SEED: Confidential Big Data Workflow Scheduling with Intel SGX Under Deadline Constraints

Ishtiaq Ahmed, Saeid Mofrad, Shiyong Lu,
Changxin Bai
Department of Computer Science
Wayne State University
Detroit, Michigan, USA
Email: {ishtiaq, saeid.mofrad, shiyong changxin}@wayne.edu

Fengwei Zhang*
Department of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen, Guangdong, China
Email: zhangfw@sustech.edu.cn

Dunren Che
School of Computing
Southern Illinois University
Carbondale, Illinois, USA
Email: dche@cs.siu.edu

***Abstract*—Recently, cloud platforms play an essential role in large-scale big data analytics and especially running scientific workflows. In contrast to traditional on-premise computing environments, where the number of resources is bounded, cloud computing can provide practically unlimited resources to a workflow application based on a pay-as-you-go pricing model. One challenge of using cloud computing is the protection of the privacy of the confidential workflow's tasks, whose proprietary algorithm implementations are intellectual properties of the respective stakeholders. Another one is the monetary cost optimization of executing workflows in the cloud while satisfying a user-defined deadline. In this paper, we use the Intel Software Guard eXtensions (SGX) as a Trusted Execution Environment (TEE) to support the confidentiality of individual workflow tasks. Based on this, we propose a deadline-constrained and SGX-aware workflow scheduling algorithm, called SEED (SGX, Efficient, Effective, Deadline Constrained), to address these two challenges. SEED features several heuristics, including exploiting the longest critical paths and reuse of extra times in existing virtual machine instances. Our experiments show that SEED outperforms the representative algorithm, IC-PCP, in most cases in monetary cost while satisfying the given user-defined deadline. To our best knowledge, this is the first workflow scheduling algorithm that considers protecting the confidentiality of workflow tasks in a public cloud computing environment.**

***Keywords*-Scientific workflows, Cloud computing, Confidential workflows, Confidential big data processing, Intel SGX and big data security**

I. Introduction

Recently, cloud platforms play an essential role in large-scale big data analytics and especially running scientific workflows [1]. Cloud platforms provide a cost-efficient and enormous amount of computing and storage resources to workflow applications and help them handle more extensive and more complex scientific problems, also known as big data scientific workflows [2]–[5]. One main challenge of using cloud computing is protecting the confidentiality of the confidential workflow's tasks, whose proprietary algorithm implementations are the intellectual properties of the respective stakeholders. Virtual machines (VM)s in the cloud are often subject to insider or outsider attacks, including hypervisor attacks. External adversaries may find vulnerabilities in the cloud hypervisor or cloud system management software, then by using those vulnerabilities, getting unauthorized access, or modifying workflow tasks' code and data. Another main challenge of using cloud computing is the optimization of performance, cost and/or their trade-off, since using VMs in the cloud is costly and the use of more VMs does not necessarily imply better performance due to data movement, the time cost of provisioning and de-provisioning VMs, and the inherent structure of workflow applications. To address these issues, *the big data workflow scheduling problem* is formulated to answer a series of questions: given a workflow w, how many VMs are needed? When do these VMs need to be provisioned and de-provisioned? To which VM should a workflow task be assigned and when? Which data product should be moved from which VM to which VM and when? When is a workflow task ready to be executed? And, how shall we measure the performance of a workflow execution in time and monetary cost? One sub-problem is the so-called *deadline-constrained big data workflow scheduling problem*, which focuses on the monetary cost optimization of executing workflows in the cloud while satisfying a user-defined deadline. However, existing workflow scheduling algorithms have not considered the presence of confidential workflow' tasks, which need their confidentiality to be assured, during their execution in a public cloud. In this paper, we leverage Intel Software Guard eXtensions (SGX) [6] as a tool to create a trusted execution environment (TEE) that guarantees the confidentiality of individual workflow's tasks and data during the runtime. Intel SGX is a set of new CPU instructions introduced to the x86 architecture, aiming to provide confidentiality of code and data at runtime. Intel SGX enables users to create a secure and encrypted area referred to as *enclave* inside the system memory that protects the confidentiality of code and data against strong adversaries. Therefore, when workflow tasks are executed inside enclaves, their confidentiality is assured.

The main contributions of this paper are as following:

1)- We propose a novel deadline-constrained and SGX-aware big data workflow scheduling algorithm, SEED (SGX, Efficient, Effective, Deadline Constrained), that considers confidential tasks, which will be executed in some dedicated SGX-enabled machines.

2)- SEED features several heuristics, including exploiting

Fengwei Zhang* is the corresponding author.

the longest critical paths and reuse of extra times in existing VMs. Our experiments show that SEED outperforms the representative algorithm, IC-PCP [7], in most cases in monetary cost while satisfying the given user-defined deadline. Our comparison uses *the C score*, a performance metric introduced to compare deadline-constrained workflow scheduling algorithms that aim to minimize monetary cost. 3)- The proposed SEED, as well as IC-PCP, have been implemented in a modified version of DATAVIEW, which is one of the most usable big data workflow management systems in the community, to support confidential tasks in a workflow DAG with Intel SGX. Our experiments demonstrate the feasibility and usability of the proposed approach.

