End-to-End Delay Minimization for Scientific Workflows in Clouds under Budget Constraint

Chase Qishi Wu, Member, IEEE, Xiangyu Lin, Student Member, IEEE, Dantong Yu, Member, IEEE, Wei Xu, Member, IEEE, and Li Li, Member, IEEE

Abstract—Next-generation e-Science features large-scale, compute-intensive workflows of many computing modules that are typically executed in a distributed manner. With the recent emergence of cloud computing and the rapid deployment of cloud infrastructures, an increasing number of scientific workflows have been shifted or are in active transition to cloud environments. As cloud computing makes computing a utility, scientists across different application domains are facing the same challenges of reducing the cost of computing. In addition to meeting the traditional goal of performance optimization, We develop a prototype generic workflow system by leveraging existing technologies for a quick evaluation of scientific workflow optimization strategies. We construct analytical models to quantify the network performance of scientific workflows using cloud-based computing resources, and formulate a task scheduling problem to minimize the workflow end-to-end delay under a user-specified financial constraint. We rigorously prove that the proposed problem is not only NP-complete but also non-approximable. We design a heuristic solution to this problem, and illustrate its performance superiority over existing methods through extensive simulations and real-life workflow experiments based on prototype implementation and deployment in a local cloud testbed.

Index Terms—Scientific workflows, workflow scheduling, cloud computing

1 INTRODUCTION

In today's large-scale scientific applications, computing tasks for data generation, processing, and analysis are often assembled and constructed as workflows comprised of many interdependent modules. A workflow module communicates with others through the sharing of data sets, which are either stored in a shared file system or transferred from one node to another by the workflow management system [24]. Such scientific workflows are typically executed in a distributed manner in heterogeneous network environments such as grids, which were developed as a distributed computing infrastructure to utilize various computing and networking resources deployed at different geographical locations [21].

With the emergence of cloud computing and the rapid deployment of cloud infrastructures, an increasing number of scientific workflows have been moved or are in active transition to clouds. Built on virtualization technologies over wide-area networks, cloud computing has established a new computing paradigm that creates a more flexible and cost-effective computing environment than any previous computing solution for scientific applications. In fact, cloud computing is becoming one of the major paradigms of modern scientific research, due to its great flexibility in scaling and large capacity in computing and storage brought by virtualization. Instead of managing and utilizing concrete nodes as in grids, clouds provision virtual servers working in federation through the Internet [21] to reduce the users' cost in purchasing, operating, and maintaining a physical computing infrastructure [15]. Furthermore, virtualization technologies allow multiple virtual machines (VMs) to be configured on one physical machine, resulting in more flexible sharing and higher utilization of physical resources [21].

The current cloud services are categorized into: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS clouds provide virtualized hardware and storage for users to deploy their own applications, and therefore are most suitable for executing scientific workflows. Real-world IaaS cloud services such as Amazon EC2 [1], provide VM instances with different CPU or memory capacities to meet different demands of various applications. Such VM instances are usually priced according to their processing powers but not necessarily linearly [26], and charged by the provisioned time units, such as hours.1 Within the same cloud, VMs work in a structure as a virtual cluster and data transfers are typically performed through a shared storage system without financial charge [2], [21]; while across different clouds, users generally need to pay for inter-cloud data transfers.

The shifting of scientific workflows to clouds has reaped the most significant benefit of resource virtualization by untangling application users from complex management and maintenance of underlying resources, but meanwhile has also brought many new challenges for workflow execution and

1. In the cost evaluation of workflow execution, any partial hours are often rounded up as in the case of EC2 [24].

C.Q. Wu and X. Lin are with the Department of Computer Science, University of Memphis, Memphis, TN 38152. 
E-mail: cqwu@memphis.edu.
D. Yu, W. Xu, and L. Li are with the Computational Science Center, Brookhaven National Laboratory, Upton, NY 11973. 
E-mail: dlyu, wxu, lilii@bnl.gov

For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2358220
performance optimization. Due to the nature of cloud computing that makes computing a utility, one major objective of resource provisioning in clouds is to allocate thus pay for only those cloud resources that are truly needed and used [26].

Hence, to support cost-effective execution of scientific workflows in cloud environments, scientists are now facing the challenge of reducing financial cost in addition to the traditional goal of optimizing workflow performance.

