An Evolutionary Algorithm for Solving Task Scheduling Problem in Cloud-Fog Computing Environment

Huynh Thi Thanh Binh
Hanoi University of Science and Technology
Hanoi, Vietnam
binhht@soict.hust.edu.vn

Tran The Anh
Hanoi University of Science and Techonology
Hanoi, Vietnam
anhtt17895@gmail.com

Do Bao Son
University of Transport Technology
Hanoi, Vietnam
sondb19@gmail.com

Pham Anh Duc
Hanoi University of Science and Techonology
Hanoi, Vietnam
phamanhduc070597@gmail.com

Binh Minh Nguyen
Hanoi University of Science and Technology
Hanoi, Vietnam
minhnb@soict.hust.edu.vn

ABSTRACT

Recently, IoT (Internet of Things) has grown steadily, which generates a tremendous amount of data and puts pressure on the cloud computing infrastructures. Fog computing architecture is proposed to be the next generation of the cloud computing to meet the requirements of the IoT network. One of the big challenges of fog computing is resource management and operating function, as task scheduling, which guarantees a high-performance and cost-effective service. We propose TCaS - an evolutionary algorithm to deal with Bag-of-Tasks application in cloud-fog computing environment. By addressing the tasks in this distributed system, our proposed approach aimed at achieving the optimal trade-off between the execution time and operating costs. We verify our proposal by extensive simulation with various size of data set, and the experimental results demonstrate that our scheduling algorithm outperforms 38.6% Bee Life Algorithm (BLA) in time-cost trade-off, especially, performs much better than BLA in execution time, simultaneously, satisfies user's requirement.

CCS CONCEPTS

• **Software Engineering and New Computing Trends → Mobile and Ubiquitous Computing**;

KEYWORDS

Task scheduling, fog computing, cloud computing, genetic algorithm, IoT

ACM Reference Format:
Huynh Thi Thanh Binh, Tran The Anh, Do Bao Son, Pham Anh Duc, and Binh Minh Nguyen. 2018. An Evolutionary Algorithm for Solving Task Scheduling Problem in Cloud-Fog Computing Environment. In *The Ninth International Symposium on Information and Communication Technology (SoICT 2018), December 6–7, 2018, Danang City, Viet Nam.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3287921.3287984

1 INTRODUCTION

Nowadays, Internet of things (IoT) is becoming an increasingly growing concept that not only has the potential to impact how we live but also how we work. With IoT, the Internet connection is extended beyond traditional smart devices, such as smartphones, tablets, to a myriad of devices (sensors, machines, vehicles, ...) to carry out many different services such as health care, medical treatment, traffic control, energy management, etc. Everything connected to the Internet generates a tremendous amount of data, and these data should be stored, processed and analyzed to obtain valuable information that meets the user's needs. However, IoT devices, even the most powerful smart devices, can not serve by themselves. Cloud computing environment, known as a large resource center, allowing ubiquitous access to sharing and providing resources to users in a flexible way through virtualization mechanism, seems to be a solid platform to support IoT development. Limitations of existing smart devices, such as battery life, processing power, storage capacity, network resources, can be minimized by bringing time- and resource-consuming tasks to a powerful platform which is cloud computing environment, leaving simple tasks for smart devices.

The number of IoT devices increased 31% year-over-year to 8.4 billion in the year 2017 and it is estimated that there will be 30 billion devices by 2020 [12]. With the explosion of the number of connected devices, the cloud architecture platform with traditional centralized processing characteristics, where computing and storage resources are aggregated and located in several big data centers will not be able to fulfill the IoT application's requirements of processing, storage, and communication. Moreover, IoT devices are usually very far from the cloud, which makes Internet cannot avoid the network congestion when a very huge amount of data is generated. Delay in transmission results in poor Quality of Service (QoS) when seriously affecting the user experience, while IoT applications are very sensitive in latency. It motivates an idea to extend cloud computing capabilities to the edge of network.

Consequently, Fog computing [2] was proposed, firstly by Cisco. Instead of pushing all processing operations into the cloud, many tasks now can be processed near where data is generated on network edge devices, which called fog node. Any device capable of computing, storing, and connecting to a network can be considered a fog node, such as: controllers, switches, routers, embedded servers, surveillance cameras, etc. Placing resources at the edge of the network helps data spend little time to reach the processing station, so tasks are optimized for transmission time. However, the power of fog nodes is limited, so small and sensitive latency tasks are prioritized to be processed in the fog computing layer, while the cloud still takes the vital responsibility for latency-tolerant and large-scale tasks. Eventually, fog computing complements the cloud to form a new computing paradigm, cloud-fog computing.

Cloud-fog computing has many advantages, including minimizing latency, reducing network traffic and energy efficiency, but this new paradigm is also challenging. One of them involves resource allocation and task scheduling. The highly distributed system as cloud-fog computing is an ideal platform to deploy Bag-of-Tasks (BoT) applications, which are those parallel applications whose tasks are independent to each other. BoT applications appear in many scenarios, including data mining, massive searches (such as key breaking), fractal calculations, computational biology [14], computer imaging [13], video encoding/decoding, and various other IoT applications. The main challenge is scheduling tasks in the pool of processing nodes including cloud nodes (such as servers or virtual machines) and fog nodes. The objective of task scheduling in cloud-fog system is aimed at the benefit of users or service providers. On the users side, they are concerned about some criteria of makespan, budget, deadline, security, and cost. On the other hand, the purpose of the service providers is load balancing, resource utilization, and energy efficiency. To guarantee the QoS, response time plays an important role when directly influencing the user's experience. In addition, the implementation cost is also an aspect that users are very interested. A task schedule, which minimizes completion time and saves monetary cost, will satisfy Service Level Agreement (SLA) signed with users.

