2019 IEEE International Congress on Big Data (BigData Congress)

LPOD: A Local Path Based Optimized Scheduling Algorithm for Deadline-Constrained Big Data Workflows in the Cloud

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

Dunren Che†
Department of Computer Science
Southern Illinois University†
Carbondale, Illinois, USA
Email: dche@cs.siu.edu†

Aravind Mohan‡
Department of Computer Science
Allegheny College‡
Meadville, Pennsylvania, USA
Email: amohan@allegheny.edu‡

***Abstract*—List based scheduling algorithms have been proven an optimistic strategy with a shorter response time to generate feasible solutions for the workflow scheduling problem. Data-intensive and computation-intensive workflow applications have different characteristics in terms of the ratio between data transfer time and task execution time. Workflow scheduling algorithms in a cloud-based environment should adequately consider the characteristics of the underlying cloud platform such as the on-demand resource provisioning strategy, the practically unlimited compute capacities, the booting times of virtual machines, the homogeneous network and the pay-as-you-go price model to produce an optimal scheduling solution within the deadline constraint of a given workflow. In this paper, a path based scheduling algorithm, named LPOD, is proposed to find the best workflow schedule solution with minimum monetary cost in a cloud computing environment. A series of case studies have been carefully conducted using synthetic workflows based on DATAVIEW, which is a popular open-source big data workflow management system. The experimental results show that the proposed algorithm is efficient and can generate better workflow schedules than the state-of-the-art algorithms such as IC-PCP and SGX-E2C2D.**

***Keywords*-Workflow; Cloud; Deadline Constrained; Scheduling; Optimization;**

1. Introduction

Workflow as a parallel high-performance computing model has been broadly studied and used to orchestrate different computation tasks involving complex data dependencies in various research domains such as astronomy [1], biology [2] [3], seismology [4], medical care [5] [6], and big data analysis [7]. A workflow is typically represented as a directed acyclic graph (DAG), in which nodes represent tasks and edges represent data dependencies between two consecutive tasks. Modern Workflow management systems such as DATAVIEW [8] facilitate and automate the complex execution process of a workflow on top of a high-performance computing platform through a user-friendly interface.

In the last few years, cloud computing has become a promising paradigm over the Internet for scientific computing applications. Cloud computing comes with numerous benefits: elasticity, the pay-as-you-go pricing model, practically unlimited capacity of compute resources, etc., which are particularly suitable for exploratory scientific computing applications. Scientific workflow management systems [8] have started to embrace cloud computing, especially Infrastructure-as-a-Service (IaaS), as their new underlying computing platforms to deliver high-performance computing capabilities to scientific applications. Virtual machine (VM) instances are the fundamental compute infrastructure to be leveraged by scientific workflow systems to facilitate scientific workflow applications.

Workflow scheduling is the most challenging core module in any workflow management systems [9] [10]. Workflow scheduling by its nature is DAG scheduling, which is a well-known NP-complete [11] problem, even in the simplest case, where tasks are assigned to an arbitrary number of resources. Given a workflow, an ideal scheduling algorithm is expected to generate a schedule solution of minimal monetary cost and minimal makespan of the workflow, which is apparently a paradox. How to balance between makespan and execution cost is the key research issue in workflow scheduling. This issue has been studied in respective cases with different focuses: the deadline-constrained workflow scheduling [12], budget-constrained workflow scheduling [13], and the deadline-budget optimization workflow scheduling [14] [15]. This paper focuses on deadline-constrained scheduling of workflows in an IaaS cloud computing environment. Given the workflow execution deadline, δ, our scheduling objective is to find a workflow schedule that results in the minimum execution cost. In order to achieve this objective, our scheduling algorithm needs to address the following issues: Given a workflow, how many VM instances are needed for its execution? When should these VM instances be provisioned and deprovisioned? How do we find the workflow schedule, sch, that minimizes the monetary cost? When does each of the workflow task start and finish?

In this paper, we propose a path-based scheduling algorithm that decides whether two consecutive tasks in a path of a given workflow should be scheduled to the same VM instance by a holistic consideration of the tasks' execution times, data transfer times between the tasks, and the billing

978-1-7281-2772-9/19/$31.00 ©2019 IEEE
DOI 10.1109/BigDataCongress.2019.00018

cycle of the underlying IaaS cloud platform.

We make the following contributions via the work presented in this paper:

1) We propose a path-based workflow scheduling algorithm that generates workflow schedules of minimum execution cost on AWS EC2 (an IaaS cloud service).
2) The proposed algorithm is implemented and integrated into a real popular workflow system, DATAVIEW.
3) We conducted systematic experiments and the acquired results showed that LPOD outperforms the state-of-the-art algorithms, IC-PCP [16] and SGX-E2C2D [17], by delivering more cost-efficient schedules for big data workflows.

This paper is structured as follows. Section 2 reviews related work regarding workflow scheduling. Section 3 introduces basic definitions to formalize the deadline-constrained workflow scheduling problem. Section 4 describes our scheduling algorithm, LPOD, in detail. The experimental results are presented and discussed in Section 5. Finally, section 6 concludes this paper.

2. Related Work

Workflow scheduling is a well-known complex problem. Heuristic-based and meta-heuristic-based approaches have been studied for the workflow scheduling problem. Popular meta-heuristic-based approaches include particle swarm optimization (PSO) [18] and genetic algorithms (GA) [19]. Several heuristic-based algorithms [20] [21] [22] [23] have been proposed for workflow scheduling in the cloud computing environment. However, existing approaches did not adequately consider the important characteristics of the cloud computing environment.

Saeid et al. [16] proposed a deadline-constrained workflow scheduling algorithm by simply assigning a whole partial critical path to a single VM instance to minimize the data transfer time between consecutive tasks. This algorithm does not explore the possibility of assigning the tasks from one path to multiple VM instances in search for potentially better schedule solutions. Another drawback is that this algorithm does not consider the time of a provisioned VM instance completing the transfer of all the output data to the local storage of the VMs executing the child tasks, before it is deprovioned. This is impractical during the workflow execution life time.

Jyoti and Deo Prakash [24] proposed a just-in-time scheduling strategy that starts with a pre-processing step that combines pipelined tasks into a single task and then assigns such combined mega task to a single VM instance. Different from most other workflow scheduling algorithms in which each task is only visited once, this algorithm needs to visit each task twice (in the pre-processing stage and schedule generation stage). What's more, their method may result in a schedule less cost-efficient than a mapping scheme that assigns pipelined tasks to multiple VM instances.