In this paper, we develop a prototype generic workflow system by leveraging existing technologies for a quick evaluation of performance optimization strategies for scientific workflows. We construct analytical models to quantify the network performance of scientific workflows in IaaS cloud environments, and formulate a task scheduling problem to minimize the workflow end-to-end delay under a user-specified financial cost constraint, referred to as Minimum End-to-end Delay under Cost Constraint (MED-CC) [25]. We rigorously prove that the proposed MED-CC problem is NP-complete and non-approximable. We design a heuristic solution to MED-CC and illustrate its performance superiority over existing methods through extensive simulations and experiments on real-life scientific workflows using the prototype workflow system deployed in a local cloud testbed. Our workflow scheduling solution can offer IaaS cloud service providers an economical resource allocation scheme to meet the budget constraint specified by the user, and meanwhile can also serve as a cloud resource provisioning reference for scientific users to make proactive and informative resource requests.

The rest of the paper is organized as follows. Section 2 conducts a survey of related work on workflow optimization especially in clouds. Section 3 presents the design of a prototype workflow system. Section 4 models the performance of scientific workflows in clouds and formulates MED-CC, whose complexity is analyzed in Section 5. Section 6 details the design of a workflow scheduling algorithm. Section 7 presents simulation and experimental results based on real-life scientific workflows in the prototype workflow system. Section 8 concludes our work.

2 RELATED WORK

We conduct a survey of related work on workflow optimization in heterogeneous networks and its extension to clouds.

Task scheduling or module mapping for workflows has been investigated extensively in the literature in the past decade [10], [18], [19], [31], [39]. Many heuristics have been proposed to minimize the workflow makespan (i.e., execution time) in grids, such as Heterogeneous Earliest Finish Time (HEFT) [37], and Hybrid Balanced Minimum Completion Time (HBMCT) [32]. HBMCT first assigns weights to the nodes and edges of a workflow graph, and then partitions the nodes into ordered groups and schedules independent tasks within each group. These scheduling algorithms have been demonstrated to be very effective in makespan minimization in their targeted simulation or experimental settings.

There also exist a relatively limited number of workflow scheduling efforts with a budget constraint in utility grids such as [31], [32], [33], [40]. In [33], Sakellariou et al. proposed two approaches, namely, LOSS and GAIN, to schedule DAG-structured workflow applications in grid environments. They start from a standard DAG schedule (such as HEFT or HMBCT) and a least-cost schedule, respectively, and reschedule the workflow to meet the budget constraint. In [41], Yu et al. performed a cost-based scheduling process through workflow task partitioning and deadline assignment to meet a user-defined deadline with the minimal cost. The workflow tasks are first categorized into synchronization tasks and simple tasks according to the number of their parent and child tasks. Interdependent simple tasks that are executed sequentially are then grouped into branches connected by synchronization tasks. The overall deadline specified by the user is divided into sub-deadlines over the task partitions in proportion to their minimal processing time.

In [8], Abruhami and Naghibzadeh designed a QoS-based workflow scheduling algorithm based on Partial Critical Paths (PCP) in SaaS clouds to minimize the cost of workflow execution within a user-defined deadline. As many existing critical-path (CP) heuristics, they schedule modules on the critical path first to minimize the cost without exceeding their deadline. PCP are then formed ending at those scheduled modules, and each PCP takes the start time of the scheduled critical module as its deadline. This scheduling process continues recursively until all modules are scheduled. In [26], [27], Mao et al. investigated the automatic scaling of clouds with budget and deadline constraints and proposed Scaling-Consolidation-Scheduling (SCS) with VMs as basic computing elements. In [20], Hacker proposed a combination of four scheduling policies based on an on-line estimation of physical resource usage.

Several recent efforts address workflow scheduling that takes both cost and delay into consideration in grid or cloud environments. Zeng et al. proposed a budget-conscious scheduler to minimize many-task workflow execution time within a certain budget [43]. BHEFT [44] by Zheng and Sakellariou extends HEFT to include budget constraint, and Arabnejad and Barbosa proposed HBSCS [9] to minimize workflow makespan under a cost constraint. Both BHEFT and HBSCS adjust their VM/processor selection parameter based on the current budget usage at each step.

In this paper, we construct analytical models to quantify workflow performance in IaaS clouds with VMs as fundamental computing elements, and incorporate the instance-hour billing model into cost calculation. These models facilitate a rigorous formulation of a budget-constrained end-to-end delay minimization problem for scientific workflows executed in clouds.