In this paper, we concentrate on task scheduling problem in cloud-fog environment, a highly distributed computing platform, for processing large-scale BoT applications. A Time-Cost aware Scheduling (TCaS) algorithm is proposed to solve this problem. The major objective of the TCaS algorithm is to achieve a good trade-off between execution time and monetary cost to complete a bag of tasks in cloud-fog system. In addition, this algorithm can flexibly adapt with the requires of various users in which someone wants to prioritize the execution time at current time and others have a desire that their tasks are completed with a tight budget. Our approach is experimentally evaluated and compared with Bee Life Algorithm (BLA)[1] in many datasets with various size. The results prove that the proposed algorithm achieves better QoS and is more optimal in balance between time and cost efficiency than BLA.

The rest of the paper is divided into the following sections: in next section, the related works are presented; section 3 is a detailed model of task scheduling in cloud-fog computing environment; section 4 presents specifically proposed algorithm; the performance evaluation is shown in section 5; finally, we conclude the paper in section 6 with some future research directions.

2 RELATED WORKS

Task scheduling problem in cloud computing has gained attention for several years. Recently, there are many studies related to this issue when fog computing has been proposed.

To minimize power consumption and latency when allocating job in the cloud-fog environment [3]. Deng et al. solved three independent sub-problems using the existing optimization techniques to achieve the goal. The first sub-problem is to find the optimal correlation between power consumption and latency in fog computing. This sub-problem is solved by using convex optimization [7]. The second sub-problem is to find the optimal correlation between power consumption and latency in cloud computing. To solve this sub-problem, they use the nonlinear integer method [9]. Finally, they minimized the transmission latency from the fog server to the cloud server in the third sub-problem by the Hungarian [8]. This study shows that fog computing improves performance for cloud computing by passing the modest computing resources, so bandwidth and transmission latency can reduce. However, this study less well suited to the fog computing infrastructure because the cloud hub is responsible for the allocation of work. So it can reduce overall performance.

Another approach proposed by Ningning el al. proposed a load balancing mechanism in fog computing based on the graph method [10]. Tasks assigned to one or more fog nodes based on the resources required for each task. Fog nodes present by an un-directed graph. The effect of this mechanism shows on the running time of tasks. However, the main disadvantage of this method is not optimal for dynamic load balancing of fog computing because the graph often redistribute to deal with the changes of fog computing.

Guo et al. focused on trying to minimize latency in edge clouds[5]. They proposed the job allocation problem in cloud-fog computing. However, they used quite simple model to determine power consumption and latency. In particular, end-user devices send tasks to base stations. Base stations receive and send tasks to the edge cloud servers for execution. Then the results are returned to end-users.

Another approach is the work in [4], this paper study on task distribution, base station association, and virtual machine placement in medical cyber-physical systems support by fog computing. This study minimizes the overall cost to guarantee QoS. However, the proposed model does not generalize into scenarios in the cloud-fog computing environment because it bases on cellular network architecture. The model lacked the cloud entity, fog nodes assumed to co-locate with the base station and do not have computation offloading capability.

The work in [11] also focused on improving the quality of user experience (QoE) by addressing load balancing in fog computing. Firstly, local computing resources in small cells allocated. Next, clusters are set up for fog computing, for each user's request, according to a specific optimization goal for arrival time, delay and so on. This study yields good results with low computational infrastructure but can be complicated when the fog computing infrastructure expanded to large scale.

In [1], Bitam et al. proposed a task scheduling strategy using Bee Life Algorithm (BLA). The article focused on two main objectives which are execution time and memory. The experimental results are quite good. However, this article did not mention the association

Table 1: Summary of the related works discussed

Article	Ideas	Improvement criteria	Limitation
Bitam et al. [2017]	bee life algorithm	execution time memory	small data set, only in fog environment
Gu et al. [2017]	two-phase linear programming-based heuristic	communication cost, apply for cellular network	no computational offloading capability
Deng et al. [2016]	non-linear integers, Hungarian method	power consumption, delay	suitable for centralized infrastructure, not easy to apply to fog computing
Guo et al. [2016]	Markov decision problems	power consumption, delay	a continuous-time queueing system
Ningning et al. [2016]	graph partitioning, a minimum spanning tree	execution time	complexity when the number of user requests increases
Oueis et al. [2015]	heuristic	user satisfaction, task latency, power consumption	complexity for large-scale fog computing infrastructure

with cloud data centre and experimented results in a small data set. Table 1 summarizes the main ideas, the method, the improvement criteria and limitations of past related works.

In this study, we propose a task scheduling problem model in the cloud-fog environment. Then we implement TCaS algorithm - an strategy based on evolutionary algorithms to obtain optimal trade-off between execution time and processing cost.