The algorithm proposed by Xiumin et al. [25], extended from HEFT [26], utilizes a two-stage procedure to minimize execution cost and workflow makespan simultaneously. However, they do not consider a VM instance's booting time and the actual data transfer time between consecutive tasks. This algorithm selects the final scheduling solution from K identified best solutions. However, how to decide the best value for K is not addressed. Meanwhile, evaluating the K scheduling solutions in order to decide the final best one renders the scheduling algorithm itself inefficient.

3. Problem Description

In this section, we introduce a few definitions and formalize the deadline-constrained workflow scheduling problem. The goal is to find a workflow schedule sch for a given workflow W that minimizes the monetary cost of the workflow's execution in the cloud under a given deadline constraint δ. The assumption is that each task is assigned to one single VM instance and each VM instance can only run one single task at a time (no parallelism within one VM instance).

In this model, we consider the following aspects of cloud computing environment and of a given workflow: the capacity of each VM type, the execution time of each workflow task with each VM type, the data transfer rate and price in the cloud, the sizes of data products for a workflow and the booting times of VM instances. We first model the notions of big data workflow and its cloud computing execution environment as follows.

Definition 3.1 (Cloud Computing Environment): A cloud computing environment is modeled as an nine-tuple $C(VMT, VMC, VPrice, VMI, Type, DTR, DPrice, BootDelay)$, where:

- VMT is a set of VM types in the environment, and the kth VM type is denoted as VMT^k.
- $VMC : VMT \rightarrow \mathbb{R}^+$ is the VM capacity function, and $VMC(VMT^k)$ returns the computation speed of VMT^k in terms of Million Instructions Per Second (MIPS). $\mathbb{R}^+$ is the set of all positive real numbers.
- $VPrice : VMT \rightarrow \mathbb{R}^+$ is the VM price function, and $VPrice(VMT^k)$ returns the monetary cost per each billing cycle l for using a VM instance of type VMT^k.
- VMI is a set of VM instances, an individual VM instance is denoted as VMI_m.
- $Type : VMI \rightarrow VMT$ is the VM type function, and $Type(VMI_m)$ returns the VM type of instance VMI_m.
- $DTR : VMI \times VMI \rightarrow \mathbb{R}^+$ is the data transfer rate function. $DTR(VMI_{m1}, VMI_{m2})$ returns the network bandwidth between instances VMI_{m1} and VMI_{m2}.
- $DPrice : VMI \times VMI \rightarrow \mathbb{R}^+$ is the data transfer price function. $DPrice(VMI_{m1}, VMI_{m2})$ returns the

price in dollars per MBytes for transferring data from VMI_{m1} to VMI_{m2}.
- $BootDelay$ is the delay of a VM instance before it becomes available after being provisioned.

Definition 3.2 (Big Data Workflow): A Big data workflow is modeled as a four-tuple $W(T, D, TSize, Dsize)$, where:

- T is a set of tasks.
- $D \subseteq T \times T$ is a set of data dependency edges between tasks. We use $D_{i,j}$ to denote the data product that is produced by t_i and consumed by t_j.
- $TSize : T \rightarrow \mathbb{R}^+$ is the task size function, and $TSize(t_i)$ returns the size of task t_i in terms of million instructions.
- $DSize : D \rightarrow \mathbb{R}^+$ is the data product size function, and $DSize(D_{i,j})$ returns the size of data product $D_{i,j}$ in terms of MBytes.

For simplicity of presentation, we introduce two distinguished tasks, t_{entry} and t_{exit} to each workflow W such that t_{entry} is connected to each entry task in W (tasks that have no parents), and t_{exit} is connected to each exit task in W (tasks that have no children). Then, we introduce a workflow graph G to model the performance properties of workflow W under a particular cloud computing environment C as follows.

Definition 3.3 (Workflow Graph): Given a cloud computing environment C and workflow W, a workflow graph is formalized as a five-tuple $G(T, D, ET, DTT, DTC)$, where

- $T = W.T \bigcup \{t_{entry}, t_{exit}\}$. Here $W.T$ denotes the task set of W.
- $D = W.D \bigcup D_{entry} \bigcup D_{exit}$. $D_{entry} = \{D_{entry,i} \mid t_i \in entrytasks(W)\}$, and $D_{exit} = \{D_{j,exit} \mid t_j \in exittasks(W)\}$.
- $ET : T \times VMT \rightarrow \mathbb{R}^+$ is the task execution time function, and $ET(t_i, VMT^k)$ returns the execution time of t_i on a VM instance of type VMT^k, which is defined as:

$$ET(t_i, VMT^k) = \frac{TSize(t_i)}{VMC(VMT^k)} \tag{1}$$

- $\overline{ET} : T \rightarrow \mathbb{R}^+$ is the average task execution time function, which is defined as:

$$\overline{ET}(t_i) = \frac{\sum_{k=1}^{|VMT|} ET(t_i, VMT^k)}{|VMT|} \tag{2}$$

- $DTT : D \times VMI \times VMI \rightarrow \mathbb{R}^+$ is the data transfer time function, and $DTT(D_{i,j}, VMI_{m1}, VMI_{m2})$ returns the time for transferring $D_{i,j}$ from VMI_{m1} to VMI_{m2}, which is defined as:

$$DTT(D_{i,j}, VMI_{m1}, VMI_{m2}) = \begin{cases} 0, & if\ m1{=}m2 \\ \frac{DSize(D_{i,j})}{DTR(VMI_{m1}, VMI_{m2})} & otherwise \end{cases} \tag{3}$$

- $\overline{DTT} : D \rightarrow \mathbb{R}^+$ is the average data transfer time function, and $\overline{DTT}(D_{i,j})$ returns the average transfer time of data from t_i to t_j, which is defined as:

$$\overline{DTT}(D_{i,j}) = \frac{DSize(D_{i,j})}{\overline{DTR}} \tag{4}$$

where $\overline{DTR}$, the average data transfer rate among C, is defined by:

$$\overline{DTR} = \frac{\sum_{v_1 \in VMI, v_2 \in VMI, v_1 \neq v_2} DTR(v_1, v_2)}{|VMI| \times (|VMI| - 1)/2} \tag{5}$$

- $DTC : D \times VMI \times VMI \rightarrow \mathbb{R}^+$ is the data transfer monetary cost function, and $DTC(D_{i,j}, VMI_{m1}, VMI_{m2})$ returns the monetary cost of transferring $D_{i,j}$ from task t_i running in VM instance VMI_{m1} to task t_j running in VM instance VMI_{m2}. It is defined as:

$$DSize(D_{i,j}) \times DPrice(VMI_{m1}, VMI_{m2}) \tag{6}$$

Next, we model the notion of a workflow schedule, which will be produced by a particular workflow scheduler (a.k.a. workflow planner) in a workflow system, such as the one proposed in the next section.