II. Related work

Though there are a number of strategies to solve scheduling problems in the clouds, there remains an opportunity to get better result considering QoS. Several metaheuristic approaches, such as particle swarm optimization (PSO) and genetic algorithm (GA) on grids and clouds, are found in the literature [8] which demand a longer period to achieve satisfactory result. An alternate approach [9] optimizes the number of resources that need to be granted for minimizing the execution cost. In [10], VM possession delay was not considered. Another "DCP-C" algorithm initiates instances on Amazon EC2 platform for task scheduling without considering the transfer time from one task to another [11]. SHEFT algorithm optimizes not only the execution time but also the overall cost. However, SHEFT fails to achieve significantly better performance when the number of tasks lies under 200. Moreover, to the best of our knowledge, there is no confidential workflow scheduling algorithm in the literature. In [12] the capacity and implications of using Intel SGX for securing a range of applications from small application to the enterprise and cloud-based workloads has been discussed. *VC3* [13] proposed a secure Map/Reduce framework with Intel SGX to protect the integrity and confidentiality of Map/Reduce job execution and its results. Also, researchers in [14] used Intel SGX and AMD SEV [15] to protect the big data workflow execution in the heterogeneous cloud environments.

III. Scheduling system model

This section describes the overall application model, cloud resource model and problem formulation for workflow in clouds.

A. Application model

An application model can be configured by the four distinct tuple $W(T, D, TSize, Dsize)$ [16], [17], where:

- T represents the set of tasks.
- $D \subseteq T \times T$ is a set of data dependency edges between tasks. We use $D_{i,j}$ to represents data product which is created by t_i and consumed by t_j.
- $DSize : D \rightarrow \mathbb{R}^+$ is the data product size function, and $DSize(D_{i,j})$ returns the size of data product $D_{i,j}$ in MB.

As our algorithm requires two dummy nodes T_{start} and T_{end}, they are connected with zero execution time and zero transfer time to the workflow from beginning and end points to avoid having multiple starting and exit nodes. A sample workflow is shown in Figure 1, depicting all tasks and their dependencies in between, a start node and an end node.

B. Problem definition

Given a DAG workflow W, actual finish time AFT of each of the task and a deadline δ,an IaaS computing environment C, the deadline-constrained workflow scheduling problem is to find the optimal chedule sch_{opt} that optimize the operational cost of executing workflow W within deadline δ [16]:

$$\begin{gathered} sch_{opt} = \underset{sch}{\arg\min}\, WC(sch) \\ subject\ to\ sch.AFT(sch.t_{exit}) \leqslant \delta \end{gathered} \tag{1}$$

C. Some definitions

In order to understand our proposed algorithms, some basic definitions are needed.

1) EST, EFT, and AST: The earliest start time (EST) of a task denotes the time of the task that can executes at its earliest convenience without breaking the given constraints. The minimum execution time (MET). $SV(t_i)$ denotes the selected virtual machine instance for task t_i in a particular schedule. $ET(t_i, SV(t_i))$ stands for the execution time of the predefined service virtual machine $SV(t_i)$ for task t_i. EST is defined by following equation:

$$\max_{t_p \in parents(t_i)} \begin{cases} 0, & \text{(i)} \\ EST(t_p)+ET(t_p,SV(t_p))+TT(e_{p,i}), & \text{(ii)} \\ EST(t_p)+MET(t_p)+TT(e_{p,i}), & \text{(iii)} \end{cases} \tag{2}$$

Similarly, the earliest finish time (EFT) is defined below:

$$EFT(t_i)=\begin{cases} EST(t_i)+ET(t_i,SV(t_i)), & \text{if } SV(t_i) \text{ is defined.} \\ EST(t_i)+MET(t_i), & \text{otherwise.} \end{cases} \tag{3}$$

Finally, $AST(t_i)$ is the actual starting time of t_i on a VM.

2) LFT: Latest finish time (LFT) refers to the time by which an unscheduled task completes its execution so that the whole workflow finishes without breaking the constraint deadline δ. Usually, LFT is calculated recursively from the end task backward in the workflow. Initially, the end task is first calibrated per the given deadline δ and other tasks' $LFT(t_i)$ are then derived as follows in respect to three conditions: i) if $t_i = t_{end}$, ii) if $SV(t_c)$ is defined.

$$\min_{t_c \in children(t_i)} \begin{cases} \delta, & \text{(i)} \\ LFT(t_c)-ET(t_c,SV(t_c))-TT(e_{i,c}), & \text{(ii)} \\ LFT(t_c)-MET(t_c)-TT(e_{i,c}), & \text{(iii)} \end{cases} \tag{4}$$

3) Critical child and critical path: The critical child ($CC(t_i)$) of a particular task t_i is an unassigned task which has the maximal completion time among all its siblings as defined as follows, respectively fulfilling conditions i) if $SV(t_i)$ is defined and ii) otherwise.

$$\arg\max_{t_c \in children(t_i)} \begin{cases} EST(t_i)+ET(children(t_i),SV(children(t_i))), & \text{(i).} \\ EST(t_i)+MET(children(t_i))+TT(e_{i,c}), & \text{(ii).} \end{cases} \tag{5}$$

The critical path is constructed as follows: 1) Every task of the critical path must finish its execution within its LFT. 2) This new schedule utilizes the unused time quota of previously assigned instances.

IV. The proposed workflow scheduling algorithm

Our scheduling approach tries to minimize the execution cost of confidential workflow by first considering the cheapest available resources in IaaS clouds as long as the given deadline constraint can be satisfied. Implementation of our scheduling approach involves three core algorithms, respectively presented and discussed in this section.

