Hindawi
Security and Communication Networks
Volume 2018, Article ID 2863793, 12 pages
https://doi.org/10.1155/2018/2863793

Research Article

Scheduling Parallel Intrusion Detecting Applications on Hybrid Clouds

Yi Zhang[1], **Jin Sun**[1], **Zebin Wu**[1,2], **Shuangyu Xie**[1] and **Ruitao Xu**[1]

[1]*School of Computer Science and Engineering, Nanjing University of Science and Technology, No. 200 Xiaolingwei Street, Nanjing, China*
[2]*Lianyungang E-Port Information Development Co. Ltd., Lianyungang, China*

Correspondence should be addressed to Yi Zhang; yzhang@njust.edu.cn

Received 29 April 2018; Accepted 5 July 2018; Published 16 October 2018

Academic Editor: Xuyun Zhang

Recently, Parallel Intrusion Detection (PID) becomes very popular and its procedure of the parallel processing is called a PID application (PIDA). This PIDA can be regarded as a Bag-of-Tasks (BoT) application, consisting of multiple tasks that can be processed in parallel. Given multiple PIDAs (i.e., BoT applications) to be handled, when the private cloud has insufficiently available resources to afford all tasks, some tasks have to be outsourced to public clouds with resource-used costs. The key challenge here is how to schedule tasks on hybrid clouds to minimize makespan given a limited budget. This problem can be formulated as an Integer Programming model, which is generally NP-Hard. Accordingly, in this paper, we construct an Iterated Local Search (ILS) algorithm, which employs an effective heuristic to obtain the initial task sequence and utilizes an insertion-neighbourhood-based local search method to explore better task sequences with lower makespans. A swap-based perturbation operator is adopted to avoid local optimum. With the objective of improving the proposal's efficiency without loss of any effectiveness, to calculate task sequences' objectives, we construct a Fast Task Assignment (FTA) method by integrating an existing Task Assignment (TA) method with an acceleration mechanism designed through theoretical analysis. Accordingly, the proposed ILS is named FILS. Experimental results show that FILS outperforms the existing best algorithm for the considered problem, considerably and significantly. More importantly, compared with TA, FTA achieves a 2.42x speedup, which verifies that the acceleration mechanism employed by FTA is able to remarkably improve the efficiency. Finally, impacts of key factors are also evaluated and analyzed, exhaustively.

1. Introduction

Cloud computing is a novel service-based paradigm that delivers large-scale computational resources in the form of a pay-as-you-go model. Recently, some innovative providers (e.g., VMware partnered with IBM) deliver hybrid cloud construction solutions (e.g., VMware Cloud Foundation (http://www.vmware.com/products/cloud-foundation.html)), which enable creating an extension of a private cloud on public clouds (as seen in Figure 1). As a result, administrators/programs (e.g., application/task schedulers) of the private cloud are able to use resources of public clouds seamlessly and transparently through unified tools/interfaces, since both the private cloud and its extension use the same virtualization technique provided by hybrid cloud construction solutions. In other words, these administrators/programs can perform actions on public clouds just like on their own private cloud. For example, an administrator wants to create an instance of a small VM type for executing a task. When the private cloud has insufficient resources, the administrator can create an instance of the same small VM type on a public cloud to handle the task.

Intrusion Detection (ID) has been widely used to protect computer/network systems from diverse attacks. Recently, taking advantage of distributed computing technologies (e.g., cloud computing), Parallel Intrusion Detection (PID) becomes very popular because of its high efficiency [1, 2]. PID is an ID whose critical part can be processed in parallel. For instance, in an ID using data mining methods, the data can be divided into multiple partitions. As a result, the entire mining job on all the data is divided into multiple subjobs (or called tasks), which only perform mining work on partitions

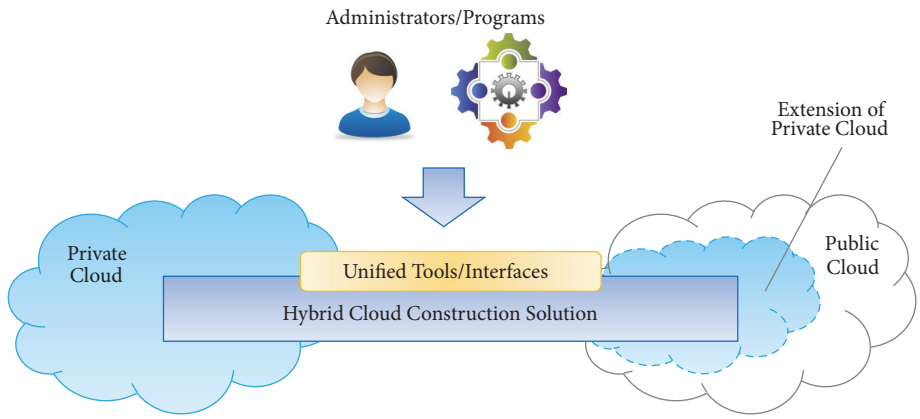

Figure 1: Administrators/programs of private cloud use resources of public clouds through unified tools/interfaces provided by hybrid cloud construction solutions [6].

and can accordingly be executed in parallel. Besides, an ID applying deep learning methods whose time-consuming training procedures can be performed in parallel is also a typical PID [3]. The procedure of the parallel processing in PID is called a PID application (PIDA) in this paper. Actually, these PIDAs can be considered as Bag-of-Tasks (BoT) applications, consisting of many independent tasks processed in parallel without synchronization or communication [4]. Theoretically, a cloud computing environment is the ideal platform to execute BoT applications, since it delivers cloud resources in a pay-as-you-go manner [5]. This is the reason why customers are willing to execute BoT applications on clouds.

Actually, customers may have private clouds, whose resources are free to use. Given multiple PIDAs to be processed for protecting different types of computer/network systems, these customers have to outsource some tasks to public clouds with additional costs, when their private clouds cannot afford all applications' tasks. Technically, tasks outsourced to public clouds can be achieved easily by the aforementioned hybrid cloud construction solutions. The key issue here is, given a limited budget, how to schedule tasks on hybrid clouds to minimize the total execution time (a.k.a. makespan).

This paper aims to schedule PIDAs on hybrid clouds, which is actually BoT Scheduling Problem (BTSP) with resource demands and budget constraints on hybrid clouds to minimize the makespan. In our previous work [6], this problem was formulated as an Integer Programming (IP) model, which is generally NP-Hard [7]. Accordingly, we also proposed an Effective Heuristic (EH) to solve the problem. EH starts from a task sequence generated by Longest Task First method (LTF) and uses a Task Assignment (TA) method to schedule all tasks in the obtained sequence to calculate the makespan. Although EH was verified to outperform the well-known RoundRobin method, we observe that the quality of the task schedule output by TA depends on its input task sequence, significantly. In order to further improve the schedule's quality, in this paper, we construct an Iterated Local Search (ILS) algorithm, which employs LTF to obtain the initial task sequence and utilizes an insertion-neighbourhood-based local search method to explore better task sequences with lower makespans. A swap-based perturbation operator is adopted to avoid local optimum. With the objective of improving the proposal's efficiency without loss of any effectiveness, instead of using TA to calculate task sequences' objectives, we construct a Fast TA (FTA) method by integrating TA with an acceleration mechanism designed through theoretical analysis. Accordingly, the proposed ILS is named as FILS. Experimental results show that FILS outperforms the existing best algorithm EH, considerably and significantly. More importantly, compared with TA, FTA achieves a 2.42x speedup and identical effectiveness, which verifies that the acceleration mechanism employed by FTA is able to remarkably improve the efficiency without losing effectiveness. The contributions of this paper are summarized below.