3 A PROTOTYPE SCIENTIFIC WORKFLOW SYSTEM

For a quick evaluation of workflow scheduling strategies on real-life scientific applications, we develop a prototype web-based scientific workflow system by leveraging open-source workflow technologies, including the well-established biology workflow systems such as Galaxy [17] and Tavern [22], MapReduce workflow systems such as Yahoo’s Oozie [30], Cascading [13], and Pig [29]. The incorporation of these existing technologies supports a point-and-click execution of complex workflows, starting from raw stream data to experiment metadata, log information, and images that can be further studied off-line at selected data centers.
or users’ home institutions. This system is able to track data provenance and versions as data are created/analyzed by either on-line data requisition and processing systems or users. It provides users an experimental notebook to track the path from raw data to intermediate and ultimately final analysis results with a desirable quality. Such data generation and processing paths are automatically recorded for users to replay at a later time.

The system consists of three main components: a user interface, a workflow manager, and an underlying resource layer to host data, tools, and computing resources. Through a web-based graphical interface, users are able to visually compose high-level workflows, specify input data sets and locations, and define the parameters of the data analysis tools and the desirable output locations. This web-based user interface allows users to access their data via smart devices or computers anywhere with an Internet connection, and also enables users to launch and steer the composed workflows, monitor the execution progress, and view the results of intermediate analyses in 2D or 3D images or animations.

Furthermore, we create a set of workflow template libraries and repositories for users to search and adapt for their own use. Users are also provided with the flexibility to customize the selected workflow template, and thereby specify the proper parameters via the graphical web interface or a simple Linux command-line interface. Some of these functionalities are already implemented in Galaxy [17], a leading open-source workflow system with graphical user interfaces for today’s genomics science. To save our development and implementation efforts, we leverage Galaxy’s web interface towards a cost-effective prototype workflow system.

The backend of the workflow system is a multi-layered architecture that consists of a web form handler processing user inputs, a central workflow manager including a job tracker and a workflow engine, and a database storing user information and job history. We adopt the Django framework [3] for pragmatic design and rapid development. The workflow system manages a set of data processing tools ranging from simple data loading functions to complex domain-specific programs. All these tools are made available to users for pointing-and-clicking. When a particular tool is selected, its inputs and configuration options are displayed on the web interface for convenient user interactions.

After the user composes and runs a data processing pipeline, a dynamic job instance for each individual tool is created, collectively forming a workflow that is represented as a directed acyclic graph (DAG), and then submitted to the workflow engine for executing, scheduling, tracking and reporting. The outputs of a data processing pipeline include multi-dimensional (1D, 2D, 3D, time-evolving 4D, and even higher) images, in which pixels or voxels represent the values of various physics properties, such as temperature, density, attenuation, and current.

In some cases, users might need to access the workflow system through their mobile devices for remote demonstrations. Since pipeline outputs could contain multi-gigabit image data, it is challenging, if not at all impossible, to transfer such data across the Internet to clients’ devices for remote user interactions. To address this challenge, we adopt WebGL [4], a JavaScript API for rendering interactive 3D visualization within any compatible web browser. WebGL leverages the graphics processing units at the client site to accelerate computation-intensive operations. Also, prior to visualization in the data processing pipeline, we downsample the image data to make it commensurate with the available LAN/WAN bandwidth. We develop network-aware image sampling/streaming modules to adapt the WebGL-based visualization system to display visualization results in heterogeneous network environments, including wireless and mobile cell-phone networks. This design enables large-scale big data processing prior to visualization, and effectively supports users to navigate through data and select a range of interests for further in-situ visualization and investigation.

For illustration purposes, we apply the prototype workflow system to the reconstruction pipeline in spectroscopy tomograph, where a collection of 2D absorption contrast (attenuation) projections at different sample-beam angles are measured with a series of incident monochromatic X-ray with a fixed range of energy spectrums, thereby generating a 4D dataset, as shown in Fig. 1. This processing pipeline enables 3D chemical mapping to gain insights into chemical compositions in biomaterials and nano-materials. Fig. 2 illustrates the reconstruction result of simulated samples, in which two chemical components are embedded into multiple spheres and the density of one element decreases from the center to the perimeter while the other increases.