A. Initialization algorithm

Algorithm 12, as preparation step of the main algorithm SEED, initializes an input workflow graph for scheduling and then calls for the next core subalgorithm. Line 5 determines the available virtual machines and the corresponding billing time units. Line 6 and 7 initialize the EST of t_{start} and the LFT of t_{end} by 0 and δ, respectively. Lines 8-9 calculate EST, EFT, LFT recursively for the graph G. Line 11 transforms control to Algorithm 2 on the prepared graph G.

Algorithm 1: ScheduleWorkflow (**G**)

```
1  $startNodes \leftarrow nodesWithoutParent$ in G;
2  $endNodes \leftarrow nodesWithoutChildren$ in G;
3  connect $t_{start}$ with $startNodes$;
4  connect $t_{end}$ with $endNodes$;
5  determine the available virtual machines with cost and billing unit
     cycle;
6  $EST(t_{start}) \leftarrow 0$;
7  $LFT(t_{end}) \leftarrow$ Deadline $\delta$;
8  calculate $EST$, $EFT$ from $t_{start}$ to $t_{end}$ recursively;
9  calculate $LFT$ from $t_{end}$ to $t_{start}$ recursively;
10 $G \leftarrow$ remove $t_{start}$, $t_{end}$, and edges in between from G;
11 $AssignChildren(G)$;
12 return optimized schedule;
```

B. Assigning children algorithm

Algorithm 2 is the procedure of assigning children. In line 1, graph G is sorted in topological order and assigned to a LinkedList $list$. At lines 2-21 the algorithm iterates on the list till it becomes empty. Line 3 initializes $criticalPath$ as empty. The algorithm picks the first element of $list$ and assign it to t_i (lines 4-5); then add t_i to $criticalPath$. Afterward, via the while loop (lines 7-10) the algorithm constructs the critical path by recursively adding a critical child along the path (level by level). The critical path is then split (at line 11) as multiple paths by confidential tasks along the path as separators. Afterwards, each of the paths is in turn allocated to a virtual machine (lines 12-20).

Algorithm 2: AssignChildren (**G**)

```
1  $list \leftarrow$ sort tasks in $G$ in topological order;
2  while $list \neq \emptyset$ do
3      $criticalPath \leftarrow \emptyset$;
4      $t_i \leftarrow list[1]$                          ▷ Selecting first task
5      add $t_i$ in $criticalPath$;
6      while $t_i$ has children do
7          $t_i \leftarrow criticalChild(t_i)$;
8          add $t_i$ in $criticalPath$;
9      end
10     $paths \leftarrow$ split apart $criticalPath$ by confidential tasks as
         separators;
11     foreach $path \in paths$ do
12         $AllocateVirtualMachine(path)$;
13         foreach $t_i \in path$ do
14             recalculate LFT of the ancestors $\{t_k : t_k \notin path\}$
                 of $t_i$;
15             recalculate EST, EFT of the descendants
                 $\{t_k : t_k \notin path\}$ of $t_i$;
16         end
17         $G \leftarrow$ remove all tasks and edges involving tasks in $path$
             from $G$;
18         $list \leftarrow$ remove all tasks involving in $path$ from $list$;
19     end
20 end
21 return optimized scheudle;
```

C. Allocating Virtual Machines

Algorithm 3 allocates the tasks in $path$ (input parameter) to a virtual machine. We use variable VMI^c to keep track of all existing VM instances launched for the workflow G under processing; We introduce variable vmi^t to generally denote a VM instance of type t. Algorithm 3 starts by initializing (line 1) VMI^c to the set of all existing VM instances launched for workflow G. If a secure execution environment is required for the tasks in $path$, a secure VM instance vmi^{SGX} is launched for $path$ and added to T^c (lines 3-4). The big loop (line 6-32) then iterates on each of the existing instance $vmi \in VMI^c$ in ascending order of cost rates. Line 8 retrieves the queue of tasks previously assigned to vmi in turn into Q. Then it tries to find the cheapest (but capable) existing VM instance for executing the tasks in $path$. Three different cases are in turn explored. Case 1 (lines 10-17) succeeds when a task $t_k \in Q$ is found being a child of the last task in $path$ – in this case, the algorithm inserts $path$ into Q after task t_k and recalculates EST, EFT, LFT of the affected tasks in Q. If the updated assignment of tasks to vmi is a valid one (satisfying the deadline constraint), we are simply done. Otherwise, we restore the old task assignment of vmi and continue to explore subsequent cases. Case 2 (lines 19-24) tries to insert $path$ at the beginning of

Q, if resulting in a new valid assignment, we are simply done. Otherwise, we restore the old assignment of vmi and continue to the next case. Case 3 (lines 26-31) tries to insert $path$ at the end of Q. Similar to prior cases, if resulting in a new valid assignment, we are simply done. Otherwise, we restore the old assignment of vmi, and continue with the next iteration. After quitting the loop (lines 6-32), we know we cannot find an existing VM instance suitable the tasks in $path$.