(i) We regard PIDA scheduling as BTSP, which can be formulated as an IP model.

(ii) We establish an effective algorithm FILS to solve the problem.

(iii) We propose an efficient heuristic FTA, which includes an acceleration method designed by theoretical analysis, to improve the efficiency.

(iv) We perform exhausted experiments to verify the proposed algorithms' effectiveness and efficiencies.

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 presents the problem description. The proposed FILS is described in Section 4. A full performance evaluation is shown in Section 5. Conclusions are given in Section 6 finally.

2. Related Works

In the literature, many efforts have been made to study BTSP in cloud environments, such as [6, 8–30]. Reference [4] presented a thoroughly comprehensive review on the state-of-the-art. As this paper considers budget-constrained BTSP

on hybrid clouds, we detail the two contributions and eight works related to budget constraints and hybrid clouds.

The following two contributions tackled BTSP with budget constraints in the environment of multiple public clouds. Reference [17] presented an algorithm for solving BTSP with either budget or deadline constraints. In their proposed algorithm, all the VM instances are initialized in the same type and iteratively replaced to be other different VM types. As a result, the objective (the total cost if a deadline constraint is considered or the makespan if a budget constraint is given) can be reduced without violating the constraint. Reference [18] proposed an approach to scale cloud resources for solving BTSP with both deadline and budget constraints while minimizing the total cost. They formulated the considered problem as an Integer Programming problem and developed a policy to determine the number of each VM type that can meet both constraints. In comparison with the problem tackled in these two papers, our considered problem has a different environment.

In the literature, many efforts have been made to study BTSP in cloud environments. Reference [4] presented a thoroughly comprehensive review on the state-of-the-art. As this paper considers BTSP on hybrid clouds, we detail the eight works related to hybrid clouds. Van den Bossche et al. [22] considered and formulated deadline-constrained BTSP as an IP and used the IBM CPLEX to obtain solutions. Later, the same authors [23] proposed two cost-efficient heuristics considering both computational and data-transferred costs. Similar heuristics are presented in [24] and a comprehensive analysis is performed through simulation experiments to show the effectiveness. Reference [26] tackled a similar deadline-constrained problem, in which physical machines on the private cloud are taken into account. The authors proposed a greedy heuristic that dispatches tasks to available physical machines on the private cloud and assigns them to public clouds while there are no available ones. Reference [27] solved deadline-constrained BTSP with the variation of tasks' runtimes. Thus, the authors constructed a method to estimate tasks' runtimes so that the scheduling plan can be updated accordingly. Reference [28] considered deadline-constrained BTSP on cloud federations (a term of hybrid clouds) and formulated it as an IP. The CPLEX was used to solve the problem with the results showing that the cloud federations benefit customers, compared with single cloud provider. Different from the three aforementioned works assuming that each task can be executed in an instance of any VM type, [29, 30] considered computation-intensive BTSP with resource demands and deadline constraints on hybrid clouds from the perspective of cloud providers with an objective of maximizing profit. Both papers employed Particle Swarm Optimization algorithms to solve the considered problems. Obviously, compared with the problems handled in the aforementioned eight papers (i.e., *deadline*-constrained BTSP with *cost minimization*), our considered problem (i.e., *budget*-constrained BTSP with *makespan minimization*) shares neither constraints nor objectives.

Our previous work [6] formulated the considered problem as an IP and proposed EH to solve it. EH uses LTF to generate the initial task sequence and employs TA to schedule all tasks in the obtained sequence to calculate the makespan. In this paper, we establish FILS that is demonstrated to outperform EH by experiments. As FTA is used rather than TA to calculate task sequences' objectives, we achieve a 2.42x speedup without loss of any effectiveness.

3. Problem Description

The formulation of the considered problem was given in our previous work [6]. For the completeness, we also introduce the formulation in this paper, briefly. We use $CP_0, CP_1, \ldots, CP_m$ to denote the $m + 1$ cloud providers. CP_0 represents the private cloud and the others are m public clouds. The private cloud provides k VM types $VM_1, VM_2, \ldots, VM_k$. Each VM type $VM_q (q = 1, 2, \ldots, k)$ has two performance parameters CPU_q and Mem_q that denote the number of CPUs and the amount of memory, respectively. When a task is outsourced to a public cloud, an instance of the VM type demanded by the task's application should be created to tackle the task on the public cloud. As aforementioned, technically, this procedure can be achieved easily by the above mentioned hybrid cloud construction solutions. Therefore, we can equivalently regard that the m public clouds also provide the k VM types. Additionally, $p_{qh} (q = 1, 2, \ldots, k,\ h = 1, 2, \ldots, m)$ represents the price (per time unit) for using an instance of VM_q provided by a public cloud $CP_h (h = 1, 2, \ldots, m)$. The private cloud's resources are free to use.

There are n applications. Each application $a_i (i = 1, 2, \ldots, n)$ requires a user-specified VM type. We use a binary variable x_{iq} to denote this relationship. $x_{iq} = 1$ means a_i demands VM_q; $x_{iq} = 0$ otherwise. Meanwhile, each application consists of T_i tasks $t_{i1}, t_{i2}, \ldots, t_{iT_i}$. Like most of existing contributions (such as [22–24, 29]), in our considered problem, one task is executed in one VM instance exclusively at a time, and each task is executed consecutively (i.e., no preemption is allowed). Those problems where one VM instance can run multiple tasks simultaneously or tasks can be executed preemptively are beyond the scope of this paper. A task $t_{ij} (i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, T_i)$ has a runtime $r_{ij} (i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, T_i)$. In other words, r_{ij} is the execution duration when t_{ij} is executed in an instance of the VM type required by its application.

Like [22–24, 29], setup times for VM instances (such as VM image loading, software installing, and network configuration) are regarded to be zero. Actually, there are cloud providers that are able to deliver VM instances in minutes (e.g., Amazon EC2) and even in seconds (e.g., qingcloud (https://www.qingcloud.com/)). However, in our considered problem, tasks' runtimes are longer than one hour at least. Setup times for VM instances are negligible compared with tasks' runtimes and are thus assumed to be zero. Additionally, though some traditional cloud providers (e.g., Amazon EC2) charge VM instances in hours, there are some innovative ones delivering resources in minutes (e.g., Microsoft Azure) or even in seconds (e.g., qingcloud and TecentCloud (https://www.qcloud.com/)). Obviously, users prefer using resources provided by these providers since they do not need to pay for an entire hour while only fraction

Table 1: Notations for problem description.