The workflow scheduling algorithm we develop in the later sections is integrated into the workflow engine in the workflow manager of the prototype workflow system to maximize the efficiency of online workflow-based data processing in clouds. This prototype system is meant to evaluate the performance of our algorithm in real-life network environments.

4 Mathematical Models and Problem Formulation

We shall start with the generic notations in [38] for independent tasks, and extend them towards our workflow execution model with inter-module dependencies to identify and quantify the key financial and time cost components in clouds.
4.1 Cost Models

4.1.1 Time Cost

The execution time \( T_{i,j} \) of program \( P_i \) on virtual machine \( VM_j \) can be decomposed as:

\[
T_{i,j} = T(I_i) + T(E_{i,j}) + T(R_i),
\]

where i) \( T(I_i) \) denotes the startup time of \( VM_j \), ii) \( T(E_{i,j}) \) denotes the time for running \( P_i \) on \( VM_j \), and iii) \( T(R_i) \) denotes the time \( P_i \) takes to download and upload data to and from the virtual machine if it is running on.

4.1.2 Financial Cost

We consider a set \( VM \) of \( n \) available virtual machines and a set \( P \) of \( m \) programs to be executed. The financial cost \( C_{i,j} \) of executing a specific program \( P_i \), \( i \in \{0, 1, \ldots, m-1\} \), on a \( VM_j \), \( j \in \{0, 1, \ldots, n-1\} \), during a time span of \( T_{i,j} \) is the sum of four cost components:

\[
C_{i,j} = C(I_i) + C(E_{i,j}) + C(R_i) + C(S_i),
\]

where i) \( C(I_i) = I(VM_j) \) denotes the cost of initializing \( VM_j \), ii) \( C(E_{i,j}) \) denotes the cost of running program \( P_i \) on \( VM_j \), iii) \( C(R_i) \) denotes the cost of transferring the data required (downloading input data) and produced (uploading output data) by \( P_i \), and iv) \( C(S_i) \) denotes the data storage cost of \( P_i \). These cost components come from the VM leasing prices provided by cloud providers. The time duration \( T_{i,j} \) spans from the initialization of \( VM_j \) to the end of output data transfer from \( P_i \).

4.2 Workflow and Cloud Models

As illustrated in Fig. 3, there are three layers in the workflow scheduling problem in cloud environments: i) the task graph layer comprised of interdependent workflow modules, ii) the resource graph layer representing a network of VMs, and iii) the cloud infrastructure layer consisting of physical computer nodes connected by network links. Note that in the task graph layer, we consider scientific workflows that have been preprocessed by an appropriate clustering technique based on the inter-module dependencies and the volumes of inter-module data transfer [14], [23], [34], [36] such that a group of modules in the original workflow are bundled together as one aggregate module in the resulted task graph. The clustering techniques have been well studied in the literature and are beyond the scope of our work. We consider a one-to-one mapping scheme in our work such that each aggregate module in the task graph is assigned to a different VM for execution. This consideration requires us to choose the most suitable VM type, i.e., execution environment, for each module. However, in practice, once the schedule is obtained, we can always reuse the VMs of the same type to reduce the actual number of VMs being created and used based on the workflow structure and physical resource availability.

We decompose the layer-based hierarchy of cost models as follows:

- A task graph or workflow modeled as a directed acyclic graph \( G_{w}(V_w, E_w) \), consisting of \( |V_w| = m \) modules with \( |E_w| \) directed edges \( l_{ij} \) representing the data dependency between module \( w_i \) and \( w_j \). Each module \( w_i \in V_w \) carries a certain amount of workload \( WL_{w_i} \) and each edge \( l_{ij} \) transfers a certain data size \( DS_{l_{ij}} \).
- A set of available VM types \( VT = \{v_{t_0}, v_{t_1}, \ldots, v_{t_{m-1}}\} \), where each type \( v_{t_i} \) is associated with both cost- and performance-related attributes. We define each \( v_{t_i} \) as:

\[
v_{t_i} = \{vp_{t_i}, C_{V_{t_i}}\},
\]

where \( vp_{t_i} \) represents the overall processing power of the virtual machine of type \( v_{t_i} \) including its processor speed, disk volume and memory space, and \( C_{V_{t_i}} \) denotes the overall financial cost for using this virtual machine per time unit including all cost components such as VM initialization, module execution, and data transfer. Here, \( vp_{t_i} \) represents the sum of all
individual cores’ processing power in the case of a VM with a multi-core processor. For a parallel program, the overall processing power is used to calculate its execution time, while for a serial program, the processing power of one individual core is used.