Definition 3.4 (Workflow Schedule): Given a cloud computing environment C and a workflow W, a workflow schedule is modeled as a seven-tuple $sch(T, VMI, M, AST, AFT, Prov, Deprov)$, where:

- T is the set of tasks in workflow W, thus $T = W.T$.
- VMI is the set of VM instances in C, thus $VMI = C.VMI$.
- $M : T \rightarrow VMI$ is a task assignment function, and $M(t_i)$ is the VM instance that task t_i is assigned to.
- $AST : T \rightarrow \mathbb{R}^+$: is an actual start time function, and $AST(t_i)$ returns the actual start time of task t_i on VM instance $M(t_i)$.
- $AFT : T \rightarrow \mathbb{R}^+$ is an actual finish time function, and $AFT(t_i)$ returns the actual finish time of task t_i on VM instance $M(t_i)$.
- $Prov : VMI \rightarrow \mathbb{R}^+$ is a VM instance provisioning time function, and $Prov(VMI_m)$ returns the time that VM instance VMI_m is to be provisioned.
- $Deprov : VMI \rightarrow \mathbb{R}^+$ is a VM instance deprovisioning time function, and $Deprov(VMI_m)$ returns the time that VM instance VMI_m is to be deprovisioned.

The total execution time and monetary cost of a schedule sch are defined by workflow makespan and workflow cost, respectively, as follows.

Definition 3.5 (Workflow makespan *WMS*): Given a workflow schedule sch, the makespan of workflow sch is defined as:

$$WMS(sch) = sch.AFT(sch.t_{exit}) \tag{7}$$

Definition 3.6 (Workflow cost *WC*): Given a workflow schedule sch, the total execution monetary cost of sch in cloud computing environment C is defined as:

$$WC(sch) = \sum_{v \in VMI} VPrice(Type(v)) * \lceil \frac{Deprov(v) - Prov(v)}{l} \rceil \quad (8)$$

The above definition assumes a zero data movement monetary cost among VM instances, which is the case for cloud services such as Amazon EC2. The above definition also rounds up VM instance usage to its nearest full billing cycle. Finally, our scheduling problem can be formulated as follows.

Definition 3.7 (Deadline-constrained Workflow Scheduling Problem): Given a workflow W, an IaaS computing environment C, and a deadline δ, the deadline-constrained workflow scheduling problem is to find the optimal workflow schedule sch_{opt} that minimizes the monetary cost of running workflow W within deadline δ:

$$\begin{aligned} sch_{opt} &= \underset{sch}{\operatorname{argmin}} WC(sch) \\ subject\ to\ WMS(sch) &\leqslant \delta \end{aligned} \quad (9)$$

4. The Proposed Workflow Scheduling Algorithm

This scheduling algorithm attempts to minimize the total execution cost by minimizing the cost of each partial path. Our approach to the deadline constrained scheduling problem consists of two phases: the partial path construction phase and the resource selection phase. In the first phase, a workflow graph G is partitioned into several partial paths. Then, in the second phase, a path based optimization algorithm is employed to find a cost-minimized assignment for each path.

This section first introduces fundamental definitions utilized in the proposed algorithm. Then, the algorithm is introduced followed by a time complexity analysis of the algorithm. Finally, an illustrative example demonstrates how the algorithm works.

A. Basic Definitions

The priority rank of a task is introduced to measure the length of the partial path from a task to the exit node t_{exit}. This rank will used by the proposed algorithm to divide a workflow graph G into partial paths.

Definition 4.1 (Priority Rank Pri): Given a workflow graph G, the priority rank of a task t_i is defined as follows:

$$Pri(t_i) = \begin{cases} \overline{ET}(t_i), & if\ t_i = t_{exit} \\ \overline{ET}(t_i) + \max\limits_{t_j \in (t_i)^\bullet} (\overline{DTT}(D_{i,j}) + Pri(t_j)), & otherwise \end{cases} \quad (10)$$

where $(t_i)^\bullet$ denotes the child tasks of t_i in G.

The minimum execution time of a task is defined as the execution time of the task on an instance of the fastest VM type in C.

Definition 4.2 (Minimum Execution Time MET): Given a workflow graph G and a cloud computing environment C, the minimum execution time of a task t_i is defined as:

$$MET(t_i) = \min_{k=1}^{|VMT|} ET(t_i, VMT^k) \quad (11)$$

The estimated execution time of a task is the task's running time on an assigned instance of a VM type in C.

Definition 4.3 (Estimated Execution Time ET^*): Given a workflow graph G, a cloud computing environment C and a VM assignment function M, the estimated execution time of a task t_i is defined as:

$$ET^*(t_i) = \begin{cases} MET(t_i), & if\ M(t_i) = null \\ ET(t_i, Type(M(t_i))), & otherwise \end{cases} \quad (12)$$

B. The Proposed Algorithm LPOD

Algorithm 1 represents the main steps of the LPOD algorithm for scheduling a workflow. The algorithm calculates the priority rank for each task and sorts all tasks in the list $tlist$ in decreasing order of their priority ranks (line 2). Line 3 computes the earliest start time (EST) of each task. For task t_{entry}, the EST is set to 0, and the ESTs for other tasks are calculated as follows:

$$EST(t_i) \leftarrow \max_{t_p \in {}^\bullet(t_i)} (EST(t_p) + ET^*(t_p) + \overline{DTT}(D_{p,i})) \quad (13)$$

where ${}^\bullet(t_i)$ denotes the parent tasks of t_i in G. Line 4 computes the latest finish time (LFT) of each task with $LFT(t_{exit})$ set to δ. The LFTs of other tasks are computed by:

$$LFT(t_i) \leftarrow \min_{t_c \in (t_i)^\bullet} (LFT(t_c) - ET^*(t_c) - \overline{DTT}(D_{i,c})) \quad (14)$$

After that, the (main) algorithm calls Algorithm 2 to construct all the partial paths stored in the list, PP. Lastly, Algorithm 3 is called to assign each partial path to a set of appropriate VM instances assuring minimum cost for the tasks on each path, resulting in a workflow schedule sch.