Notations	Indications	Notations	Indications
CP_0	Private cloud	CP_h	h-th public cloud ($h = 1, 2, \ldots, m$)
m	Total number of public clouds	VM_q	q-th VM type
k	Total number of VM types	CPU_q	Amount of CPU of VM_q
Mem_q	Amount of Memory of VM_q	p_{qh}	Price of VM_q provided by CP_h
a_i	i-th application	n	Total number of applications
T_i	Task number of a_i	c_i	completion time of a_i
t_{ij}	j-th task of a_i	r_{ij}	Runtime of t_{ij}
st_{ij}	Start time of t_{ij}	c_{ij}	completion time of t_{ij}
CPU^*/Mem^*	Capacity of CPU/Memory in CP_0	s	Time slot
B	Budget	x_{iq}	A variable $x_{iq} = 1$: a_i requires VM_q $x_{iq} = 0$: otherwise
y_{ijh}	A decision variable $y_{ijh} = 1$: t_{ij} is dispatched to CP_h $y_{ijh} = 0$: otherwise	z_{ijs}	A decision variable $z_{ijs} = 1$: t_{ij} is executed at slot s on CP_0 $z_{ijs} = 0$: otherwise

of this hour is used. Accordingly, in this paper, we regard resources are charged in seconds; i.e., the time unit is set as a second. As a result, it is not necessary to consider how to make use of an entire hour when we formulate the problem. The time axis is divided into several slots with the granularity of a second.

The private cloud CP_0 has limited number of available resources. The capacities of CPU and memory are denoted as CPU^* and Mem^*, respectively. In other words, for any time slot $s(s = 0, 1, \ldots)$, the amount of consumed resources cannot exceed CPU^* and Mem^*. All the m public clouds are regarded to have infinite resources. Let c_{ij} and c_i be the completion time of a task t_{ij} and the application's completion time, respectively. We have

$$c_i = \max\left\{c_{ij} \ \left(j = 1, 2, \ldots, T_i\right)\right\} \tag{1}$$

Accordingly, we can define the maximum of time slots S satisfying $S \geq \max\{c_i(i = 1, 2, \ldots, n)\}$. Let st_{ij} be the start time of the task t_{ij}. We can calculate c_{ij} by

$$c_{ij} = st_{ij} + r_{ij} \tag{2}$$

Let $y_{ijh}(i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, T_i,\ h = 0, 1, \ldots, m)$ and $z_{ijs}(i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, T_i,\ s = 0, 1, \ldots, S)$ be two decision variables. $y_{ijh} = 1$ means t_{ij} is dispatched to CP_h and $y_{ijh} = 0$ otherwise. $z_{ijs} = 1$ indicates t_{ij} is in execution at time slot s on CP_0 and $z_{ijs} = 0$ otherwise. Obviously, if a task t_{ij} is dispatched to the private cloud (i.e., $y_{ij0} = 1$), its start time $st_{ij} = \text{argmin}\{s \mid z_{ijs} = 1\}$; otherwise, $st_{ij} = 0$ (like [29], we also focus on computation-intensive BoT applications which require tiny amounts of data and the short duration of transferring these tiny-amount data can be negligible compared with tasks' runtimes. As setup times for VM instances have been reasonably assumed to be zero, we can regard that tasks can be started at time slot 0 on public clouds). With the consideration of the variables defined above, the total cost $Cost$ can be calculated by

$$Cost = \sum_{i=1}^{n}\sum_{j=1}^{T_i}\sum_{h=1}^{m}\sum_{q=1}^{k} x_{iq} y_{ijh} r_{ij} p_{qh} \tag{3}$$

Let B be the budget and c_{max} be the objective makespan. All the notations for problem description are listed in Table 1. The problem can be formulated as an Integer Programming (IP) model given below.

Minimize the makespan c_{max}:

$$c_{max} = \max\left\{c_i \ \ (i = 1, 2, \ldots, n)\right\} \tag{4}$$

s.t.

$$Cost \leq B \tag{5}$$

$$\sum_{h=0}^{m} y_{ijh} = 1, \qquad i = 1, 2, \ldots, n,\ \ j = 1, 2, \ldots, T_i \tag{6}$$

$$\sum_{i=1}^{n}\sum_{q=1}^{k}\sum_{j=1}^{T_i} z_{ijs} x_{iq} CPU_q \leq CPU^*, \quad s = 0, 1, \ldots, S \tag{7}$$

$$\sum_{i=1}^{n}\sum_{q=1}^{k}\sum_{j=1}^{T_i} z_{ijs} x_{iq} Mem_q \leq Mem^*, \quad s = 0, 1, \ldots, S \tag{8}$$

Equation (4) is the objective. Equation (5) guarantees that the total cost is not beyond the budget. Equation (6) ensures that a task is assigned to a unique cloud. Equations (7) and (8) make sure that the consumption of CPU and memory of the private cloud at any time slot cannot exceed CPU^* and Mem^*, respectively.

```
1: Initialize the solution and regard it as the current solution;
2: Regard the initial solution as the best solution;
3: while (termination criterion is not met) do
4:     Perform a local search method on the current solution;
5:     Update the best solution if a new one is found;
6:     Perform a perturbation operator on the best solution and regard the obtained solution as the current solution;
7: end while
8: return The best solution;
```

Algorithm 1: Framework of general ILS.

4. Fast Iterated Local Search Algorithm (FILS)

The framework of general ILS is given in Algorithm 1. We can see that an ILS starts with an initial solution. If the termination criterion is not met, a local search method is performed on the current solution to explore new good solutions, and a perturbation operator is used to avoid local optimum. In this paper, we proposed a FILS, in which task sequences are considered solutions. LTF is used to generate the initial solution. An Insertion-Based Local Search Method (ILSM) is employed to explore better task sequences with lower makespans. A Swap-Based Perturbation Operator (SPO) is used to perturb the current solution. FTA is constructed to calculate makespans of task sequences. Details are given in this section.

4.1. Longest Task First (LTF). In our previous work [6], four heuristics including Highest Lowest Public Cost (LPC) First (HLPCF), Lowest LPC First (LLPCF), Longest Task First (LTF), and Shortest Task First (STF) were examined by experiments with the results showing that LTF is the best and helps the proposed EH to achieve good effectiveness. Accordingly, we also use LTF to generate the initial task sequence of FILS. LTF arranges all tasks by their runtimes in the nonascending order.