Algorithm 3: AllocateVirtualMachine (**path**)

```
1  $VMI^c \leftarrow$ existing-VM-instances;
2  if secure environment is required for $path$ then
3      launch a new instance $vmi^{SGX}$ for $path$;
4      $VMI^c \leftarrow vmi^{SGX}$;
5      return;
6  foreach existing VM instance $vmi \in VMI^c$ in ascending order
     of cost do
7      oldAssignment ← assignment($vmi$);
8      $Q \leftarrow assignment(vmi)$;                 ▷ queued tasks at $vmi$
9                                   ▷ Case 1: Insert $path$ into middle of $Q$
10     if there exists a task $t_k \in Q$ such that $t_k$
         $\in path[last].children$ then
11         insert $path$ into $Q$ before $t_k$ and recalculate EST, EFT,
             LFT of affected tasks in $Q$;
12         if assignment($vmi$) is valid then
13             return;              ▷ keep current $assignment(vmi)$
14         else
15             assignment($vmi$) ← oldAssignment;
16         end
17     end
18                                   ▷ Case 2: Insert $path$ at head of $Q$
19     insert $path$ at head $Q$ and recalculate EST, EFT, LFT of
         affected tasks in $Q$;
20     if assignment($vmi$) is valid then
21         return;                  ▷ keep current $assignment(vmi)$
22     else
23         assignment($vmi$) ← oldAssignment;
24     end
25                                   ▷ Case 3: Insert $path$ at end of $Q$
26     insert $path$ at end of $Q$ and recalculate EST, EFT, LFT of
         affected tasks in $Q$;
27     if assignment($vmi$) is valid then
28         return;                  ▷ keep current $assignment(vmi)$
29     else
30         assignment($vmi$) ← oldAssignment;
31     end
32 end
33                     ▷ No existing VM instance can accommodate $path$
34 launch a new cheapest but capable VM instance $vmi^c$ for
     executing the tasks in $path$ before their LFT;
35 $VMI^c \leftarrow vmi^c$;
36 return VM allocation of tasks;
```

D. An illustrative example

Figure 1 includes ten tasks starting from t_1 to t_{10} connecting by edges which represents the dependencies in between them. Two additional tasks t_{start} and t_{end} are added to the graph. While connecting these two tasks in the graph, transfer time is omitted and considers zero execution cost. We assume there are three different types of T such as T^1, T^2 and T^3 for executing this workflow considering T^1 is the fastest machine, demanding more money, while T^3 is the slowest with the cheapest rate. Table II portrays numerous execution times for various tasks into above mentioned Ts. We also assume the time interval for leasing VM instance is 30 minutes and the overall deadline is 70 minutes. Furthermore, each of the Ts charges 5$, 8$ and 13$, for T^3, T^2 and T^1 accordingly. We also consider T^{SGX} for confidential execution of tasks and we also consider the execution time t_9 in SGX machine demands 7 minutes accordingly.

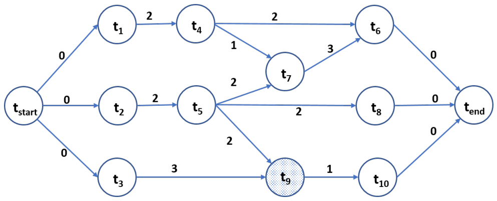


Figure 1: Sample workflow representing nodes as tasks and arcs as dependencies.

First, let us consider that the workflow does not have any confidential tasks. EST, EFT, LFT values are initialized in Table I. The numbers which are altered from the previous step, a (†) symbol is inserted after the corresponding values. At the very beginning, running topological sorting produces sorted order: t_1, t_4, t_2, t_5, t_8, t_7, t_6, t_3, t_9 and t_{10}. In Step 1.0, task t_1 from this list constructs critical path $\{t_1, t_4, t_6\}$. Since T^3 gives us the cheapest strategy, we instantiated the first instance VMI_1^3 and updated their EST, EFT, LFT. In Step 2.0, choosing t_2 from the first element of list, produces critical path $\{t_2, t_5.t_8\}$. Step 3.0: selecting first element t_7 from $list$, creates critical path, $\{t_7\}$. We observe that the unused time of VMI_1^3 is greater than the execution time of this critical path, then we insert this path before t_6 and and shifting t_6 right side produces $\{t_1, t_4, t_7, t_6\}$. In Step 4.0, choosing t_3 from $list$ generates new critical path $\{t_3, t_9, t_{10}\}$. When the scheduling procedure is finished, then all the tasks are allocated into different VM instances according to Figure 2a. The overall cost is $41 while IC-PCP charges $51 according to 2b.

Now let us consider that t_9 is required to execute in confidential environment T^{SGX}. Since the Step 1.0, 2.0 and 3.0 are identical to previous non-confidential schedules, confidential task t_9 splits apart this critical path into three different paths: $\{t_3\}$, $\{t_9\}$, and $\{t_{10}\}$. Then these three different paths are assigned one after another. Table V depicts the changes of EST, EFT, and LFT for each steps.

V. Securing DATAVIEW with SGX TEE

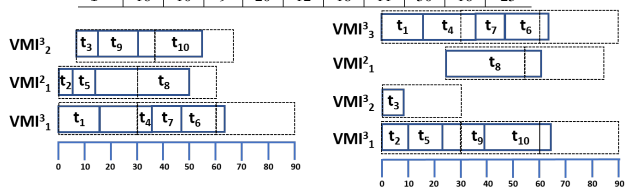
DATAVIEW [2] allows us to execute big data workflows in the cloud. This system leverages Amazon EC2 AWS for running tasks in the cloud. However, the existing solution does not protect the confidentiality of the security-sensitive tasks during the runtime. We refer our reader to the original DATAVIEW paper for detailed information [2]. We have

Table I: Changes of EST, EFT and LFT for execution of SEED for non-confidential tasks.