3 TASK SCHEDULING PROBLEM

3.1 System architecture

In the fog environment, the processing takes place in a data hub on a smart device, or in a smart router, gateway, switcher or local server, etc. Because of limited in processing capacity, some fog nodes cooperate regionally together, simultaneous connect to cloud nodes known as cloud virtual machines to satisfy mobile user's need. Assuming that our system consists of f fog nodes and c cloud nodes and a fog broker additionally. Fog nodes communicate directly to mobile users. All requests from mobile users are forwarded immediately to fog broker, who is responsible to analysis, estimate and then schedule all tasks to be executed in cloud-fog system. Fog broker is laid close to fog nodes so the time consuming for data communication between each other is negligible.

To guarantee the performance of system, the TCaS algorithm, which is installed into fog broker, aims at finding an optimal task executing schedule achieving time and cost efficiency. The operation of our system model is described step by step in Figure 1.

Firstly, mobile user sends a request (step 1) which is handled by fog node it connects. This request (or job) is immediately forwarded to fog broker (step 2). In order to be processed in distributed system, each job is decomposed into a set of tasks (step 3) which are estimated the number of instructions and the resource usage needed (step 4). Handling all information of tasks and nodes, fog broker runs a scheduling algorithm (step 5) to find a good task assignment. Following the output, tasks are sent to the corresponding cloud nodes and fog nodes (step 6). Each node is responsible for processing all tasks assigned (step 7) then sends results back to

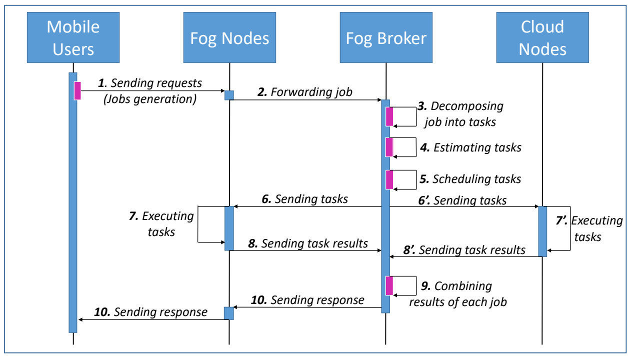

Figure 1: The operation of cloud-fog system.

fog broker(step 8). Waiting for all tasks completed, result of job is combined (step 9) by fog broker, subsequently,the response is sent to mobile user through the fog node that user connects with (step 10).

3.2 Problem formulation

When requests from BoT applications are sent to the fog layer, they will be decomposed into small and independent tasks in order to be processed over the cloud-fog computing infrastructure. Each task has properties: number of instructions, memory required, size of input and output files. Assuming that, at each time, there is a set of n independent tasks sent to the system, as follows:

$$T = \{T_1, T_2, T_3, ..., T_n\}$$

The cloud-fog computing infrastructure consists of processors: cloud nodes and fog nodes, which have same attributes such as CPU clock rate, CPU usage fee, memory usage fee, and bandwidth usage fee. However, cloud nodes typically are more powerful than the fog nodes, but the cost of using them is greater. The set of m processors including c cloud nodes and f fog nodes in the system

($N = N_{cloud} \cup N_{fog}$) is expressed as:

$$N = \{N_1, N_2, N_3, ..., N_m\}$$

Each task $T_k (T_k \in Tasks)$ is assigned to the processor $N_i (N_i \in Nodes)$, which is represented as T_k^i. One processor can be assigned to process a set of one or more tasks:

$$N_i Tasks = \left\{T_x^i, T_y^i, ..., T_z^i\right\}$$

The task scheduling problem in cloud-fog computing environment could be formulated as searching of a set:

$$NodeTasks = \left\{T_1^a, T_2^b, T_3^c, ..., T_n^p\right\}$$

For a set of tasks $N_i Tasks$, the execution time (EXT) that node N_i needs to complete all tasks assigned is:

$$EXT(N_i) = \sum_{T_k^i \in N_i Tasks} ExeTime(T_k^i) = \frac{\sum_{T_k^i \in N_i Tasks} length(T_k)}{CPUrate(N_i)} \tag{1}$$

where $ExeTime(T_k^i)$ is the execution time of T_k processed in node N_i, which is determined by the number of instructions $length(T_k)$ of task T_k and the CPU clock rate $CPUrate(N_i)$ of node N_i:

$$ExeTime(T_k^i) = \frac{length(T_k^i)}{CPUrate(N_i)} \tag{2}$$

Makespan is the total time for the system to complete all tasks, defined from the time when the request is received to the time that the last task is completed, or the time when the last machine finishes. *Makespan* is determined by the formula :

$$Makespan = \underset{1 \le i \le m}{Max} [EXT(N_i)] \tag{3}$$

Let *MinMakespan* be the lower bound of *Makespan*, literally, be the shortest time that the system needs to complete all tasks. Ideally, when all nodes finish all tasks assigned at the same time, *MinMakespan* will be obtained and calculated by:

$$MinMakespan = EXT(N_1) = ... = EXT(N_m)$$

so:

$$MinMakespan = \frac{\sum_{1 \le k \le n} length(T_k)}{\sum_{1 \le i \le n} CPUrate(N_i)} \tag{4}$$

When one task is executed in cloud-fog system, a monetary amount must be paid including: processing cost, memory usage cost, and bandwidth usage cost. The cost estimated when node N_i processes the task T_k is expressed as:

$$Cost(T_i^k) = c_p(T_i^k) + c_m(T_i^k) + c_b(T_i^k) \tag{5}$$

In equation (5), each cost is calculated as follows.