- A cloud infrastructure graph modeled as an arbitrary weighted graph $G_c(V_c, E_c)$ consisting of a set of physical computer nodes. Each computer node $c_i \in V_c$ has a processing power $P_{c_i}$. The cost of a data transfer $R_{i,j}$ of size $D_{i,j}$ from module $w_i$ to $w_j$ is calculated as:

$$C(R_{i,j}) = C_R \cdot D_{i,j},$$

where $C_R$ is the fixed data transfer cost per data unit determined by the cloud infrastructure. In the case of intra-cloud data transfer, $C_R = 0$.

- A fully connected virtual resource graph modeled as an arbitrary weighted graph $G'_c(V'_c, E'_c)$ consisting of $|V'_c|$ VMs interconnected by $|E'_c|$ virtual links. Each virtual link $l'_{p,q}$ has a bandwidth $BW'_{p,q}$, which is a function of the physical bandwidth between two physical machines provisioning VMs $c'_p$ and $c'_q$. Note that each VM $c' \in V'_c$ is an instance of a certain VM type. The time of a data transfer $R_{i,j}$ of data size $D_{i,j}$ from module $w_i$ to $w_j$ is calculated as:

$$T(R_{i,j}) = \frac{D_{i,j}}{BW'_{p,q}} + d'_{p,q},$$

where $d'_{p,q}$ denotes the delay of the virtual link.

- The time $T(E_{i,j})$ of the execution $E_{i,j}$ of module $w_i$ on a VM of type $v_{t_j}$ is a function of $w_i$’s workload and $v_{t_j}$’s performance-related attributes:

$$T(E_{i,j}) = \frac{WL_i}{tp_{t_j}},$$

and the corresponding module execution cost is calculated as:

$$C(E_{i,j}) = T(E_{i,j}) \cdot C_{V_{t_j}}.$$

We denote each module $w_i$’s earliest start time and earliest finish time as $esl(w_i)$ and $efl(w_i)$, respectively, and denote its latest start time and latest finish time as $isl(w_i)$ and $lfl(w_i)$, respectively. These time parameters are dynamically calculated during the scheduling phase since the CP might change at each step when a module is rescheduled to a different type of VM. The module’s buffer time $buffer(w_i)$, defined as $isl(w_i) - esl(w_i)$ or $lfl(w_i) - efl(w_i)$, is the amount of time the execution of module $w_i$ can be delayed without affecting the overall end-to-end delay of the entire workflow. The critical path is the longest path in the task graph weighted with time cost, which consists of all the modules with zero buffer time.

Since our work targets cloud environments in a single datacenter where physical machines are interconnected by high-bandwidth links, we do not specifically consider the initialization and data transfer cost components. We use $T_{i,j}$ and $C_{i,j}$ to denote the overall execution time and cost of module $w_i$ on VM type $v_{t_j}$.

### 4.3 Problem Formulation

Based on the mathematical models constructed above, we formally formulate a workflow scheduling problem for Minimum End-to-end Delay under a user-specified Cost Constraint in cloud environments, referred to as MED-CC:

**Definition 1 (MED-CC).** Given a DAG-structured workflow graph $G_w(V_w, E_w)$, a set of available virtual machine types $VT = \{v_{t_0}, v_{t_1}, \ldots, v_{t_{m-1}}\}$, and a fixed financial budget $B$, we wish to find a task schedule $\forall w_i \rightarrow v_{t_j}, \forall w_i \in V_w, \exists t_j \in VT$ such that the minimum end-to-end delay of the one-to-one mapped workflow is achieved:

$$MED = \min_{all \ possible} (T_{Total}) = \min_{all \ possible} \left( \sum_{\forall w_i \in CP} T_{i,j} \right),$$

subject to the financial constraint:

$$C_{Total} = \sum_{i=0}^{m-1} C_{i,j} \leq B.$$

In the above definition, $T_{Total}$ and $C_{Total}$ denote the total execution time and the total financial cost of a mapped workflow, respectively, and $C_{i,j}$ denotes the execution cost of module $w_i$ on a virtual machine of type $v_{t_j}$. For convenience of reference, we tabulate the notations used in our cost models and algorithm design in Table 1.