1) Partial Paths Construction: Algorithm 2 constructs partial paths based on the priority rank of each task computed by Eq. (10), starting from task t_{exit} recursively (backward). A temporary list, cp, is created to hold the tasks in a partial path (line 3). The task with the highest priority rank is first selected from $tlist$, added into cp and then removed from $tlist$ (lines $4-5$). Afterwards, via the while loop, the algorithm constructs each partial path with a depth-first search (lines $6-10$). Once a path is found, it is added into PP (line 11).

2) Paths Assignment: Algorithm 3 takes all the paths identified by Algorithm 2 as input, and sequentially schedules all the paths. For each path $P_i = [t_1, \ldots, t_n]$, we use a two-phase procedure to schedule the tasks of P_i to a set of VM instances. The first phase is to allocate a prefix of P_i, $[t_1, \ldots, t_{lj}]$, to existing VM instances, with each task in the prefix possibly assigned to either the same or different VM instances (lines $1-16$). Algorithm 4 is called in the second phase (to be elaborated in the next section) to assign $[t_{lj+1}, \ldots, t_n]$ to new VM instances (line 17). Each assigned VM instance, VMI_i, has two attributes in the scheduling algorithm: the available time point $vavat$ and the leftover time in the current billing cycle $vleft$, which are used to identify the free computing resource for an unassigned task during the first phase. Three conditions will be checked (line 4) to decide whether a task can be assigned to an existing VM instance with leftover time available:

- The EST of a task should be after a VM instances available time point.
- The leftover time of the VM instance should be long enough to finish the task.
- The task should be finished before its latest finish time.

The algorithm starts from the first task in P_i. Once a task is assigned to an existing VM instance, the workflow schedule sch and the two attributes of the VM instance are updated (line 7). The length of the prefix, lj, is also updated (line 8). Then the next task is tried to be assigned in the same logic. This process stops until a task in P_i cannot be assigned to an existing VM instance (lines $13-15$). Once all tasks in a path P_i are scheduled sequentially, the actual start time and actual finish time can be decided. Then the EST of t_i is updated with the actual start time of t_i and the LFT of t_i is updated with the actual finish time (lines $19-20$). The algorithm then updates the EST of each unassigned succeeding task t_c of t_i (line 21) using Eq. (13) and updates LFT of each unassigned preceding task t_p of t_i (line 22) using Eq. (14).

3) Single Path Assignment: Algorithm 4 takes a path, $P = [t_1, \cdots, t_n]$, as input and schedules the tasks of the path on a set of VM instances with the objective of minimizing the total execution cost of the whole path by using a dynamic programming strategy to assign a set of possible VM instances (or only one instance, which is a special case) to it. In contrast to ICPCP and SGX-E2C2D, which assign a whole path to one single VM instance, this algorithm explores the possibility to assign a path to multiple VM instances, resulting an optimized VM assignment solution for a single path. The possible VM instance assignment for the current task t_i is based on its direct parent task t_{i-1}'s possible VM instance assignment by considering the leftover time $tleft$ for current billing cycle of the VM instance. Specifically, this strategy is implemented based on a matrix, represented as a table TB. Each solution is represented as a tuple in TB. It is based on the recurrence relationship in which a solution to a partial path $[t_1, \cdots, t_i]$ is based on a solution to its prefix $[t_1, \cdots, t_{i-1}]$. While t_1 is always assigned to a new VM instance (lines $2-7$), the main point is to decide whether a new VM instance needs to be assigned to t_i. This depends on the temporal assignment of $[t_1, \cdots, t_{i-1}]$ (lines $8-18$). More specifically, TB stores a list of tuples $(id, t, tst, tft, acccost, tleft, idref, maxedge, VMtype)$ computed for each task t_i in P, resulting in one or multiple tuples for t_i. Each tuple of t_1 is calculated by tentatively assigning an instance of each VM type within its LFT and inserted into TB (lines $3-7$). Except for t_1, each tuple of t_i is computed based on a tuple of t_{i-1}. Let pt be the tuple of t_{i-1} based on which a new tuple nt of t_i is being computed. The attributes of nt in table TB are:

- id is a unique identifier for each tuple, which is automatically generated by the function genID();
- t stores the information of task t_i in path P.
- tst stores the temporary actual start time of task t_i. We set tst to $BootDelay$ if $nt.t = t_1$; otherwise tst is computed by:

$$\begin{cases} \max(EST(t_i), pt.tft), \ if\ nt.VMtype = pt.VMtype \\ \max(EST(t_i), pt.tft) + \overline{DTT}(D_{i-1,i}), \quad otherwise \end{cases} \tag{15}$$

- tft stores the temporary actual finish time of task t_i, which is computed by:

$$nt.tft \leftarrow nt.tst + ET(t_i, VMtype(t_i)) \tag{16}$$

- $acccost$ stores the temporary accumulated cost from the first task in P to task t_i. We differentiate three cases: If $nt.t = t_1$ (case 1), $acccost$ is computed by:

$$nt.acccost \leftarrow \lceil \frac{nt.tft}{l} \rceil \times VPrice(VMtype(t_i)) \tag{17}$$

Otherwise, if $nt.VMtype = pt.VMtype$ (case 2), then $aacost$ is computed by:

$$pt.acccost + \begin{cases} 0, & if\ nt.tft \leq pt.tft + pt.tleft \\ \lceil \frac{nt.tft - pt.tleft}{l} \rceil \times VPrice(VMtype(t_i)), & otherwise \end{cases} \tag{18}$$

Otherwise, if $nt.VMtype \neq pt.VMtype$ (case 3), then $aacost$ is computed by

$$nt.acccost \leftarrow pt.aaccost + \lceil \frac{ET(t_i, VMtype(t_i))}{l} \rceil \times VPrice(VMtype(t_i)) \tag{19}$$

- $tleft$ stores the leftover time in the current billing cycle of the assigned VM instance. We have three cases. If $nt.t = t_1$ (case 1), then $tleft$ is calculated by:

$$nt.tleft \leftarrow \lceil \frac{nt.tft}{l} \rceil \times l - nt.tft \tag{20}$$

Otherwise, if $nt.VMtype = pt.VMtype$ (case 2), then $tleft$ is computed by:

$$\begin{cases} pt.tft + pt.tleft - nt.tft, \\ \qquad if\ nt.tft \leq pt.tft + pt.tleft \\ \lceil \frac{nt.tft - pt.tleft}{l} \rceil \times l - (nt.tft - pt.tleft),\ otherwise \end{cases} \tag{21}$$

Otherwise, if $nt.VMtype \neq pt.VMtype$ (case 3), then $tleft$ is calculated by:

$$nt.tleft \leftarrow \lceil \frac{nt.tft}{l} \rceil \times l - nt.tft \tag{22}$$

- $idref \leftarrow pt.id$, that is, it stores the id of the tuple of t_{i-1}, the direct parent of task t_i in the partial path P, based on which the current tuple of t_i is calculated.
- $maxedge$ stores the largest data transfer time from t_i to its direct children, which is computed by:

$$nt.mexedge \leftarrow \max_{t_c \in (t_i)^\bullet} \overline{DTT}(D_{i,c}) \tag{23}$$

- $VMtype$ stores the VM type that is assigned to t_i temporarily in the algorithm.