Meanwhile, it is worth introducing LPC, which will be used to describe FILS in Section 4.4. Like [6], the costs of executing tasks on public clouds are defined as public costs. A task's LPC can be defined as the minimum of all its public costs. Given a task t_{ij}, assume that the index of the VM type demanded by its application is u; i.e., $x_{iu} = 1$. The task's LPC can be calculated by

$$LPC_{ij} = \min\left\{r_{ij}p_{uh} \ \ (h = 1, 2, \ldots, m)\right\} \tag{9}$$

Accordingly, the corresponding public cloud is called the task's "Ideal" Public Cloud (IPC). Obviously, the index of a task's IPC should meet

$$IPC_{ij} = \operatorname{argmin}\left\{h \mid r_{ij}p_{uh}\right\} \tag{10}$$

4.2. Insertion-Based Local Search Method. Given a task sequence ts with the length T, ILSM first regards it as a temp task sequence $temp$. Then, ILSM removes the $u(u = 1, 2, \ldots, T)$-th task from $temp$ and reinserted this task to the left $temp$ at each position except the task's original one. As a result, $T - 1$ new task sequences are generated and evaluated by TA/FTA (corresponding to CILS/FILS) to calculate their makespans. If one generated task sequence gts has a lower makespan than ts, both ts and $temp$ are set as gts. Afterwards, ILSM processes the $(u + 1)$-th task in $temp$ in the same way. After the T-th task has been processed, ILSM terminates. Obviously, the complexity of ILSM is $O(T^2 \cdot O(TA/FTA))$, in which $O(TA/FTA)$ represents the complexity of TA/FTA.

We use an example to clarify the procedure of ILSM. In this example, we have one application with three tasks. Given a task sequence $ts = (t_{11}, t_{12}, t_{13})$ with makespan 10, ILSM first sets $temp \longleftarrow ts$. Then, ILSM removes the first task t_{11} and reinserted it to $temp$ at each position except the first one. As a result, two new task sequences $ts_1 = (t_{12}, t_{11}, t_{13})$ and $ts_2 = (t_{12}, t_{13}, t_{11})$ are generated. Assume their makespans are 12 and 9, respectively. In other words, ts_2 gets a lower makespan than ts and we set $temp \longleftarrow ts \longleftarrow ts_2$. Afterwards, the second task in $temp$ (i.e., t_{13}) is removed and reinserted. The two obtained task sequences are $ts_3 = (t_{13}, t_{12}, t_{11})$ and $ts_4 = (t_{12}, t_{11}, t_{13})$ with the makespans 8 and 12, respectively. Accordingly, we set $temp \longleftarrow ts \longleftarrow ts_3$. Finally, the third task in $temp$ (i.e., t_{11}) is removed and reinserted. The two generated task sequences are $ts_5 = (t_{11}, t_{13}, t_{12})$ and $ts_6 = (t_{13}, t_{11}, t_{12})$ with the makespans 6 and 14, respectively. In other words, ts_5 obtains a lower makespan than ts does. Consequently, we set $temp \longleftarrow ts \longleftarrow ts_5$ and ts is the final result of ILSM.

4.3. Swap-Based Perturbation Operator. SPO is used to help the two ILSs to avoid local optimum. It iterates the following procedure l rounds: randomly select a pair of tasks in a given task sequence ts and swap them. Obviously, this task swap operator is able to adjust the relative orders of tasks partially and tries to make the two ILSs jump out from local optimum if they have already been trapped in. l is a very important parameter and will be determined by an experiment in Section 5.2. It is obvious that the complexity of SPO is $O(l)$.

4.4. Fast Task Assignment Method. FTA is developed by integrating an acceleration mechanism with TA without loss of any effectiveness. In TA (details of TA can be seen in our previous work [6]), we can find that the makespan corresponding to the case that the task is assigned to the private cloud (i.e., c_{max}^{ij0}) is first calculated and then compared with the one corresponding to the case that the task is

Input: a task sequence ts
Output: makespan c_{max}
1: Set $S \longleftarrow MAX$, $c_{max} \longleftarrow 0$, $budget \longleftarrow B$;
2: Set $cpu_s \longleftarrow CPU^*$ and $mem_s \longleftarrow Mem^*$ for each time slot $s \in \{0, 1, \ldots, S\}$;
3: **for** (each task t_{ij} in ts) **do**
4: Set $st_{ij} \longleftarrow 0$ and $AssignedToPrivateCloud \longleftarrow FALSE$;
5: **while** ($TRUE$) **do**
6: Set $AssignedToPrivateCloud \longleftarrow TRUE$;
7: **for** (each time slot $s \in \{st_{ij}, st_{ij} + 1, \ldots, st_{ij} + r_{ij} - 1\}$) **do**
8: **if** ($\sum_{q=1}^{k} x_{iq} CPU_q > cpu_s$ OR $\sum_{q=1}^{k} x_{iq} Mem_q > mem_s$) **then**
9: Set $AssignedToPrivateCloud \longleftarrow FALSE$ and break;
10: **end if**
11: **end for**
12: **if** ($AssignedToPrivateCloud$) **then**
13: Calculate $c_{max}^{ij0} \longleftarrow \max\{c_{max}, st_{ij} + r_{ij}\}$;
14: break;
15: **else**
16: Set $st_{ij} \longleftarrow st_{ij} + 1$;
17: **if** ($st_{ij} > \max\{c_{max} - r_{ij}, 0\}$ AND $budget \geq LPC_{ij}$) **then** ▷ **The acceleration mechanism using Theorem 1 is utilized**
18: Set $c_{max}^{ij0} \longleftarrow MAX$ and break;
19: **end if**
20: **end if**
21: **end while**
22: Calculate $c_{max}^{ij(IPC_{ij})} \longleftarrow \max\{c_{max}, r_{ij}\}$;
23: **if** ($LPC_{ij} \leq budget$ AND $c_{max}^{ij(IPC_{ij})} < c_{max}^{ij0}$) **then**
24: Set $c_{max} \longleftarrow c_{max}^{ij(IPC_{ij})}$ and $budget \longleftarrow budget - LPC_{ij}$;
25: **else**
26: Set $c_{max} \longleftarrow c_{max}^{ij0}$;
27: **for** (each time slot $s \in \{st_{ij}, st_{ij} + 1, \ldots, st_{ij} + r_{ij} - 1\}$) **do**
28: Update $cpu_s \longleftarrow cpu_s - \sum_{q=1}^{k} x_{iq} CPU_q$;
29: Update $mem_s \longleftarrow mem_s - \sum_{q=1}^{k} x_{iq} Mem_q$;
30: **end for**
31: **end if**
32: **end for**
33: **return** c_{max};

Algorithm 2: FTA.

dispatched to its IPC ($c_{max}^{ij(IPC_{ij})}$). The calculation of c_{max}^{ij0} is time-consuming because we need to determine the task's start time first. However, actually, we do not need to calculate c_{max}^{ij0} for some special cases, since the following theorem is true.

Theorem 1. *In TA, given a task t_{ij} to be scheduled, if $st_{ij} > \max\{c_{max} - r_{ij}, 0\}$ and $budget \geq LPC_{ij}$, the task will be assigned to its IPC.*

Proof. For the case that $c_{max} > r_{ij}$, according to Theorem 1, we have $c_{max}^{ij(IPC_{ij})} = c_{max}$. Additionally, due to $st_{ij} > \max\{c_{max} - r_{ij}, 0\} = c_{max} - r_{ij} > 0$, $st_{ij} + r_{ij} > c_{max}$ is true. And, according to Theorem 1 given in our previous work [6], we have $c_{max}^{ij0} = \max\{c_{max}, st_{ij} + r_{ij}\}$. So, $c_{max}^{ij0} = st_{ij} + r_{ij} > c_{max} = c_{max}^{ij(IPC_{ij})}$. As a result, the task will be assigned to its IPC.