### 5 Complexity Analysis

We have the following theorem on the complexity of the MED-CC problem.

**Theorem 1.** MED-CC is NP-complete.

**Proof.** Obviously, MED-CC $\in$ NP, since given any solution (a schedule), we can always verify in polynomial time if it is a feasible solution by calculating and comparing the associated cost with the given budget.

We prove the NP-hardness of MED-CC by showing that the Multiple-Choice Knapsack Problem (MCKP) [28], which is a well-known classical NP-complete problem, is a special case of MED-CC. We first provide the definition of MCKP as follows:

**Definition 2 (MCKP).** Given $m$ classes $N_1, \ldots, N_m$ of items to pack in a knapsack of capacity $c$, where each item $j \in N_i$, $i = 1, 2, \ldots, m$, has a profit $p_{ij}$ and a weight $w_{ij}$, the problem is to choose exactly one item from each class such that the sum of profits is maximized while the sum of weights does not exceed capacity $c$, formulated as:

- Maximize $z = \sum_{i=1}^{m} \sum_{j \in N_i} p_{ij}x_{ij}$.

- Subject to
  $$\sum_{i=1}^{m} \sum_{j \in N_i} w_{ij}x_{ij} \leq c_i,$$
  $$\sum_{j \in N_i} x_{ij} = 1, i = 1, \ldots, m,$$
  $$x_{ij} \in \{0, 1\}, j \in N_i, i = 1, \ldots, m.$$
TABLE 1
The Notations Used in the Cost Models and Algorithm Design

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_{w}(V_{w}, E_{w})$</td>
<td>a computing workflow</td>
</tr>
<tr>
<td>$m$</td>
<td>the number of modules in the workflow</td>
</tr>
<tr>
<td>$w_i$</td>
<td>the $i$-th computing module</td>
</tr>
<tr>
<td>$w_0$</td>
<td>the entry (start) module</td>
</tr>
<tr>
<td>$w_{m-1}$</td>
<td>the exit (end) module</td>
</tr>
<tr>
<td>$est(w_i)$</td>
<td>the earliest start time</td>
</tr>
<tr>
<td>$lat(w_i)$</td>
<td>the latest start time</td>
</tr>
<tr>
<td>$buf(w_i)$</td>
<td>the module buffer time</td>
</tr>
<tr>
<td>$e_{i,j}$</td>
<td>the dependency edge from module $w_i$ to $w_j$</td>
</tr>
<tr>
<td>$DS_{i,j}$</td>
<td>the data size associated with dependency edge $e_{i,j}$</td>
</tr>
<tr>
<td>$T(R_{i,j})$</td>
<td>the time of a data transfer $R_{i,j}$ from module $w_i$ to $w_j$</td>
</tr>
<tr>
<td>$WL_{i,j}$</td>
<td>the workload of module $w_i$</td>
</tr>
<tr>
<td>$C(R_{i,j})$</td>
<td>the cost of a data transfer $R_{i,j}$ from module $w_i$ to $w_j$</td>
</tr>
<tr>
<td>$C_{R}$</td>
<td>the data transfer cost per data unit</td>
</tr>
<tr>
<td>$G_v(V', E')$</td>
<td>a virtual resource graph</td>
</tr>
<tr>
<td>$m'$</td>
<td>the number of virtual nodes in the network</td>
</tr>
<tr>
<td>$c'_{i}$</td>
<td>the virtual link from node $v_i'$ to $v_j'$</td>
</tr>
<tr>
<td>$l_{i,j}'$</td>
<td>the link delay of $l_{i,j}'$</td>
</tr>
<tr>
<td>$BW_{i,j}'$</td>
<td>the virtual bandwidth of $l_{i,j}'$</td>
</tr>
<tr>
<td>$G_c(V_c, E_c)$</td>
<td>a cloud infrastructure graph</td>
</tr>
<tr>
<td>$c_{i}$</td>
<td>the $i$-th computer node</td>
</tr>
<tr>
<td>$l_{i,j}$</td>
<td>the network link from node $v_i$ to $v_j$</td>
</tr>
<tr>
<td>$d_{i,j}$</td>
<td>the link delay of $l_{i,j}$</td>
</tr>
<tr>
<td>$BW_{i,j}$</td>
<td>the bandwidth of link $l_{i,j}$</td>
</tr>
<tr>
<td>$VT$</td>
<td>a set of available VM types</td>
</tr>
<tr>
<td>$n$</td>
<td>the number of VM types</td>
</tr>
<tr>
<td>$v_{t_j}$</td>
<td>the $j$-th VM type</td>
</tr>
<tr>
<td>$C_{V_j}$</td>
<td>the cost per time unit for using a VM of type $v_{t_j}$</td>
</tr>
<tr>
<td>$C_{p_{j}}$</td>
<td>the overall computing power of $v_{p_{j}}$</td>
</tr>
<tr>
<td>$B$</td>
<td>the budget</td>
</tr>
<tr>
<td>$T_{Total}$</td>
<td>the total execution time of workflow execution</td>
</tr>
<tr>
<td>$C_{t_{j}}$</td>
<td>the execution cost of $v_{t_{j}}$ on a VM of type $v_{t_{j}}$</td>
</tr>
<tr>
<td>$\Delta_{C_{t_{j}}}$</td>
<td>the cost difference in rescheduling</td>
</tr>
<tr>
<td>$T_{i,j}$</td>
<td>the execution time of $v_{t_{j}}$ on a VM of type $v_{t_{j}}$</td>
</tr>
<tr>
<td>$\Delta_{T_{i,j}}$</td>
<td>the time difference in rescheduling</td>
</tr>
<tr>
<td>$C_{min}$</td>
<td>the minimum financial cost</td>
</tr>
<tr>
<td>$C_{Total}$</td>
<td>the total financial cost</td>
</tr>
<tr>
<td>MED</td>
<td>the minimal end-to-end delay among all schedules</td>
</tr>
<tr>
<td>$\mathcal{S}$</td>
<td>a schedule mapping $V_{w}$ to $VT$</td>
</tr>
</tbody>
</table>

