-objective optimization for computation offloading in fog computing," IEEE Internet of Things Journal, vol.5, no.1, Feb. 2018.