Processing cost is defined as:

$$c_p(T_k^i) = c_1 * ExeTime(T_k^i) \tag{6}$$

where c_1 is the CPU usage cost per time unit in node N_i, and $ExeTime(T_k^i)$ is defined as (2).
Given c_2 be the memory usage cost per data unit in node N_i and $Mem(T_k)$ be the memory required by task T_k, the memory usage cost would be:

$$c_m(T_k^i) = c_2 * Mem(T_k^i) \tag{7}$$

Task T_k processed in node N_i needs an amount of bandwidth $Bw(T_k)$ which is the sum of input and output file size. Let c_3 be the bandwidth usage cost per data unit, the bandwidth usage cost is defined as follow:

$$c_b(T_k^i) = c_3 * Bw(T_k^i) \tag{8}$$

Total cost for all tasks to be executed in cloud-fog system is calculated as follows:

$$TotalCost = \sum_{T_k^i \in NodeTasks} Cost(T_k^i) \tag{9}$$

Assuming that *MinTotalCost* is the lowest cost needed for a set of tasks T to be completed in cloud-fog system. Obviously, it can be obtained when each task is assigned to the cheapest node. Provided the information of each node, it is easy to determine which node processes task T_k with the lowest cost, known as $MinCost(T_k)$. Therefore, *MinTotalCost* is specific with one set of tasks T, and can be defined as:

$$MinTotalCost = \sum_{T_k \in T} MinCost(T_k) = \sum_{T_k \in T} \underset{1 \le i \le m}{Min} (Cost(T_k^i)) \tag{10}$$

We can define an utility function which computes the trade-off between the makespan and total cost as follows:

$$F = \alpha * \frac{MinMakespan}{Makespan} + (1 - \alpha) * \frac{MinTotalCost}{TotalCost} \tag{11}$$

where $\alpha (\alpha \in [0, 1])$ is the balance coefficient between makespan and total cost. $\alpha = 0.5$ means that time and cost have same priority in optimizing; if $\alpha > 0.5$, our mechanism focuses on minimizing makespan with higher priority than total cost, it is in case that user is willing to pay more money to obtain better performance; inversely, $\alpha < 0.5$, cost is more prioritized than time while user has a tight budget.

The objective is optimizing execution time and processing cost, which means finding a solution that *Makespan* and *TotalCost* are minimum and close to *MinMakespan* and *MinTotalCost*, correspondingly. Consequently, the closer to 1 the utility function F is, the more optimal solution we obtain.

The task scheduling problem in cloud-fog computing environment can be formulated:

Input:

$T = \{T_1, T_2, T_3, ..., T_n\}$: a set of independent tasks

$N = \{N_1, N_2, N_3, ..., N_m\}$: a set of processing nodes including cloud nodes and fog nodes

Output:

$NodeTasks = \left\{T_1^a, T_2^b, T_3^c, ..., T_n^p\right\}$: an assignment of all tasks executed into processing nodes

Objective:

Maximize the utility function F:

$$F = \alpha * \frac{MinMakespan}{Makespan} + (1 - \alpha) * \frac{MinTotalCost}{TotalCost} \longrightarrow Max$$

4 PROPOSED EVOLUTIONARY ALGORITHM FOR TASK SCHEDULING PROBLEM

In order to deal with task scheduling problem in cloud-fog computing environment, the TCaS algorithm is proposed based on an evolutionary algorithm - Genetic Algorithm (GA) as follows.

4.1 Encoding of chromosomes

An individual specified by a chromosome in genetic algorithm represents a solution of task scheduling. For encoding of chromosomes, a n-dimensional array corresponding to n genes is used. The sequence number of gene is the sequence number of task, each gene has a value of an integer k in range $[1; m]$ with m is the number of nodes, which indicates that the corresponding task is assigned to the node numbered k.

For example, a set of 10 tasks is executed in a cloud-fog system of 3 nodes, one solution of task scheduling can be:

$$NodeTasks = \{T_1^3, T_2^1, T_3^3, T_4^2, T_5^1, T_6^2, T_7^2, T_8^3, T_9^1, T_{10}^3\}$$

The chromosome expresses the above solution would be:

$$chromosome = [3, 1, 3, 2, 1, 2, 2, 3, 1, 3]$$

Node 1 executes the set of tasks $\{2, 5, 9\}$, the set of tasks $\{4, 6, 7\}$ is assigned to be processed in node 2, while node 3 is responsible for the set of tasks $\{1, 3, 8, 10\}$

This chromosome encoding method is chosen because of flexibly performing genetic operations, such as crossover and mutation, to create new individuals exploring the solution search space, simultaneously, inheriting quality gene segments from parents. The order of tasks processed is not considered because performing tasks in a different order on one machine does not affect the objectives, namely makespan and total cost.