Tasks		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
Primary	EST	0	0	0	6	5	21	15	11	11	19
	EFT	4	3	3	14	9	31	18	36	18	37
	LFT	43	36	41	53	42	70	57	70	51	70
Step 1.0	EST	0	0	0	$16^\dagger$	5	$36^\dagger$	$37^\dagger$	11	11	19
	EFT	$16^\dagger$	3	3	$36^\dagger$	9	$54^\dagger$	$40^\dagger$	36	18	37
	LFT	$32^\dagger$	36	41	$52^\dagger$	42	70	$49^\dagger$	70	51	70
Step 2.0	EST	0	0	0	16	$6^\dagger$	36	37	$15^\dagger$	$17^\dagger$	$25^\dagger$
	EFT	16	$6^\dagger$	3	36	$15^\dagger$	54	40	$51^\dagger$	$24^\dagger$	$43^\dagger$
	LFT	32	$25^\dagger$	41	52	$34^\dagger$	70	49	70	51	70
Step 3.0	EST	0	0	0	16	6	$48^\dagger$	37	15	17	25
	EFT	16	6	3	36	15	$66^\dagger$	$48^\dagger$	51	24	43
	LFT	32	25	41	52	34	70	49	70	51	70
Step 4.0	EST	0	0	$8^\dagger$	16	6	48	37	15	17	$33^\dagger$
	EFT	16	6	$17^\dagger$	36	15	66	48	51	$33^\dagger$	$58^\dagger$
	LFT	32	25	$29^\dagger$	52	34	70	49	70	$45^\dagger$	70
Final allocation		VMI_1^3	VMI_1^2	VMI_2^3	VMI_1^3	VMI_1^2	VMI_1^3	VMI_1^3	VMI_1^2	VMI_2^3	VMI_2^3

Table II: Execution time matrix.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
T^1	4	3	3	8	4	10	3	25	7	18
T^2	10	6	6	12	9	15	7	36	10	20
T^3	16	10	9	20	12	18	11	50	16	25



(a) Schedule generated by SEED on non-confidential workflow.

(b) Schedule generated by IC-PCP on non-confidential workflow.

Figure 2: Task allocation comparison among Non-confidential SEED and IC-PCP with 30 minutes time interval.

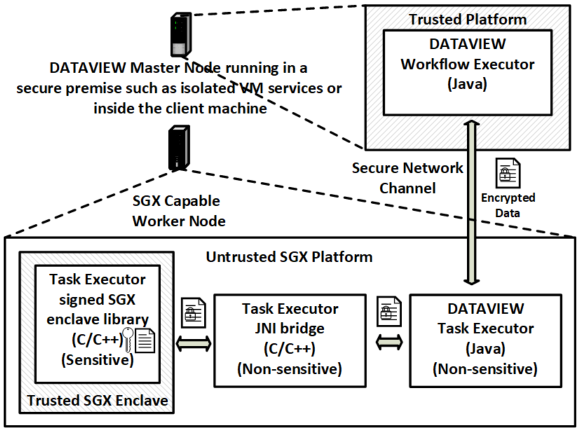


Figure 3: The architecture of DATAVIEW Task Executor with SGX support.

modified DATAVIEW to support the SEED algorithm and Intel SGX-enabled servers. In the modified part of the DATAVIEW that visualized in Figure 3, there are two Java-written modules referred to as Workflow Executor and Task Executor. Workflow Executor is executed in secure premises, and Task Executor is performed in SGX platforms. After deploying the Task Executor, it first initializes its Java component that includes non-sensitive code that only used for invoking Java Native Interface (JNI) calls and passing encrypted data and results. Then Task Executor creates an SGX enclave and invokes the crafted SGX task, including workflow's task code on its security-sensitive data after decrypting data inside the enclave. After a task job is finished, Task Executor first encrypts its results and then it sends the task's encrypted results to its children tasks in the workflow. We used Authenticated Encryption with Associated Data (AEAD) in AES-256 GCM mode and SSL protocol for file transfer and protecting data confidentiality at rest. In this prototype we assume the presence of a secure key exchange protocol through an SGX hardware and enclave attestation mechanism such as Intel Remote Attestation protocol [18] that assures the authenticity of Intel SGX CPU and Task Executor's enclave to provision the shared encryption secrets between the enclaves and data owner. We also put the denial-of-service, network traffic analysis, SGX side-channels, fault injections, and access pattern leakage attacks out of this paper's scope.

Table III: Non-confidential execution cost distribution for various VMIs

Type	Start	End	Duration	Interval cycle	Cost	Selected task
VMI_1^3	0	66	66	3	15	$\{t_1, t_4, t_7, t_6\}$
VMI_1^2	0	51	51	2	16	$\{t_2, t_5, t_8\}$
VMI_2^3	8	58	50	2	10	$\{t_3, t_9, t_{10}\}$

Table IV: Confidential execution cost distribution for various VMIs

Type	Start	End	Duration	Interval cycle	Cost	Selected task
VMI_1^3	0	66	66	3	15	$\{t_1, t_4, t_7, t_6\}$
VMI_1^2	0	51	51	2	16	$\{t_2, t_5, t_8\}$
VMI_2^3	0	9	9	1	5	$\{t_3\}$
VMI_1^{SGX}	17	24	7	1	25	$\{t_9\}$
VMI_3^3	25	50	25	1	5	$\{t_{10}\}$

VI. Performance evaluation

A. Experimental testbed for confidential experiments

To conduct experiments on confidential workflow tasks, we use AWS VM instances and a physical Intel SGX server. Figure VI visualizes our testbed configuration. For the software configuration, all Amazon AMI use Ubuntu

Table V: Changes of EST, EFT and LFT for execution of SEED for confidential tasks.