If the child task is assigned to a different VM, the current VM can only be deprovisioned after all the output data has been transferred to the VM running the child task. It is not necessary for the VM executing the child task to be active when the input data files are being transferred to the corresponding storage volume with the isolated lifetime of Elastic Block Store (EBS) [27]. In this case, only the data transfer time from one VM instance executing a task to other VM instances running the child tasks is included in the total billing cycles. Let $pret$ be the tuple containing the minimal accumulated cost to execute t_n in the path P (line 19). The workflow schedule sch is updated based on the attributes in $pret$ (line 20). Then the backtracking strategy is used based on TB to decide the tuple of t_n's parent task denoted by ct. If the VM types in $pret$ and ct are of the same type, the provisioning time of VM instance in $pret$ needs to be updated (lines $24-26$). When the VM types in $pret$ and ct are different, the provisioning time of the two VM types are updated sequentially (lines $27-41$) by considering the tuple pt calculated on ct with two cases (lines $31-36$). In this way, the optimal VM instance that is assigned to each task is determined. A special case is that if there is one task in P, then the provisioning time of the VM instance needs to be updated by the actual start time minus the VM delay time (lines $44-46$).

C. Time complexity

A workflow graph G with n tasks and e edges is inputted into the LPOD algorithm. The maximum number of dependencies in G is $\frac{(n-1)(n-2)}{2}$. In order to calculate all the partial critical paths and initialize the EST and LFT for each task, all the nodes and edges are visited once and the complexity is $\mathcal{O}(n+e)$.

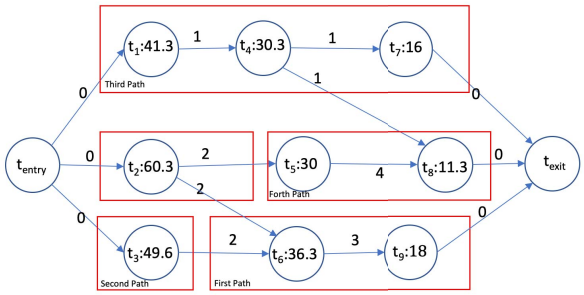


Figure 1: A sample workflow graph with priority ranks on nodes and average transfer times on edges.

Another part is the procedure $AssignPath$, which assigns a prefix of a partial path to those VM instances with sufficient leftover time. Suppose that the maximum number of existing instances is m, the complexity of the assignment process is expressed as $\mathcal{O}(m \times n)$. The last part is to assign the suffix of the partial path to a set of feasible VM instances with the minimum cost. It uses the dynamic programming strategy by assigning each VM type for each task within the LFT, and the maximum number of task in a path is n, so the complexity of this process is $\mathcal{O}(n \times |VMT|^2 * \delta)$. Overall, the complexity for the LPOD is $\mathcal{O}(n+e) + \mathcal{O}(m \times n) + \mathcal{O}(n \times |VMT|^2 * \delta) \simeq \mathcal{O}(n^2)$.

Algorithm 1: Workflow Scheduling Algorithm

Input : A workflow graph $G(T, E)$ and deadline δ
InOut: A workflow schedule sch
1 $sch \leftarrow \emptyset$;
2 $tlist \leftarrow$ the list of tasks in G in descending order of priority ranks;
3 $EST(t_{entry}) \leftarrow 0$ and compute EST for other tasks;
4 $LFT(t_{exit}) \leftarrow \delta$ and compute LFT for other tasks ;
5 $PP \leftarrow$ FindAllPcps($tlist$);
6 PathsAssignment(PP,sch);

D. An illustrative example

A sample workflow is used to illustrate how the algorithm works for different task execution times in three types of VM computing services. We assume three types of VMT in this case (more types may be used with a specific cloud provider). The billing cycle of current computing services is set to 10, the cost is 5, 2 and 1 for VMT^1, VMT^2 and VMT^3, respectively. The delay of a VM instance is set to 1, and the deadline for the workflow is 50.

When the local path optimized scheduling algorithm, i.e., $Algorithm1$ is called, for the sample workflow of Fig. 1, the EST and the LFT for each task are calculated at the initial step in Table I. All the partial

Algorithm 2: FindAllPcps

Input : A sorted task list $tlist = [t_1, \cdots, t_n]$
Output: A path list PP

$PP \leftarrow \emptyset$;
foreach t_i *in* $tlist$ **do**
  $cp \leftarrow \emptyset$;
  $cp.add(t_i)$;
  $tlist.remove(t_i)$;
  while t_i *has children in* $tlist$ **do**
    $t_i \leftarrow t_i's$ first child in $tlist$;
    $cp.add(t_i)$;
    $tlist.remove(t_i)$;
  end
  $PP.add(cp)$;
end
Return PP ;