4.2 Initialize population

The initial population is the set of all individuals that are used in the GA to find out the optimal solution. Assuming that the size of population is $\mathbb{N}$. In order to discover many areas in the search space, also to ensure the diversity of the population in the first generation, this $\mathbb{N}$ individuals are initialized randomly. From the initial population, the individuals are selected, and some operations are applied on them to form the next generation.

4.3 Fitness function

The fitness function is the measure of the superiority of an individual in the population. The individual with high fitness value represents for a good quality solution. The fitness value of each individual is estimated by the utility function F shown in Equation 11. Depending on the fitness value, the individual can survive or die out through each generation.

4.4 Genetic operators

4.4.1 Crossover operator. With chromosome encoding as an array of integers, two-point crossover operation is chosen to generate offspring inheriting good genes from parents. This operation is shown in Figure 2, where two crossover points are randomly selected, the first parent exchanges the middle gene segment to the

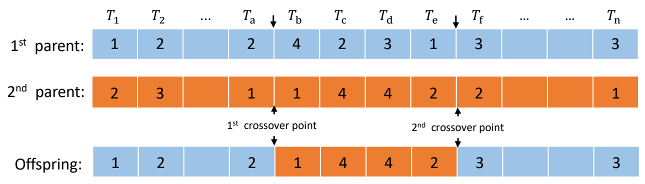
T_1 T_2 ... T_a T_b T_c T_d T_e T_f ... ... T_n
1st parent: 1 2 2 4 2 3 1 3 3
2nd parent: 2 3 1 1 4 4 2 2 1
1st crossover point
2nd crossover point
Offspring: 1 2 2 1 4 4 2 3 3

Figure 2: Two-point crossover operator.

second parent, the remaining genes remain unchanged, forming a new individual.

In the crossover process on over the population, parental selection influences seriously the efficiency of the algorithm. Each individual has a crossover rate of α. Each individual in the population is considered to be the first parent with the probability α, then the Roulette wheel technique is applied to select the second parent in the population to participate in crossover process. With this selection technique, high quality individuals with a larger fitness value have higher probability of being selected as parent, thus ensuring good gene segments are more likely preserved in the next generation.

4.4.2 Selection strategy. The selection mechanism is used to select individuals to form a population for the next generation based on the Darwin's law of survival. Immediately after the crossover process, natural selection is conducted. The offspring is calculated for the fitness value and is compared to the parent, if the offspring is better than the parent, the individual is preserved to form the next-generation population. Otherwise, this offspring is discarded and the parent is retained in the next generation.

This selection maintains the diversity of the population over generations when the genes of the lesser individuals are not eliminated so rapidly and still contribute to the exploration of new regions on the search area, simultaneously, good gene segments are not repeated too many times in individuals in one population.

4.4.3 Mutation operator. After crossover and natural selection process, new population is formed with $\mathbb{N}$ individuals. Each individual participates in one-point mutation with the rate of γ. The location of the mutant gene was randomly selected and replaced by a different value (see Figure 3), by this way, the task chosen is assigned to be processed in another node.

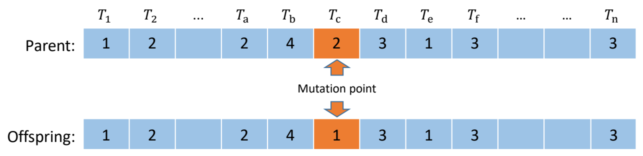
T_1 T_2 ... T_a T_b T_c T_d T_e T_f ... ... T_n
Parent: 1 2 2 4 2 3 1 3 3
Mutation point
Offspring: 1 2 2 4 1 3 1 3 3

Figure 3: One-point mutation operator.

Mutation perfects the limitations of crossover operator by finding the optimal solution when individuals are vicious around the extremes, or escaping from the local extremes to explore the other areas in the solution space.

5 PERFORMANCE EVALUATION

5.1 Experimental settings

In our experimental study, we conducted a set of simulation tests in a cloud-fog infrastructure to evaluate the effectiveness of TCas algorithm in scheduling a bag of tasks. Cloud and fog nodes have different processing power as well as resource usage cost. We assume that each node has its own processing capacity represented by processing rate (measured by MIPS - million instructions per second), along with CPU, memory and bandwidth usage cost. There are 15 processing nodes constructing the cloud-fog system with the characteristics shown in Table 2. In the fog layer, fog nodes have limited processing capacity such as routers, gateways, workstations, or personal computers. While in cloud layer, servers or virtual machines in high performance data centers are responsible for handling tasks. Therefore, the processing speed of cloud nodes is much faster than fog nodes. In contrast, the cost of using resources in the cloud is more expensive than in the fog. These costs are calculated according to Grid Dollars (G$) - a currency unit used in simulation to substitute for real money.