For the case that $c_{max} \leq r_{ij}$, according to Theorem 1, we have $c_{max}^{ij(IPC_{ij})} = r_{ij}$. Additionally, due to $st_{ij} > \max\{c_{max} - r_{ij}, 0\}$, $st_{ij} > 0$ is true. Because of $c_{max} \leq r_{ij}$ and $st_{ij} > 0$, we have $c_{max}^{ij0} = \max\{c_{max}, st_{ij} + r_{ij}\} = st_{ij} + r_{ij} > r_{ij} = c_{max}^{ij(IPC_{ij})}$. Therefore, the task will be assigned to its IPC as well. □

Theorem 1 shows that if $st_{ij} > \max\{c_{max} - r_{ij}, 0\}$ and $budget \geq LPC_{ij}$, the task should be assigned to its IPC without respect to c_{max}^{ij0}. In other words, we do not need to calculate c_{max}^{ij0} for these special cases and the efficiency can thus be improved. Accordingly, we construct an acceleration mechanism that uses Theorem 1 to discover these special cases. As a result, FTA is established by integrating TA with this acceleration mechanism and is described in Algorithm 2.

FTA uses the aforementioned acceleration mechanism (Lines 17-19) to improve the efficiency without loss of effectiveness, which is ensured by Theorem 1. If the two conditions in Line 17 are true, c_{max}^{ij0} is set as a sufficiently large value MAX (so that $MAX > c_{max}^{ij(IPC_{ij})}$) and the "while" loop in Line 5 is

Input: all applications' tasks
Output: makespan c_{max}
1: Determine all tasks' IPCs and calculate their LPCs;
2: Use LTF to generate a task sequence ts;
3: $best \longleftarrow ts$ and $c_{max} \longleftarrow$ FTA(ts);
4: Set $temp \longleftarrow ts$, $c_{max}^{temp} \longleftarrow c_{max}^{ts} \longleftarrow c_{max}$;
5: **while** (termination criterion is not met) **do**
6: $Improved \longleftarrow TRUE$;
7: **while** ($Improved$) **do**
8: $Improved \longleftarrow FALSE$;
9: **for** ($u \longleftarrow 1$ to T) **do**
10: Remove the u-th task in $temp$;
11: Reinsert the removed task in $temp$ at each position except the original one;
12: **for** (each generated task sequence gts) **do**
13: $c_{max}^{gts} \longleftarrow$ FTA(gts);
14: **if** ($c_{max}^{gts} < c_{max}^{ts}$) **then**
15: $Improved \longleftarrow TRUE$;
16: Set $temp \longleftarrow ts \longleftarrow gts$ and $c_{max}^{temp} \longleftarrow c_{max}^{ts} \longleftarrow c_{max}^{gts}$;
17: **end if**
18: **end for**
19: **end for**
20: **end while**
21: **if** ($c_{max}^{ts} < c_{max}$) **then**
22: Set $best \longleftarrow ts$ and $c_{max} \longleftarrow c_{max}^{ts}$;
23: **end if**
24: Set $temp \longleftarrow best$;
25: **for** ($u \longleftarrow 1$ to l) **do**
26: Select a pair of tasks in $temp$ randomly and swap them;
27: **end for**
28: $c_{max}^{temp} \longleftarrow$ FTA($temp$);
29: Set $ts \longleftarrow temp$ and $c_{max}^{ts} \longleftarrow c_{max}^{temp}$;
30: **end while**
31: **return** c_{max};

Algorithm 3: FILS.

terminated. As a result, the two conditions in Line 23 are met; i.e., the task is assigned to its IPC. Obviously, the complexity of FTA is identical to that of TA. Nevertheless, compared with TA, FTA obtains much better efficiency (details can be seen in Section 5.3).

4.5. Description of FILS. Let *best* and c_{max} be the current best found task sequence and its makespan, respectively. Based on the aforementioned LTF, ILSM, SPO, and FTA, we can describe the proposed FILS in Algorithm 3. FILS starts from a task sequence generated by LTF (Line 2). ILSM (Lines 9-19) is iterated until no improvement is obtained (see the condition in Line 7). If a new best task sequence is found, *best* and c_{max} are accordingly updated (Lines 21-23). Afterwards, SPO is invoked to perturb *best* so that FILS can jump out from local optimum (Lines 24-27). Task sequences' makespans are calculated by FTA (Lines 3, 13, and 28).

5. Experimental Results

Following most of existing contributions, we use simulation experiments to evaluate algorithms' performance.

5.1. Testing Instances. We use the testing instances given in our previous work [6]. For the completeness of this paper, we describe these testing instances as follows. Three different Regions (us-east, us-west, and eu-east) of Amazon EC2 and GoGrid are regarded as four public clouds, and 7 different VM types are considered. The configurations and prices are described in Table 2. Note that the prices in Table 2 are shown per hour and we will convert them to values per second when we implement algorithms. The private cloud also provides the same seven VM types.

In order to explore the compared algorithms' performance on problems of different sizes, we consider the total number of tasks (i.e., T) as the problem size factor and construct a Testing Instance Set (TIS), which contains 3 groups with $T \in \{20, 50, 100\}$. Each group contains 3 same-sized subgroups corresponding to 3 different problem types: Small Application Type (SAT), Medium Application Type (MAT), and Large Application Type (LAT). The SAT problem has many small applications that have only a few tasks, while the LAT problem has a few large applications that include lots of tasks. The MAT problem is in between them. So, multiple types of problems are taken into account in this experiment.

Table 2: VM types provided by GoGrid and Amazon EC2.

VM Type	CPU	Memory	Prices (GoGrid)	Prices (EC2 us-east)	Prices (EC2 us-west)	Prices (EC2 eu-east)
t2.micro	1	1	0.02	0.013	0.017	0.014
t2.small	1	2	0.03	0.026	0.034	0.028
t2.medium	2	4	0.06	0.052	0.068	0.056
m3.medium	1	3.75	0.09	0.070	0.077	0.077
m3.large	2	7.5	0.17	0.140	0.154	0.154
m3.xlarge	4	15	0.34	0.280	0.308	0.308
m3.2xlarge	8	30	0.68	0.560	0.616	0.616

Table 3: Instances' sizes in TIS.

Instance Type	T	n	T	n	T	n
SAT	20	$[1, 0.8T]$	50	$[1, 0.8T]$	100	$[1, 0.8T]$
MAT	20	$[1, 0.5T]$	50	$[1, 0.5T]$	100	$[1, 0.5T]$
LAT	20	$[1, 0.2T]$	50	$[1, 0.2T]$	100	$[1, 0.2T]$

Each subgroup has 10 different instances and there are 90 instances in total. In order to generate instances of SAT, MAT, and LAT, the application number n is set as a random integer uniformly distributed within intervals $[1, 0.8T]$, $[1, 0.5T]$, and $[1, 0.2T]$, respectively. Each task is attributed to the n applications with the same probability $1/n$, separately. TIS is summarized in Table 3. The runtime of each task is an integer uniformly distributed in $[1 \times 3600, 24 \times 3600]$ (i.e., from one hour to one day). The VM type required by each application is randomly selected from the 7 considered VM types.