Algorithm 3: PathsAssignment

Input : A path list $PP = [P_1, \cdots, P_m]$
InOut : A workflow schedule sch

foreach $P_i = [t_1, \cdots, t_n]$ *in* PP **do**
  $lj \leftarrow 0$;
  foreach t_j *in* $[t_1, \cdots, t_n]$ **do**
    $found \leftarrow false$;
    foreach VMi *in* VMI **do**
      if $VMi.vavat \leq EST(t_j)$ *and* $VMi.vleft \geq ET(t_j, Type(VMi))$ *and* $LFT(t_j) \geq EST(t_j) + ET(t_j, Type(VMi))$ **then**
        $sch.T.insert(t_j)$;
        $sch.AST.insert(t_j, EST(t_j))$;
        $aft \leftarrow EST(t_j) + ET(t_j, Type(VMi))$
        $sch.AFT.insert(t_j, aft)$;
        $sch.M.insert(t_j, VMi)$;
        $VMi.vavat \leftarrow EST(t_j) + ET(t_j, Type(VMi))$;
        $VMi.vleft \leftarrow VMi.vleft - ET(t_j, Type(VMi))$;
        $lj \leftarrow j$;
        $found \leftarrow true$;
        break;
      end
    end
    if $found == false$ **then**
      break;
    end
  end
  AssignPath($[t_{lj+1}, \cdots, t_n]$, sch) ;
  foreach $t_j \in P_i$ **do**
    $EST(t_j) \leftarrow sch.AST(t_j)$;
    $LFT(t_j) \leftarrow sch.AFT(t_j)$;
    update EST for all successors of t_j according to Eq. (13);
    update LFT for all predecessors of t_j accoring to Eq. (14);
  end
end

paths ($t_2 \rightarrow t_6 \rightarrow t_9$; t_3; $t1 \rightarrow t_4 \rightarrow t_7$ and $t_5 \rightarrow t_8$) are constructed based on the priority rank of each task. Next, the algorithm $PathsAssignment$ is called to assign VM instances to each partial path.

The $PathsAssignment$ algorithm starts with the first path of the paths list, which is $t_2 \rightarrow t_6 \rightarrow t_9$. For this path, a table is created to calculate the lowest accumulated cost at the last task, from which a path assignment can be traced back. The algorithm assigns each task in this path to VMT^2, VMT^1 and VMT^1. The next step updates the ESTs of

Algorithm 4: AssignPath

Input : A path $P = [t_1, \cdots, t_n]$
InOut : A path schedule sch

$TB \leftarrow \emptyset$;
foreach VMT^k *in* VMT **do**
  create a new tuple nt; $nt.id \leftarrow genID()$; $nt.t \leftarrow t_1$; $nt.tst \leftarrow BootDelay$; $nt.tft$ is calculated by Eq. (16); $nt.acccost$ is calculaed by Eq. (17); $nt.tleft$ is calculated by Eq. (20); $nt.idref \leftarrow NULL$; $nt.maxedge$ is calculated by Eq. (23); $nt.VMtype \leftarrow VMT^k$;
  if $nt.tft \leq LFT(t_1)$ **then**
    $TB.insert(nt)$
  end
end
foreach t_i *in* $[t_2, \cdots, t_n]$ **do**
  foreach *tuple* pt *of* t_{i-1} *in TB* **do**
    foreach VMT^k *in* VMT **do**
      create a new tuple nt; $nt.id \leftarrow genID()$; $nt.t \leftarrow t_i$; $nt.tst$ is calculated by Eq. (15); $nt.tft$ is calculated by from Eq. (16);
      if $nt.tft \leq LFT(t_i)$ **then**
        $nt.acccost$ is calculaed by Eq. (18) or Eq. (19); $nt.tleft$ is calculated by Eq. (21) or Eq. (22); $nt.idref \leftarrow pt.id$; $nt.maxedge$ is calculated by Eq. (23); $nt.VMtype \leftarrow VMT^k$;
        $TB.insert(nt)$
      end
    end
  end
end
$pret \leftarrow$ select $*$ from TB where $t = t_n$ order by $acccost$ limit 1;
$preindex \leftarrow pret.idref$; $sch.T.insert(t_n)$;
$sch.AST.insert(t_n, pret.tst)$; $sch.AFT.insert(t_n, pret.tft)$;
$preVM \leftarrow$ initialize a new instance from $pret.VMtype$;
$preVM.vavat \leftarrow pret.tft$; $preVM.vleft \leftarrow pret.tleft$;
$sch.VMI.insert(preVM)$; $sch.M.insert(t_n, preVM)$;
$sch.Prov.insert(preVM, pret.tst)$;
$sch.Deprov.insert(preVM, (pret.tft + pret.maxedge))$;
foreach t_i *in* $[t_{n-1}, \cdots t_1]$ **do**
  $ct \leftarrow$ select $*$ from TB where $id = pret.idref$;
  $sch.AST.insert(t_i, ct.tst)$; $sch.AFT.insert(t_i, ct.tft)$; $sch.T.insert(t_i)$;
  if $ct.VMtype == pret.VMtype$ **then**
    $sch.M.insert(t_i, preVM)$;
    $sch.Prov.update(preVM, ct.tst)$;
  end
  else
    $cVM \leftarrow$ initialize a new instance from $ct.VMtype$;
    $cVM.vavat \leftarrow ct.tft$; $cVM.vleft \leftarrow ct.tleft$;
    $sch.VMI.insert(cVM)$; $preVM \leftarrow cVM$;
    $sch.M.insert(t_i, cVM)$;
    $sch.Deprov.insert(cVM, ct.tft + ct.maxedge)$;
    $sch.Prov.insert(cVM, ct.tst)$;
    if $t_i \neq t_1$ **then**
      $pt \leftarrow$ select $*$ from TB where $id = ct.idref$;
      if $pt.VMtype == ct.VMtype$ **then**
        $sch.Prov.update(cVM, pt.tst)$;
      end
      else
        $sch.Prov.update(cVM, pt.tst - BootDelay)$;
      end
    end
    else
      $sch.Prov.update(cVM, pret.tst - BootDelay)$;
    end
  end
  $pret \leftarrow ct$;
end
if n==1 **then**
  $sch.Prov.update(preVM, pret.tst - BootDelay)$;
end

Table I: Changes of EST and LFT of each task.

Tasks		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
Initial	EST	0	0	0	5	12	12	14	22	23
	LFT	30	27	27	39	40	37	50	50	50
t_2, t_6, t_9	EST	$1^\dagger$	$1^\dagger$	$1^\dagger$	$6^\dagger$	$27^\dagger$	$27^\dagger$	$15^\dagger$	$37^\dagger$	$35^\dagger$
	LFT	30	$25^\dagger$	$25^\dagger$	39	40	$35^\dagger$	50	50	$45^\dagger$
t_3	EST	1	1	1	6	27	27	15	37	35
	LFT	30	25	$19^\dagger$	39	40	35	50	50	45
t_1, t_4, t_7	EST	1	1	1	$18^\dagger$	27	27	$30^\dagger$	37	35
	LFT	$17^\dagger$	25	19	$30^\dagger$	40	35	$46^\dagger$	50	45
Final allocation		VMT^1	VMT^2	VMT^3	VMT^2	VMT^1	VMT^1	VMT^2	VMT^1	VMT^1

Table II: Execution time matrix.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
VMT^1	4	10	6	8	6	8	10	6	10
VMT^2	10	24	10	12	16	16	16	12	16
VMT^3	16	32	18	20	22	22	22	16	28

Table III: Execution cost distribution for various VM instances.