Table 2: Characteristics of the cloud-fog infrastructure

Parameter (Unit)	Fog environment	Cloud environment
Number of nodes (nodes)	10	5
CPU clock rate (MIPS)	[500, 2000]	[3000, 10000]
CPU usage cost (G$/second)	[0.2, 0.5]	[1.0, 2.1]
Memory usage cost (G$/MB)	[0.01, 0.03]	[0.02, 0.05]
Bandwidth usage cost (G$/MB)	[0.01, 0.02]	[0.05, 0.01]

The cloud-fog system is responsible to execute all requests from user. Each request is decomposed into a set of tasks, which are analyzed and estimated resources that they needed. Assuming that each tack has some attributes as: number of instructions, memory required, input file size, and output file size. Depending on the workload of each request, the set of task may vary widely in size. For that reason, 10 data sets with a small size as 20 tasks to the big one as 200 tasks were created to participate in our simulation. Each task in data sets was generated randomly with its attributes following Table 3. With randomness, the experiment may cover various scenarios because many types of tasks are created, some of them require a huge amount of processing while others need more memory or bandwidth usage, and so on. Note that we proposed this benchmarks (eg. Table 2, Table 3) because of no benchmarks published in the literature in this research area.

The settings of the experimental environment are shown in Table 4, the simulation is developed in Java with Eclipse editor, and using iFogSim [6]. iFogSim is chosen because it is developed based on CloudSim, a cloud computing platform that has been widely used and validated in numerous studies over the years, iFogSim supports modeling and simulation of architecture as well as services in cloud-fog environment.

Table 3: Attributes of tasks

Property	Value	Unit
Number of instructions	[1, 100]	10^9 instructions
Memory required	[50, 200]	MB
Input file size	[10, 100]	MB
Output file size	[10, 100]	MB

Table 4: Simulation setup

System	Intel Core^{TM} i7-4710MQ, CPU 2.50GHz
Memory	8 GB
Simulator	iFogSim
Operating system	Windows 10 Professional

Based on the above metrics, we compare the results obtained by TCaS algorithm with those obtained by Bee Life Algorithm (BLA), which was introduced by Bitam in [1].

BLA is an optimization method that is inspired from two most important behaviors of bees namely marriage (or reproduction) and food foraging. First, $\mathbb{N}$ individuals which form the initial population are evaluated by a fitness function. After sorting the population, the best individual was chosen as the queen, the next D bees are identified as drones, the rest bees are W workers bee. Population is changed through each generation, the bee's life cycle consisting of two main behaviors: reproduction and food foraging. During the reproduction stage, the queen mates with a set of drones to produce broods by the crossover and mutation operators. The individual evaluation is conducted to find the best bee who substitute the precedent queen, the following D fittest bees to be drones and W new workers. Now, the food foraging stage is executed, each worker is responsible for finding the food source and then recruiting other workers to collect the found food. Afterward, the best worker of every region keeps alive in next population.

Table 5: BLA, TCaS parameters

Parameter		**BLA**	**TCaS**
Running times		30	30
Population size ($\mathbb{N}$)	queen	1	400
	drones (D)	150	
	workers (W)	249	
Crossover rate (α)		90%	90%
Mutation rate (γ)		1%	1%
Number of generations		1000	1000

With this idea, BLA was experienced along with our TCaS algorithm in the proposed task scheduling problem. The parameters of two algorithms are shown in Table 5.

The balance coefficient α indicates the priority of optimization between makespan and total cost. In some first simulations, we evaluate two algorithms in the scenario where time and cost have the same level of interest, which means $\alpha = 0.5$.

5.2 Experimental results

Firstly, the convergence comparison between the proposed TCaS algorithm and BLA is conducted in data set of 100 tasks over 1000 generations with coefficient balance $\alpha = 0.5$. We observe the population every 50 generations, fitness values of all individuals are synthesized and shown in Figure 4. The middle point in line indicates the average fitness value of whole population while the bottom and up of the error bar present the fitness value of the worst and best individual, respectively. To be observed, BLA expresses a fast convergence after 250 generations, however, the found solution has a small fitness value at 0.59 compared with 0.84 that TCaS algorithm achieves. In some first generations, individuals of TCaS algorithm are distributed with a wide range of fitness value. The best solution is found in 500^{th} generation, afterward, the fitness range is narrowed down, which means the population gathers around the extreme. The mechanism of BLA that all drones perform crossover operator with only one queen each generation makes the population easily fall into local extreme, consequently, the algorithm converges fast with low fitness value. Meanwhile, the proposed TCaS algorithm maintains population diversity, so can reach more optimal solution.

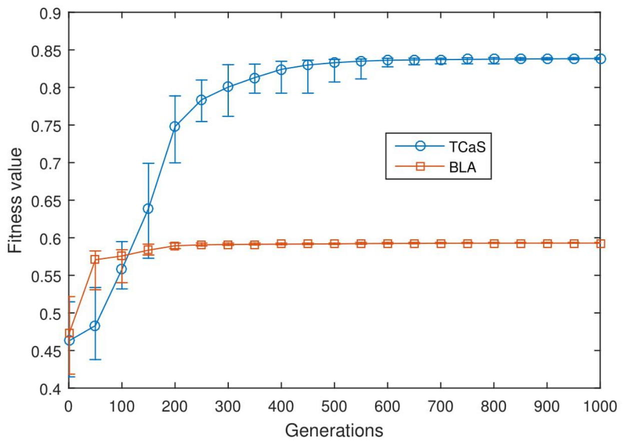

Figure 4: Convergence comparison between TCaS and BLA.