Let VM_b be the best one among all VM types. The budget is set by (11), where λ is a "Budget Factor" used to adjust the budget so that algorithms' performance with different budgets can be explored. According to (11), higher budgets are for larger testing instances.

$$B = T \times p_b \times \lambda \tag{11}$$

The private cloud's available CPU and memory capacities (i.e., CPU^* and Mem^*) are set by (12) and (13), respectively, in which ρ is a "Capacity Factor" used to adjust the two types of resources' capacities so that algorithms' performance with different capacities can be investigated. According to (12) and (13), the private cloud has more available resources for larger testing instances.

$$CPU^* = T \times CPU_b \times \rho \tag{12}$$

$$Mem^* = T \times Mem_b \times \rho \tag{13}$$

5.2. Parameter Determination. The proposed FILS has a parameter, i.e., the number of pairs of swapped tasks (i.e., l) in the SPO. This parameter is determined by experiments in this section. In order to use some statistical methods (e.g., the well-known multifactor analysis of variance (ANOVA)) to evaluate algorithms' performance, we set $\lambda \in \{1, 5, 10\}$ and $\rho \in \{0.05, 0.1, 0.15\}$, respectively. So, there are 9 factors' combinations. Each algorithm is tested on TIS with all possible factors' combinations. All algorithms are implemented in Java and run on the same PC with Dual Core Pentium (R) 3.10 GHz CPU and 4GB Memory. The termination criterion of FILS is set as the maximal number of iterations 100. Relative Error (RE) defined by (14) is adopted to evaluate the performance.

$$RE = \frac{\left(\sum_{r=1}^{R} \left(\left(c_r - c_{LB}\right)/c_{LB}\right)\right)}{R} \times 100\% \tag{14}$$

For each instance, c_r denotes the obtained makespan returned by an algorithm in the r-th replication. R is the total number of replications. c_{LB} represents the makespan's lower-bound, which can be obtained while both the private cloud's resource capacity and budget constraints are assumed to be relaxed. In other words, the private cloud's resource capacity and budget are assumed to be infinite. In this situation, all tasks can be executed in parallel. Accordingly, c_{LB} can be calculated by

$$c_{LB} = \max\left\{r_{ij} \ \ (i = 1, 2, \ldots, n, \ \ j = 1, 2, \ldots, T_i)\right\} \tag{15}$$

Smaller REs indicate better effectiveness since the same c_{LB} is used. Based on the RE for each instance, we further use ANOVA to check whether the differences in the observed average REs are statistically significant. Nonoverlapping confidence intervals between any two pairs of plotted averages mean that the observed differences in such averages are statistically significant at the indicated confidence level. FILS is executed 5 replications (i.e., $R = 5$) since it is metaheuristics including randomness. In order to determine the value of l, we set $l \in \{1, 2, 3, 4, 5\}$. In other words, l has 5 candidates. FILS is tested on TIS with the 5 candidates and the aforementioned two factors' (λ and ρ) 9 combinations. The plot of mean REs and 95% confidence LSD intervals for compared algorithms is given in Figure 2, where FILS1-FILS5 represent FILS with $l = 1, 2, 3, 4, 5$, respectively. Figure 2 shows that the mean REs of FILS with $l = 1, 2, 3, 4, 5$ are 1.083, 0.995, 1.120, 1.157, and 1.189, respectively. The parameter l is thus set as 2 in FILS.

5.3. Performance Evaluation. In order to evaluate FILS's performance, we generate another New TIS (NTIS) by using the

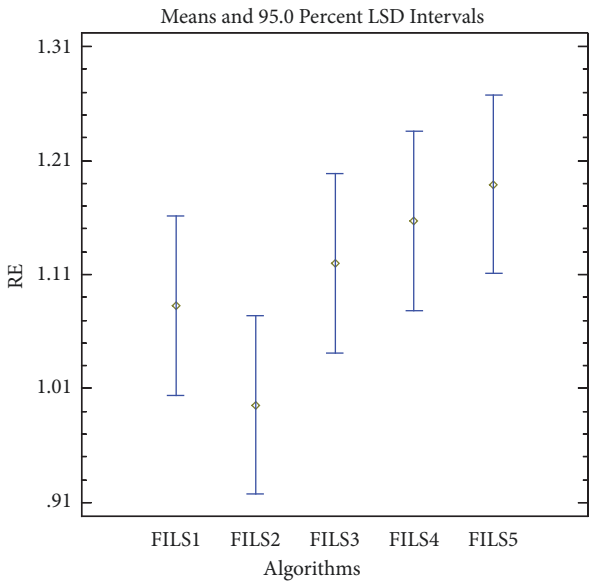

Figure 2: Plot of mean REs and LSD intervals for compared algorithms to determine the l of FILS.

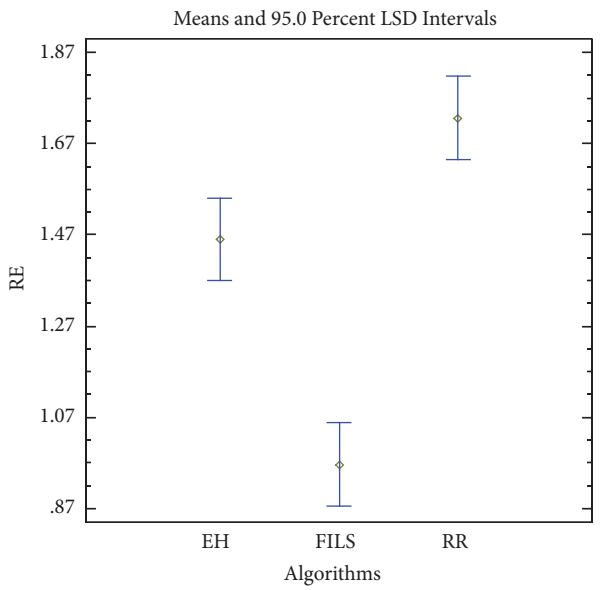

Figure 3: Plot of mean REs and LSD intervals for the compared algorithms.

same rules described in Section 5.1. Though the same rules are used, NTIS is different from TIS used in Section 5.2. EH [6] is the existing best algorithm for the considered problem and is thus regarded to be the baseline, accordingly. Meanwhile, the well-known RoundRobin (RR) is also adopted. In RR, the initial task sequence is generated randomly and each task in the obtained task sequence is assigned to all clouds randomly without violating the private cloud's resource capacity and the budget constraints. Same as those in Section 5.2, the two factors are set as $\lambda \in \{1, 5, 10\}$ and $\rho \in \{0.05, 0.1, 0.15\}$, respectively, whereas the termination criterion of FILS is set as the maximal number of iterations 100. Additionally, RR and FILS are executed 5 replications (i.e., $R = 5$). The plot of mean REs and LSD intervals (95% confidence level) for the compared algorithms is given in Figure 3.