Tasks		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
EST, EFT, and LFT of Step 1.0, 2.0, 3.0 are same as non-confidential tasks											
Step 4.0	EST	0	0	0	16	6	48	37	15	17	25
	EFT	16	6	$9^{\dagger}$	36	15	66	48	51	24	$50^{\dagger}$
	LFT	32	25	41	52	34	70	49	70	51	70
Final allocation		VMI_1^3	VMI_1^2	VMI_2^3	VMI_1^3	VMI_1^2	VMI_1^3	VMI_1^3	VMI_1^2	VMI_1^{SGX}	VMI_3^3

Table VI: Testbeds Configuration.

Testbed Machine	Intel SGX	AWS
CPU Model	Xeon E3-1275 V6	Intel(R) Xeon(R) CPU E5-2676 v3
CPU Physical Core Number	4	8
CPU Logical Thread Number	8	16
CPU Base Clock	3.8GHz	2.4GHz
CPU Boost Clock	4.2GHz	3.2GHz
Cache Type	Smart Cache	Smart Cache
Cache Size	8MB	20MB
Motherboard	Supermicro X11SSW-F	AWS
Memory	32GB DDR4 ECC	1GB DDR4 ECC
Storage	HDD	EBS
Operating System	CentOS 7.0	Linux 16.04 LTS
Kernel Version	3.10.0-862.9.1.el7.x86_64 x86_64	4.4.0-1060-aws
SGX SDK Version	SGX SDK Ver 2.2	SGX SDK Ver 2.0
AMI family	N/A	General purpose
AMI type	N/A	t2.micro
AMI Assigned CPU Core	N/A	1

16.04 LTS OS with the kernel version 4.4.0-1060-aws. Since AWS lacks the presence of real SGX TEE, we installed Intel SGX SDK in software mode so, all AMI instances could execute Intel SGX applications that compile with Intel SGX simulation library.

B. Performance comparison between SEED and IC-PCP with non-confidential tasks

To perform a comparison between these two algorithms, we synthetically generate several workflow samples based on the structure of the Montage workflow, in which the computation code of each task is simulated by a bubble sorting algorithm that performs sorting on varying size of numbers. We use three different types of Amazon VM, t2.micro, t2.large and t2.xlarge to execute these tasks, where they offer different computational power. Here, t.micro is the slowest VM whereas t2.large is the fastest VM. The average network traffic bandwidth between tasks in Amazon VMs is $20MBps$. Since the above mentioned Amazon VM types do not provide a stable performance, the overall workflow execution time might not meet the deadline. We introduce the C score formulated below as the performance metric for comparing two schedules produced by different algorithms.

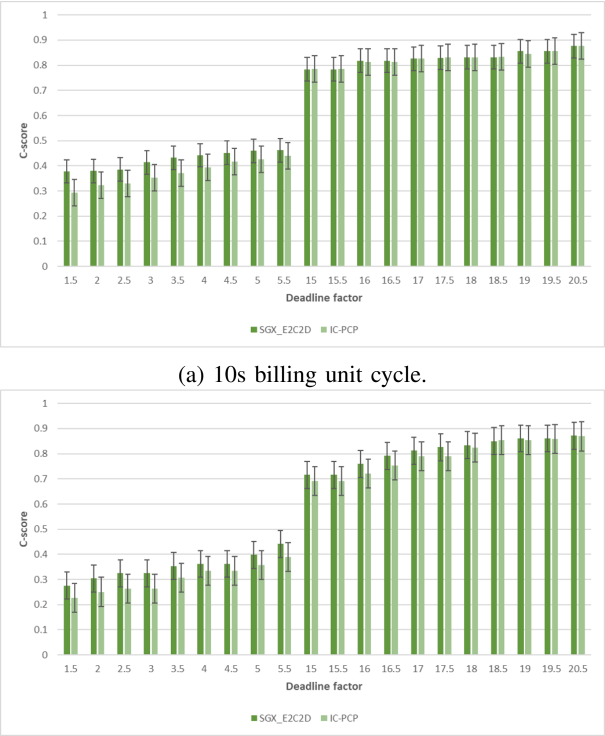
$$\text{C}=\begin{cases} 0.5+0.5*\frac{(maxcost-cost)}{maxcost}, & if\ makespan \leq D \\ 0.5-0.5*\frac{(makespan-D)}{(maxmakespan-D)}, & otherwise \end{cases} \tag{6}$$

The C score assigns 0.5 to each schedule initially and then provides a reward between 0.0 to 0.5 to those schedules that meet the deadline, and imposes a penalty between 0.0 to 0.5 on those schedules that miss the deadline. Therefore, the C score provides a value between 0.0 and 1.0 and has the following three nice properties. P1: all schedules that meet the deadline have a score between 0.5 and 1.0 and all schedules that miss the deadline have a score between 0.0 and 0.5; P2: for the schedules that meet the deadline, lower cost implies larger C score, and in particular, when the cost is near 0.0, the C score reaches near 1.0; P3: for the schedules that miss the deadline, cost becomes irrelevant to the C score since for this case, the execution time becomes the critical component, and the larger the makespan, the more penalty to the C score, and in the worst case, when the execution time reaches *maxmakespan*, the largest penalty 0.5 is applied, and the C score becomes 0.