We consider a special case of MED-CC where the workflow only consists of a linear pipeline of $m$ modules with zero data transfer time over each dependency edge. For convenience, we refer to this special case as MED-CC-Pipeline. For each VM type $v_{t_{j}}$, we compute an estimated performance vector [26] as:

$$
T_{E}[j] = [T_{0,j}, T_{1,j}, \ldots, T_{m-1,j}],
$$

where $T_{i,j}$ represents the execution time of module $w_i$ on a virtual machine of type $v_{t_{j}}$. Accordingly, we compute a vector of execution costs $C_{E}[j] = [C_{0,j}, C_{1,j}, \ldots, C_{(m-1,j)}]$, where $C_{i,j} = C_{V_j} \cdot T_{i,j}$. The optimization goal is to minimize the total execution time $T_{Total}$ of the pipeline-structured workflow subject to the constraint of a given financial budget $B$.

We show that MED-CC-Pipeline is essentially the MCKP problem. As illustrated in Fig. 4, $m$ modules in MED-CC-Pipeline correspond to $m$ classes in MCKP, and $n$ available VM types for each module $w_i$ in MED-CC-Pipeline correspond to $n$ items in each class $N_i$ in MCKP. We set $c = B$ and assign each item in a class in MCKP a weight, which is equivalent in value to the corresponding financial cost $C_{i,j}$ in MED-CC-Pipeline, and a profit, which is equivalent in value to $K - T_{i,j}$ in MED-CC-Pipeline, where $K$ is a sufficiently large constant such that $K \geq T_{i,j}, \forall i,j$. Under this problem construction, MCKP and MED-CC-Pipeline are the same problem that chooses exactly one item or VM type per class or module to maximize the total profit or minimize the total execution time $T_{Total}$, subject to the knapsack capacity or financial budget constraint $B$. Since the data transfer in MED-CC-Pipeline is ignored, $T_{Total}$ is simply the sum of execution times of all modules and the module dependency along the pipeline does not affect the complexity of the problem.

Since a special case is NP-complete, the original MED-CC problem is also NP-complete. The NP-completeness proof by restriction is established in [16].

**Theorem 2.** MED-CC does not have polynomial-time approximation scheme (PTAS), unless $P = NP$.

**Proof.** We prove the non-approximability of MED-CC by gap-reduction. Assuming that a PTAS exists, i.e., for each positive constant $\varepsilon > 0$, there exists an approximate algorithm $\mathcal{A}_{\varepsilon}$ to MED-CC problem instance $I_{MED}$ with an approximation ratio of $1 + \varepsilon$, i.e., $\mathcal{A}_{\varepsilon}(I_{MED}) = (1 + \varepsilon) \cdot OPT(I_{MED})$, where $OPT(I_{MED})$ denotes the optimal solution.