Type	[Start, End]	Internal cycle	Cost	Scheduled task
VMI_1^1	[26,45]	2	10	$\{t_6, t_9\}$
VMI_2^1	[26,43]	2	10	$\{t_5, t_8\}$
VMI_1^2	[0,27]	3	6	$\{t_2\}$
VMI_2^2	[17,46]	3	6	$\{t_4, t_7\}$
VMI_1^3	[0,21]	3	3	$\{t_3\}$
VMI_1^3	[0,18]	2	2	$\{t_1\}$

all unassigned successors and the LFTs of unassigned predecessors. The changes of EST and LFT are shown in the third row of table I marked with (†). Instead of allocating two more rows, the actual start time of a task is represented by the EST and the actual finish time of a task will be indicated by the LFT after it is scheduled.

The second partial path t_3 is assigned. The algorithm attempts to assign it to free computing resources of the existing provisioned VM instances with remaining time period for the current billing cycle. In this case, there is no available computing resource, the cheapest VM type VMT^3 is selected to execute t_3. The LFT of t_3 is updated in I.

The third partial path, $t1 \rightarrow t_4 \rightarrow t_7$, is assigned to the cheapest VM types VMT^3, VMT^2, and VMT^2 for each task, followed by updating the ESTs of their unassigned successor t_8.

At last, the final partial path $t_5 \rightarrow t_8$ is assigned. The available VM type for both tasks is VMT^1. An instance is launched to execute these two tasks sequentially. The overall cost of the sample example is 37 while IC-PCP and SGX-E2C2D both charge 42 with the data transfer time and the delay time of VM instances considered.

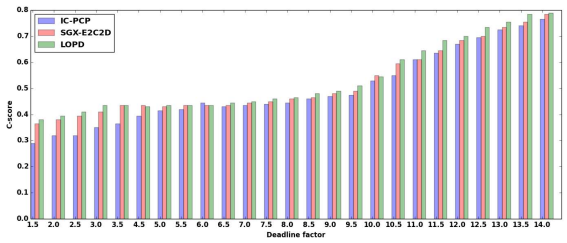
Table III shows the provisioning time, deprovisioning time, internal cycle number, cost and the scheduled tasks of each VM instance.

5. Performance Evaluation

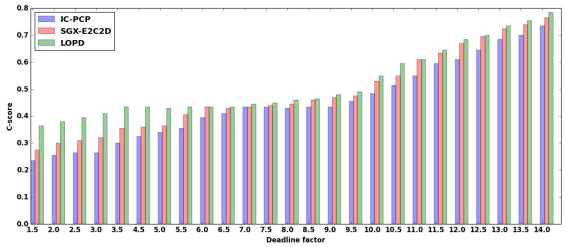
Montage and LIGO workflows are widely used in the workflow scheduling community. We generated several workflows based on the two workflow structures in the DATAVIEW system to verify the proposed algorithm performance. All the workflows can be retrieved from [28], in which tasks are simulated by the bubble sorting algorithm with different sizes or different numbers of matrix multiplication to generate different task sizes. To compare the performance of LPOD, IC-PCP, and SGX-E2C2D on an IaaS cloud environment, we used three types of VMs in Amazon EC2: t2.micro, t2.large and t2.xlarge to execute these workflow samples, in which the VM computing capacities are in an increasing order. The data transfer time from one task to another running two different VM instances is calculated based on an average bandwidth value of 20MBps between VMs in Amazon EC2 [29]. In order to handle the fluctuating performance of executing each task, an evaluation metric C score is used to assess the performance of a workflow schedule generated by each algorithm, which is slightly different from the Fitness Score in [30]. The C score first assigns 0.5 to each schedule and compares the real makespan and deadline. If the real makespan is smaller than the deadline, a reward between 0.0 to 0.5 is added to C, otherwise a penalty between 0.0 to 0.5 is subtracted from C. The definition of C is below:

$$C = \begin{cases} 0.5 + 0.5 * \frac{(maxcost - cost)}{maxcost}, & if\ makespan \leq \delta \\ 0.5 - 0.5 * \frac{(makespan - \delta)}{(maxmakespan - \delta)}, & otherwise \end{cases} \tag{24}$$

The $maxcost$ is the monetary cost by executing each workflow task on a distinct instance of the fastest computa-

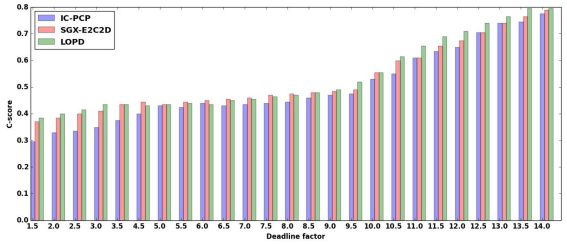


(a) 10 seconds billing cycle for Montage.

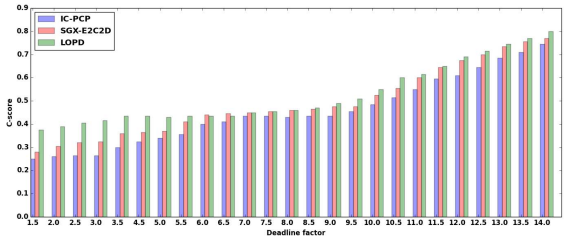


(b) 60 seconds billing cycle for Montage.

Figure 2: C-score for Montage workflow.



(a) 10 seconds billing cycle for LIGO.



(b) 60 seconds billing cycle for LIGO.

Figure 3: C-score for LIGO workflow.

tion service including all transfer times. The $maxmakespan$ is retrieved by executing all tasks in the slowest computation service.

Before experiments, we collected the average task execution time in 3 different VM types by running each task 10 times with three different VM types in the Amazon EC2 environment. One important parameter is the billing cycle, which affects the workflow schedule and the final cost. Most current commercial clouds, such as Amazon, charge users by hours. The billing cycle is scaled for this experiment. A long billing cycle $60s$ and a short billing cycle $10s$ were applied in the experiment. The deadline is another vital factor affecting the final schedule. We denoted the actual finish time of the first partial path running on the fastest computation service as M_f and set the deadline following rules specified below:

$$Deadline\ \delta = (1 + \lambda) \times M_f \tag{25}$$

Where, λ ranges from 1.0 to 15.0 with a step length of 0.5. There are 30 tasks in the Montage workflow and 25 tasks in the LIGO workflow.