Figure 5, 6 and 7 respectively compare the time-cost trade-off, makespan and the total cost between our proposed TCaS algorithm and BLA as the number of tasks changes. The average results are obtained by running two algorithms in each data set over 30 times with 1000 generations (or iterations) and the coefficient balance $\alpha = 0.5$.

In Figure 5, the TCaS algorithm obviously overtakes BLA in term of fitness value which represents time-cost trade-off. In the small set of tasks (eg: 20, 40 tasks), our TCaS algorithm is slightly better than BLA at fitness value over 0.6. Running on bigger sets, TCaS algorithm records a leap from the set size of 60 to 200 when the fitness values are above 0.8 and is better averagely 38.6% than BLA

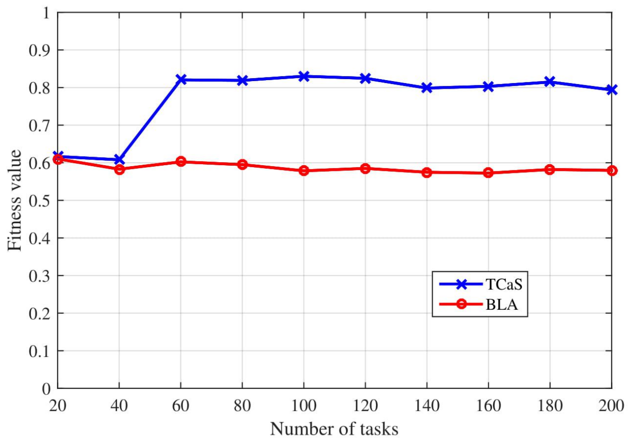

Figure 5: Time-cost trade-off factor of TCaS versus BLA.

whose fitness value remains nearly 0.6 over all data sets. This is because of the reason mentioned above, TCaS algorithm based on GA performs powerful in discovering whole search space to find the globally optimal solution while it is easy for BLA to be stuck in local extremism. The small sets of 20 and 40 tasks may contain very long tasks which last longer the makespan, consequently, the fitness value keeps small.

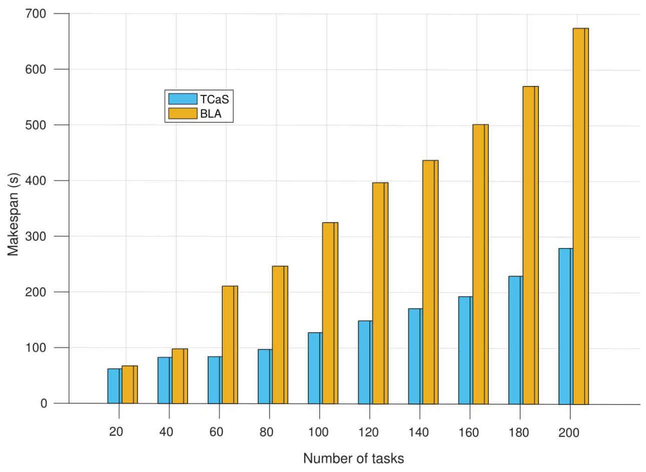

Figure 6: Makespan comparison between TCaS and BLA.

Figure 6 presents the dramatically difference in scheduling length of TCaS algorithm versus BLA, especially with the larger set of tasks. BLA consumes a lot of time to complete a set of task, which is averagely 2.26 times larger than TCaS needs. Considering the processing cost aspect, BLA performs slightly better than TCaS as shown in Figure 7 when saves about 7.35% comparing with our algorithm. However, this number is very small to compare with which TCaS shown in execution time. Actually, response time is very important in guaranteeing QoS when affecting directly to user's experience, so users are willing to spend a small amount of money experiencing much higher performance service.

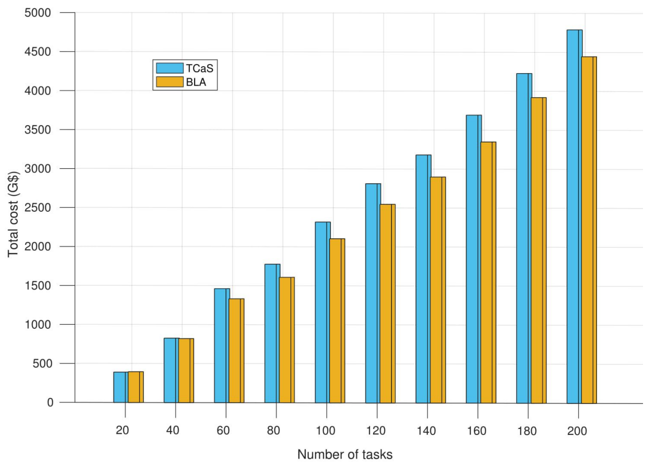

Figure 7: Processing cost comparison between TCaS and BLA.

The above results demonstrate that the TCaS algorithm can reach better trade-off between makespan and total cost than BLA, together with showing off the superiority of time optimization. BLA is greedy looking for a solution with a lower cost, soon gets lost in local extremism, consequently, achieves bad results.