In Figure 3, we can see that the mean REs of EH, FILS, and RR are 1.460, 0.966, and 1.727, respectively. This conclusion shows that FILS outperforms EH that is better than RR, remarkably and significantly. On the side of efficiency, EH and RR consume similar computation times for each testing instance. Their average computation times across over all the testing instances are in the level of tens of milliseconds, whereas that of FILS is in the level of tens of seconds. In other words, compared with the computation times of FILS, those of EH and RR can be negligible. Accordingly, we do not evaluate the efficiencies of all the three compared algorithms together. Instead, with the objective of evaluating the acceleration mechanism employed by FTA, we compare FILS with a Common ILS (CILS) that is the same as FILS except for using TA to calculate task sequences' makespans. For this purpose, we define the Normalized Efficiency (NE) as follows:

$$NE = \frac{t}{t_{baseline}} \tag{16}$$

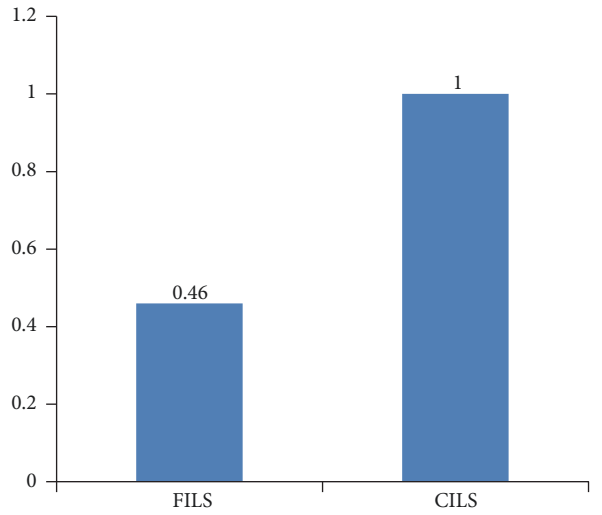

Figure 4: Plot of mean NEs for the two ILSs.

For each instance, t denotes an algorithm's computation time, and $t_{baseline}$ represents the computation time of the baseline which is selected from compared algorithms. Obviously, a lower NE indicates a better efficiency. Without loss of generality, we select CILS as the baseline in this experiment. As both algorithms are executed R replications on each instance, t and $t_{baseline}$ are the means of the R obtained computation times, while we calculate NE for each instance. The mean NEs of the two ILSs are presented in Figure 4, which shows that their mean NEs are 0.46 and 1.0 (CILS is regarded as the baseline), respectively. This conclusion denotes that FILS is much more efficient than CILS, indicating the acceleration mechanism employed by FTA improves the efficiency, considerably. Moreover, we can calculate the speedup achieved by FILS through calculating

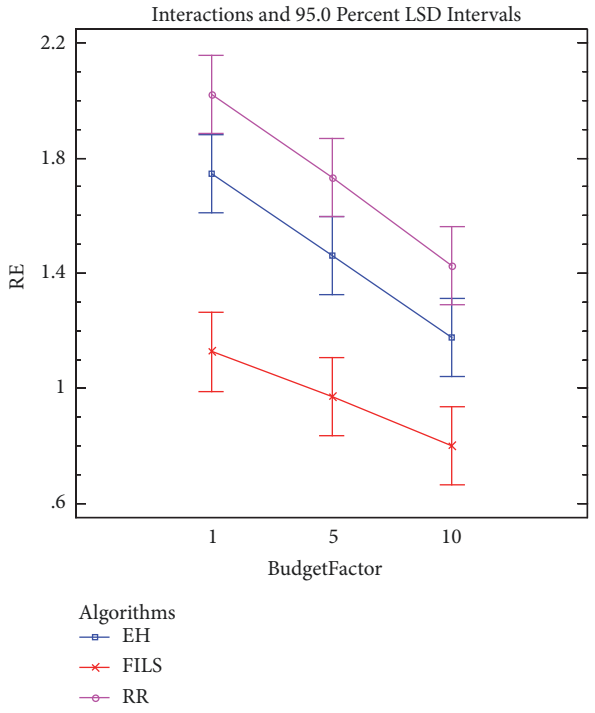

Figure 5: Plot of mean REs and LSD intervals for the interactions between the types of algorithms and the Budget Factor λ.

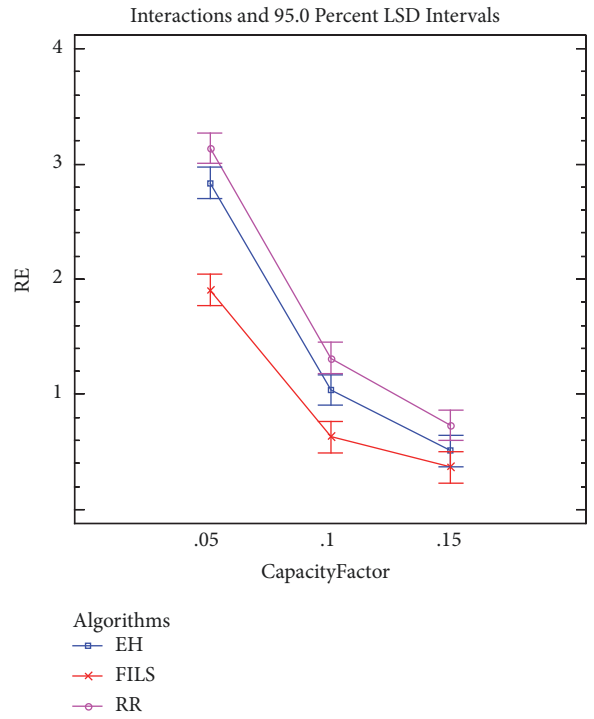

Figure 6: Plot of mean REs and LSD intervals for the interactions between the types of algorithms and the Capacity Factor ρ.

the mean of all instances' speedups, each of which is defined as the reciprocal of NE (i.e., $1/NE$). Accordingly, the speedup obtained by FILS is 2.42. In other words, FILS is 2.42x faster than CILS.

In order to investigate impacts of the two key factors (i.e., the Budget Factor λ and the Capacity Factor ρ), we present the plots of mean REs and LSD intervals (95% confidence level) for the interactions between the types of algorithms and the two factors in Figures 5 and 6, respectively. Figures 5/6 illustrates that mean REs of the three compared algorithms decrease while λ/ρ increases. Actually, this conclusion is reasonable. A larger λ indicates a higher budget, and more tasks can be executed on public clouds in parallel. A larger ρ denotes bigger resource capacity of the private cloud. Accordingly, more tasks can be executed on the private cloud in parallel when the private cloud has more resources. Therefore, we can conclude that the parallelism of task execution can be improved when the two factors are set as large values.