Initially, all three algorithms failed to meet the deadline when $\lambda < 9$ due to the unpredictable transfer time of the Amazon VMs, code execution time and system communication time. Fig. 2a, Fig. 2b, Fig. 3a and Fig. 3b shows that,in a total of 25 cases, our proposed algorithm beats other algorithms in 20, 24 cases of Montage and 19, 23 cases of LIGO workflows for the 10 seconds and 60 seconds billing cycles, respectively. The improvement of C-score is larger when the deadline factor is smaller (less than 5.0).

6. Conclusions and Future Work

In this paper, we proposed a new workflow scheduling algorithm, LPOD, that successfully finds an optimal schedule for each partial path from a workflow graph G. This algorithm minimizes the total execution cost of a workflow while meeting a user-defined deadline.

The experimental results demonstrate the performance advantage of LPOD over other state-of-the-art algorithms. The main challenge of implementation is to have the workflow executor follow the optimized schedule, which is the focus of our future study.

Acknowledgement

This work is partially supported by National Science Foundation under grants CNS-1747095 and OAC-1738929.

References

[1] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, and B. Berriman, "Experiences using cloud computing for a scientific workflow application," in *Proceedings of the 2nd international workshop on Scientific cloud computing*. ACM, 2011, pp. 15–24.

[2] M. Abouelhoda, S. A. Issa, and M. Ghanem, "Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support," *BMC bioinformatics*, vol. 13, no. 1, p. 77, 2012.

[3] V. C. Emeakaroha, M. Maurer, P. Stern, P. P. Łabaj, I. Brandic, and D. P. Kreil, "Managing and optimizing bioinformatics workflows for data analysis in clouds," *Journal of grid computing*, vol. 11, no. 3, pp. 407–428, 2013.

[4] M. Atkinson, C. S. Liew, M. Galea, P. Martin, A. Krause, A. Mouat, O. Corcho, and D. Snelling, "Data-intensive architecture for scientific knowledge discovery," *Distributed and Parallel Databases*, vol. 30, no. 5-6, pp. 307–324, 2012.

[5] F. Bhuyan, S. Lu, I. Ahmed, and J. Zhang, "Predicting efficacy of therapeutic services for autism spectrum disorder using scientific workflows," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 3847–3856.

[6] I. Ahmed, S. Lu, C. Bai, and F. A. Bhuyan, "Diagnosis recommendation using machine learning scientific workflows," in *2018 IEEE International Congress on Big Data (BigData Congress)*. IEEE, 2018, pp. 82–90.

[7] Z. Li, C. Yang, B. Jin, M. Yu, K. Liu, M. Sun, and M. Zhan, "Enabling big geoscience data analytics with a cloud-based, MapReduce-enabled and service-oriented workflow framework," *PloS one*, vol. 10, no. 3, p. e0116781, 2015.

[8] A. Kashlev, S. Lu, and A. Mohan, "Big data workflows: a reference architecture and the DATAVIEW system," *Services Transactions on Big Data (STBD)*, vol. 4, no. 1, pp. 1–19, 2017.

[9] J. Liu, E. Pacitti, P. Valduriez, D. De Oliveira, and M. Mattoso, "Multi-objective scheduling of scientific workflows in multisite clouds," *Future Generation Computer Systems*, vol. 63, pp. 76–95, 2016.

[10] H. Arabnejad and J. G. Barbosa, "Multi-QoS constrained and profit-aware scheduling approach for concurrent workflows on heterogeneous systems," *Future Generation Computer Systems*, vol. 68, pp. 211–221, 2017.

[11] M. R. Garey and D. S. Johnson, *Computers and intractability*. wh freeman New York, 2002, vol. 29.

[12] M. Ebrahimi, A. Mohan, and S. Lu, "Scheduling big data workflows in the cloud under deadline constraints," in *2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*. IEEE, 2018, pp. 33–40.

[13] A. Mohan, M. Ebrahimi, S. Lu, and A. Kotov, "Scheduling big data workflows in the cloud under budget constraints," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 2775–2784.

[14] A. Verma and S. Kaushal, "Deadline constraint heuristic-based genetic algorithm for workflow scheduling in cloud," *International Journal of Grid and Utility Computing*, vol. 5, no. 2, pp. 96–106, 2014.

[15] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 67–78.

[16] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[17] I. Ahmed, S. Mofrad, S. Lu, C. Bai, F. Zhang, D. Che, and F. A. Bhuyan, "SGX-E2C2D: A Big Data Workflow Scheduling Algorithm for Confidential Cloud Computing," University of Wayne State, Department of Computer Science, Tech. Rep., 03 2019.

[18] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *2010 24th IEEE international conference on advanced information networking and applications*. IEEE, 2010, pp. 400–407.

[19] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming*, vol. 14, no. 3-4, pp. 217–230, 2006.

[20] W.-N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 39, no. 1, pp. 29–43, 2009.

[21] A. H. Kashan, "League championship algorithm: a new algorithm for numerical function optimization," in *2009 International Conference of Soft Computing and Pattern Recognition*. IEEE, 2009, pp. 43–48.

[22] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature inspired cooperative strategies for optimization (NICSO 2010)*. Springer, 2010, pp. 65–74.

[23] A. G. Delavar and Y. Aryan, "Hsga: a hybrid heuristic algorithm for workflow scheduling in cloud systems," *Cluster computing*, vol. 17, no. 1, pp. 129–137, 2014.

[24] J. Sahni and D. P. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 2–18, 2018.

[25] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, and S. Hu, "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT," *Future Generation Computer Systems*, vol. 93, pp. 278–289, 2019.

[26] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[27] "Amazon elastic block store (Amazon EBS)," 2015. [Online]. Available: http://aws.amazon.com/ebs/

[28] "The open source DATAVIEW system," 2019. [Online]. Available: https://github.com/shiyonglu/DATAVIEW

[29] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for science grids: a viable solution?" in *Proceedings of the 2008 international workshop on Data-aware distributed computing*. ACM, 2008, pp. 55–64.

[30] S. Z. M. Mojab, M. Ebrahimi, R. G. Reynolds, and S. Lu, "iCATS: Scheduling big data workflows in the cloud using cultural algorithms," in *2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService)*. IEEE, 2019.