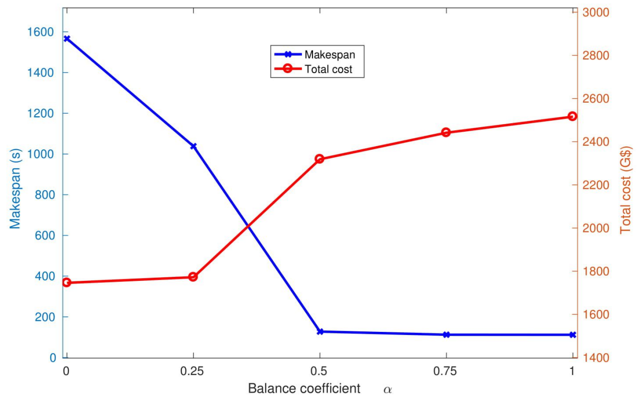

Figure 8: Time and cost as the balance coefficient α changes.

Figure 8 indicates the affect of balance coefficient α in time and cost achieved by TCaS algorithm when a set of 100 tasks is processed. In case $\alpha = 0$, monetary cost is maximized, however, makespan is too large at 1566 seconds because many tasks are assigned centrally to cheap nodes. When α to 0.5, makespan is improved 12.3 times while 1.33 times of total cost is needed comparing with case $\alpha = 0$. Increasing α to 0.75, 1, the scheduling length decreases slightly together with a small amount of cost added. It is because execution time is more concentrated to be optimized than cost. This experiment performs that by adjusting the balance coefficient α, our model can flexibly satisfy user's requirement when they interest in high performance execution or cost efficiency.

6 CONCLUSION AND FUTURE WORK

In this work, we focus on task scheduling problem for BoT applications in cloud-fog computing environment. The TCaS algorithm based on an evolutionary algorithm was proposed and evaluated performance in various scenario. Comparing with BLA, TCaS outperforms averagely 38.6% over 10 sets of tasks in the term of trade-off between time and cost execution, especially obtains much shorter scheduling length. Moreover, the proposed algorithm can flexibly satisfy user's requirement of high performance processing or cost efficiency.

In the future, we will research, improve, apply more algorithms to solve scheduling problem, especially evolutionary algorithms. In addition, we plan to expand scheduling problem by focusing on optimizing many other goals, such as time, transmission costs, computing resources, energy consumption to satisfy users. Constraints of budget, deadline, resource limitation can be added for greater practicality.

ACKNOWLEDGMENTS

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number DFG 102.01-2016.03

REFERENCES

[1] Salim Bitam, Sherali Zeadally, and Abdelhamid Mellouk. 2017. Fog computing job scheduling optimization based on bees swarm. *Enterprise Information Systems* 12, 4 (2017), 373–397.
[2] Fog Computing. 2015. the Internet of Things: Extend the Cloud to Where the Things are. *Cisco White Paper* (2015).
[3] Ruilong Deng, Rongxing Lu, Chengzhe Lai, Tom H Luan, and Hao Liang. 2016. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE Internet of Things Journal* 3, 6 (2016), 1171–1181.
[4] Lin Gu, Deze Zeng, Song Guo, Ahmed Barnawi, and Yong Xiang. 2017. Cost efficient resource management in fog computing supported medical cyber-physical system. *IEEE Transactions on Emerging Topics in Computing* 5, 1 (2017), 108–119.
[5] Xueying Guo, Rahul Singh, Tianchu Zhao, and Zhisheng Niu. 2016. An index based task assignment policy for achieving optimal power-delay tradeoff in edge cloud systems. In *Communications (ICC), 2016 IEEE International Conference on.* IEEE, 1–7.
[6] Dastjerdi A. V. Ghosh S. K. Gupta, H. and R. Buyya. 2017. iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things. *Edge and Fog Computing Environments* (2017).
[7] Jianping He, Peng Cheng, Ling Shi, Jiming Chen, and Youxian Sun. 2014. Time synchronization in WSNs: A maximum-value-based consensus approach. *IEEE Trans. Automat. Control* 59, 3 (2014), 660–675.
[8] Harold W Kuhn. 2005. The Hungarian method for the assignment problem. *Naval Research Logistics (NRL)* 52, 1 (2005), 7–21.
[9] Duan Li and Xiaoling Sun. 2006. *Nonlinear integer programming*. Vol. 84. Springer Science & Business Media.
[10] Song Ningning, Gong Chao, An Xingshuo, and Zhan Qiang. 2016. Fog computing dynamic load balancing mechanism based on graph repartitioning. *China Communications* 13, 3 (2016), 156–164.
[11] Jessica Oueis, Emilio Calvanese Strinati, and Sergio Barbarossa. 2015. The fog balancing: Load distribution for small cell cloud computing. In *Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st.* IEEE, 1–6.
[12] L. Sambath P. Sornapriya, M. Roshini. 2018. Internet-of-Things. *International Journal of Research in Engineering, Science and Management* (2018).
[13] H. Casanova S. Smallen and F. Berman. November 2001. Applying Scheduling and Tuning to On-line Parallel Tomography. *Proceedings of Supercomputing 01, Denver, Colorado, USA* (November 2001).
[14] Salpeter E.E. Salpeter M.M. Stiles J.R., Bartol T.M. 1998. Monte Carlo Simulation of Neuro- Transmitter Release Using MCell, a General Simulator of Cellular Physiological Processes. *In: Bower J.M. (eds) Computational Neuroscience. Springer, Boston, MA* (1998).