6. Conclusions

This paper schedules Parallel Intrusion Detection Applications (PIDAs) on hybrid clouds to minimize the makespan with the constraints of resource demands and budget. As this problem is NP-Hard, we construct a Fast Iterated Local Search (FILS) algorithm, which employs an effective heuristic to obtain the initial task sequence and utilizes an insertion-neighbourhood-based local search method to explore better task sequences with lower makespans. A swap-based perturbation operator is adopted to avoid local optimum. A Fast Task Assignment (FTA) method is developed by integrating an existing Task Assignment (TA) method with an acceleration mechanism designed through theoretical analysis and is used to calculate task sequences' objectives. Experimental results show that FILS outperforms the existing best algorithm for the considered problem, considerably and significantly. More importantly, compared with TA, FTA achieves a 2.42x speedup, which verifies that the acceleration mechanism employed by FTA is able to remarkably improve the efficiency. Impacts of the two key factors (the Budget Factor and the Capacity Factor) are also investigated with the results showing that the parallelism of task execution can be improved when the two factors are set as large values.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the National Natural Science Foundation of China [Grant nos. 71501096 and 61502234], by Natural Science Foundation of Jiangsu Province [Grant no. BK20150785], by China Postdoctoral Science Foundation

[Grant no. 2015M581801], and by the Fundamental Research Funds for the Central Universities [Grant no. 30916011325].

References

[1] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42–57, 2013.

[2] P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "Intrusion detection techniques in cloud environment: A survey," *Journal of Network and Computer Applications*, vol. 77, pp. 18–47, 2017.

[3] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[4] L. Thai, B. Varghese, and A. Barker, "A survey and taxonomy of resource optimisation for executing bag-of-task applications on public clouds," *Future Generation Computer Systems*, vol. 82, pp. 1–11, 2018.

[5] R. Costa, F. Brasileiro, G. Lemos, and D. Sousa, "Analyzing the impact of elasticity on the profit of cloud computing providers," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1777–1785, 2013.

[6] Y. Zhang, J. Sun, and Z. Wu, "An heuristic for bag-of-tasks scheduling problems with resource demands and budget constraints to minimize makespan on hybrid clouds," in *Proceedings of the 5th International Conference on Advanced Cloud and Big Data, CBD 2017*, pp. 39–44, China, August 2017.

[7] P. Brucker, *Scheduling Algorithms*, Springer-Verlag, 2004.

[8] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters," in *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, pp. 541–548, Rio De Janeiro, Brazil, May 2007.

[9] R. N. Calheiros and R. Buyya, "Energy-efficient scheduling of urgent bag-of-tasks applications in clouds through DVFS," in *Proceedings of the 2014 6th IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2014*, pp. 342–349, Singapore, Singapore, December 2014.

[10] G. Terzopoulos and H. D. Karatza, "Bag-of-task scheduling on power-aware clusters using a DVFS-based mechanism," in *Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2014*, pp. 833–840, Phoenix, Ariz, USA, May 2014.

[11] Y. Zhang, Y. Wang, and C. Hu, "CloudFreq: Elastic energy-efficient bag-of-tasks scheduling in DVFS-enabled clouds," in *Proceedings of the 21st IEEE International Conference on Parallel and Distributed Systems, ICPADS 2015*, pp. 585–592, Melbourne, Australia, December 2015.

[12] A.-M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom '10)*, pp. 351–359, IEEE, Indianapolis, Ind, USA, December 2010.

[13] A.-M. Oprescu, T. Kielmann, and H. Leahu, "Stochastic tail-phase optimization for bag-of-tasks execution in clouds," in *Proceedings of the 2012 IEEE/ACM 5th International Conference on Utility and Cloud Computing, UCC 2012*, pp. 204–208, Chicago, Ill, USA, November 2012.

[14] M. Vasile, F. Pop, R. Tutueanu, and V. Cristea, "HySARC2: Hybrid Scheduling Algorithm Based on Resource Clustering in Cloud Environments," in *Algorithms and Architectures for Parallel Processing*, vol. 8285 of *Lecture Notes in Computer Science*, pp. 416–425, Springer International Publishing, Cham, Switzerland, 2013.

[15] J. O. Gutierrez-Garcia and K. M. Sim, "A family of heuristics for agent-based elastic Cloud bag-of-tasks concurrent scheduling," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1682–1699, 2013.

[16] L. Thai, B. Varghese, and A. Barker, "Executing bag of distributed tasks on the cloud: Investigating the trade-offs between performance and cost," in *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2014*, pp. 400–407, Singapore, Singapore, December 2014.

[17] L. Thai, B. Varghese, and A. Barker, "Budget constrained execution of multiple bag-of-tasks applications on the cloud," in *Proceedings of the 8th IEEE International Conference on Cloud Computing, CLOUD 2015*, pp. 975–980, New York, NY, USA, July 2015.

[18] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing, Grid 2010*, pp. 41–48, Brussels, Belgium, October 2010.

[19] M. H. Farahabady, Y. C. Lee, and A. Y. Zomaya, "Non-clairvoyant assignment of bag-of-tasks applications across multiple clouds," in *Proceedings of the 13th International Conference on Parallel and Distributed Computing, Applications, and Technologies, PDCAT 2012*, pp. 423–428, Beijing, China, December 2012.

[20] I. A. Moschakis and H. D. Karatza, "Multi-criteria scheduling of Bag-of-Tasks applications on heterogeneous interlinked clouds with simulated annealing," *The Journal of Systems and Software*, vol. 101, pp. 1–14, 2015.

[21] I. A. Moschakis and H. D. Karatza, "A meta-heuristic optimization approach to the scheduling of bag-of-tasks applications on heterogeneous clouds with multi-level arrivals and critical jobs," *Simulation Modelling Practice and Theory*, vol. 57, pp. 1–25, 2015.

[22] R. Van Den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD '10)*, pp. 228–235, Miami, Fla, USA, July 2010.

[23] R. Van Den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds," in *Proceedings of the 2011 3rd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2011*, pp. 320–327, Greece, December 2011.

[24] R. van den Bossche, K. Vanmechelen, and J. Broeckhove, "Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 973–985, 2013.

[25] M. Malawski, K. Figiela, and J. Nabrzyski, "Cost minimization for computational applications on hybrid cloud infrastructures," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1786–1794, 2013.

[26] B. Wang, Y. Song, Y. Sun, and J. Liu, "Managing Deadline-constrained Bag-of-Tasks Jobs on Hybrid Clouds," in *Proceedings of the 24th High Performance Computing Symposium*, Pasadena, Calif, USA, 2016.

[27] V. Pelaez, A. Campos, D. F. Garcia, and J. Entrialgo, "Autonomic scheduling of deadline-constrained bag of tasks in hybrid clouds," in *Proceedings of the 2016 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pp. 1–8, Montreal, QC, Canada, July 2016.

[28] S. Abdi, L. PourKarimi, M. Ahmadi, and F. Zargari, "Cost minimization for deadline-constrained bag-of-tasks applications in federated hybrid clouds," *Future Generation Computer Systems*, vol. 71, pp. 113–128, 2017.

[29] X. Zuo, G. Zhang, and W. Tan, "Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 2, pp. 564–573, 2014.

[30] Y. Zhang and J. Sun, "Novel efficient particle swarm optimization algorithms for solving QoS-demanded bag-of-tasks scheduling problems with profit maximization on hybrid clouds," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, Article ID e4249, 2017.