Before running the experiment, we run each workflow task in three different types of amazon VM to collect their execution time of each task on each VM type. We denote the makespan for the fasteset schedule of the workflow as M_f and then set the deadline of the workflow is the arrival of the the workflow plus $\chi * M_f$, where χ ranges from 1.5 to 5.0 and from 15.0 to 20.5 with a step of 0.5. The Montage workflow condists of 28 tasks. Our experiments are conducted for two different billing unit cycles: 10s and 60s. We assume the fastest machine charges roughly 3X than the slowest machine. For every configuration, we run the workflow execution 10X and record the average values. Figure 4 depicts the experimental results for running both algorithms, when χ ranges from 1.5 to 5, both algorithms miss the deadline, resulting in schedules with C scores smaller than 0.5. The reason for failing the deadline is because sending files, network connection instability takes extra time. Since both algorithms fail to meet the deadline, their C scores suffer from a penalty. However, SEED performs better in most cases by taking less amount of additional time in comparison with IC-PCP, as SEED uses the longest critical path and utilization of unused time of existing instances. Hence less amount of transfer time is needed for sending required files from one machine to another. When we increase the deadline factor, the schedules produced by both algorithms have relatively better C scores. In particular, when $\chi \geq 15$, the schedules produce by both algorithms meet the deadline, and the C-scores reach around 0.8. For billing cycle of 60s, the C score is relatively smaller than that for 10s. As relatively longer billing cycle might have longer unused time, it requires an extra budget for VM allocation and hence the C score falls down. We also conduct experiments on the LIGO workflow with 28 tasks and observe a similar performance result.

C. Confidential tasks performance evaluation

In the first experiment, we use Algorithm 1 task from [19], which is a real medical diagnosis workflow task and by varying different input size of the patients for both a physical SGX and regular machine as the baseline, we record the execution time. To make a fair comparison environment for $Algorithm1$, the conventional machine uses the same



(a) 10s billing unit cycle.

(b) 60s billing unit cycle.

Figure 4: Performance analysis in Montage workflow.

C/C++ and JNI code identical to the code that is used for the confidential version which executes inside Intel SGX enclave. Also, we isolate the $Algorithm1$ task and do not consider the execution of the complete workflow to remove possible noises and slowdown imposes by a network issue or other dependency and tasks in the workflow. The input dataset starts from $1,000$ labeled instances and 250 unlabeled instances, and gradually we increase the size to $150k$ labeled and $37.5k$ unlabeled instances of the patients. The confidential task requires additional steps to get the job done. First, it needs to initialize a $2.93GB$ enclave; we use large enclave on purpose to record the performance overhead of a large enclave in the big data context. After that, it decrypts the received file and then executes the $Algorithm1$ code on the plain data. Finally, it encrypts the results and generates an encrypted file. The cryptography algorithm is AEAD AES-256 in GCM mode, and we assume the confidential task has received the shared secrets through a secure protocol. Our results show a range of $1.24X$ to $1.31X$ performance overhead over its baseline execution with $96k$-$150k$ labeled and $24k$-$37.5k$ unlabeled instances, respectively. In Figure 5, we iterate each of the experiment for ten trials and collect the average values. Figure 5a depicts the performance result in ms.

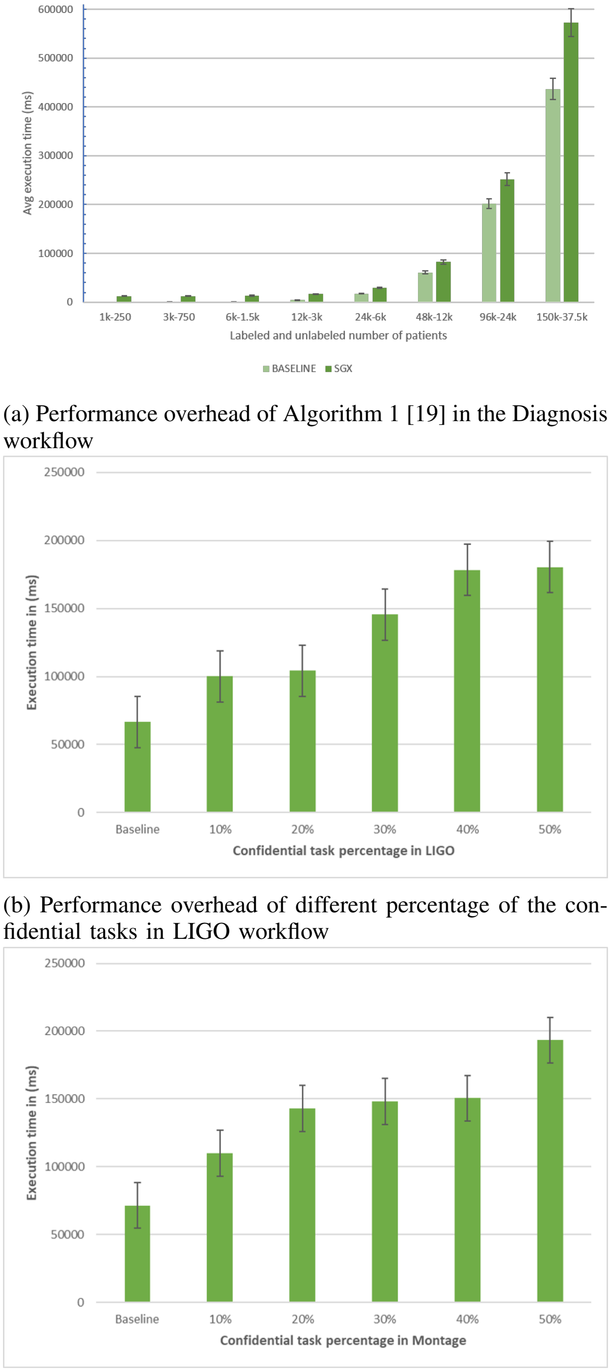
In the second experiment, we leveraged Intel SGX in the simulation mode and with AWS instances. We prepared an SGX-enabled AMI by setting up Intel SGX SDK in software mode. Since Amazon EC2 VMs do not support Intel SGX in the hardware mode at this time, all confidential tasks are compiled with the SGX simulation library, which is part of the Intel SGX SDK. We generate a dummy LIGO workflow with 10 tasks. Each task generates $100k$ random integers and sorts it with the bubble sort algorithm and afterward send the sorted result to the next child node in the workflow. We start with 10% of confidential tasks and gradually increase the percentage of confidential tasks in our dummy LIGO workflow up to 50%. Figure 5b presents the LIGO workflow results depicting the execution times of certain confidential tasks. We notice that in our baseline where all the tasks are non-confidential, the execution time is the smallest one. When we increase the percentage of confidential tasks, the overall execution time increases as well. The LIGO workflow's average performance overheads show a range between $1.81X$ in 10% and increases to $2.50X$ in 50% confidential task assignments. In the third experiment, we generate a Montage workflow with 10 tasks, and then we follow the experiment identically to the LIGO workflow, as mentioned earlier, then record the execution times. Figure 5c depicts the Montage workflow results. We observe that the execution time gradually increases as we increase the number of confidential tasks. The performance overhead of confidential workflow execution is varied since SGX settings take some extra time for the initial setup and apply cryptography on input and result data. Our results show a range of $1.34X$ to $2.50X$ performance overhead for the Montage workflow. These experiments demonstrate that our proposed algorithm with modified DATAVIEW provides security and confidentiality support with reasonable overhead.

VII. Conclusions

This paper presents an efficient, cost-effective deadline constrained algorithm SEED for scientific workflows in IaaS clouds. To the best of our knowledge, this is one of the first scheduling algorithms that can handle confidential workflow tasks. The primary intention is to schedule confidential tasks and achieve a low-cost scheduling algorithm within the given deadline while the complexity remains feasible for running large-scale workflows. We also modified the DATAVIEW framework to support confidential tasks in a workflow DAG with Intel SGX TEE. Our experimental results with different workflows show an acceptable and low amount of performance overhead while leveraging Intel SGX TEE to protect confidential tasks execution at runtime.

Acknowledgement

This work is supported by National Science Foundation, under grant NSF OAC-1738929.



(a) Performance overhead of Algorithm 1 [19] in the Diagnosis workflow

(b) Performance overhead of different percentage of the confidential tasks in LIGO workflow

(c) Performance overhead of different percentage of the confidential tasks in Montage workflow

Figure 5: Execution time overhead of running Algorithm 1 [19] of Diagnosis workflow, LIGO and Montage in SGX and baseline.

References

[1] M. H. Ghahramani, M. Zhou, and C. T. Hon, "Toward cloud computing QoS architecture: Analysis of cloud systems and cloud services," *IEEE/CAA Journal of Automatica Sinica*.

[2] A. Kashlev and S. Lu, "A system architecture for running big data workflows in the cloud," in *SCC*. IEEE, 2014.

[3] Q. Wu, M. Zhou, Q. Zhu, Y. Xia, and J. Wen, "Moels: Multiobjective evolutionary list scheduling for cloud workflows," *IEEE Transactions on Automation Science and Engineering*.

[4] L. Wang and J. Lu, "A memetic algorithm with competition for the capacitated green vehicle routing problem," *IEEE/CAA Journal of Automatica Sinica*, 2019.

[5] I. Ahmed, "Scheduling and securing big data workflows in the cloud with heterogeneous trusted execution environments," ProQuest & Wayne State University, 2019.

[6] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution." 2013.

[7] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*.

[8] W. Li, Y. Xia, M. Zhou, X. Sun, and Q. Zhu, "Fluctuation-aware and predictive workflow scheduling in cost-effective infrastructure-as-a-service clouds," *IEEE Access 2018*.

[9] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Generation Computer Systems*, 2011.

[10] K. Liu, *Scheduling algorithms for instance-intensive cloud workflows*. Swinburne University of Technology, Faculty of Engineering and Industrial Sciences, Centre for Complex Software Systems and Services, 2009.

[11] S. Ostermann and R. Prodan, "Impact of variable priced cloud resources on scientific workflow scheduling," in *European Conference on Parallel Processing*. Springer, 2012.

[12] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A comparison study of Intel SGX and AMD memory encryption technology," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2018.

[13] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "VC3: Trustworthy data analytics in the cloud using SGX," in *IEEE S&P 2015*.

[14] S. Mofrad, I. Ahmed, S. Lu, P. Yang, H. Cui, and F. Zhang, "SecDATAVIEW: a secure big data workflow management system for heterogeneous computing environments," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 390–403.

[15] D. Kaplan, J. Powell, and T. Woller, "AMD memory encryption," *White paper, Apr*, 2016.

[16] C. Bai, S. Lu, I. Ahmed, D. Che, and A. Mohan, "LPOD: A local path based optimized scheduling algorithm for deadline-constrained big data workflows in the cloud," in *2019 IEEE International Congress on Big Data (BigDataCongress)*. IEEE, 2019, pp. 35–44.

[17] S. Z. M. Mojab, M. Ebrahimi, R. Reynolds, and S. Lu, "iCATS: Scheduling big data workflows in the cloud using cultural algorithms," in *2019 IEEE Big Data Service*, 2019.

[18] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *HASP@ ISCA*, 2013.

[19] I. Ahmed, S. Lu, C. Bai, and F. A. Bhuyan, "Diagnosis recommendation using machine learning scientific workflows," in *IEEE Big Data Congress 2018*.