Consider an arbitrary instance $I_{MCKP}$ of MCKP consisting of $m$ classes with $n_i$ ($n_i \geq 1, i = 1, 2, \ldots, m$) items in the $i$th class, knapsack capacity $c$ and a positive constant $PB$. The decision version of MCKP asks if there exists a choice of one item per class such that the total profit is no less than the profit bound $PB$ with the total weight within $c$.

We first increase the number of items in each class to $n = n_{max} = \max_{i=1,2,\ldots,m}(n_i)$ by padding $n - n_i$ dummy items, each with zero profit and a weight larger than the weight of each of the original $n_i$ items, such that they would not be included in any schedule, thus not affecting the solution.
The given information for \( I_{MED} \) construction are

\[
    v_p_{\text{max}} > v_p_j, j = 1, \ldots, n,
\]

\[
    I_{MCKP} = \{ c, PB, p_{ij}, w_{ij} \}, i = 1, \ldots, m, j = 1, \ldots, n,
\]

\[
    K > \max \left\{ \frac{PB}{m}, \frac{v_p_{\text{max}, \text{max}}}{m} \right\},
\]

where \( v_p_{\text{max}} \) is the maximum computing power of VM types, \( K \) is a positive constant chosen to be greater than all profit values \( p_{ij} \) in \( I_{MCKP} \). We can always construct \( I_{MED} \) by creating a linear pipeline of \( m \) modules, and a set of \( n \) VM types, and setting the budget \( B = c \). The parameters left to be determined are

\[
    B, WL_i, i = 1, \ldots, m, v_p_j, CV_j, j = 1, \ldots, n.
\]

The key correspondences to be constructed are:

\[
    T_{i,j} = \frac{WL_i}{v_p_j} \geq K - p_{ij}, \quad (10)
\]

\[
    C_{i,j} \geq w_{ij}, i = 1, \ldots, m, \quad j = 1, \ldots, n, \quad (11)
\]

such that the execution of the \( i \)-th module on the \( j \)-th VM type in \( I_{MED} \) exactly corresponds to the \( j \)-th item from the \( i \)-th class in \( I_{MCKP} \). For each module \( w_i \) in MED-CC, we set its workload as

\[
    WL_i = v_p_{\text{max}} \cdot (K - p_{i, \text{min}}), \quad (12)
\]

where

\[
    p_{i, \text{min}} = \min_{j=1,2,\ldots,n} (p_{ij}),
\]

and we set the processing power of each VM type as

\[
    v_p_j = v_p_{\text{max}} \cdot \frac{p_{\text{max}, j}}{p_{\text{max}, \text{max}}}. \quad (13)
\]

According to Eq. (10), the execution time is

\[
    T_{i,j} = \frac{v_p_{\text{max}, \text{max}} \cdot (K - p_{\text{min}})}{p_{\text{max}, j}} \geq K - p_{i, \text{min}}
\]

\[
    \geq K - p_{ij}, \quad \forall (I_{MCKP}) \geq m \cdot K - \forall (I_{MED}),
\]

thus the inequality in Eq. (10) holds, and we can obtain the relation between the corresponding solutions to both instances.

The charging rate for \( CV_j \) for VM type with computing power \( v_p_j \) is set to be

\[
    CV_j = \frac{w_{\text{max}, j}}{T_{\text{min}, j}}, \quad (14)
\]

where

\[
    w_{\text{max}, j} = \max_{i=1,2,\ldots,m} (w_{ij}), \quad T_{\text{min}, j} = \min_{i=1,2,\ldots,m} (T_{ij}),
\]

such that the inequality in Eq. (11) holds, i.e., the selection of VM types that fit within budget \( B \) in \( I_{MED} \) always imply the selection of the corresponding items that fit within knapsack capacity \( c \) in \( I_{MCKP} \).

The approximation algorithm \( A_c \) for MED-CC returns the solution \( A_c(I_{MED}) \in \{ OPT(I_{MED}), (1 + \epsilon) \cdot OPT(I_{MED}) \} \).

For those values of \( \epsilon \) that satisfy

\[
    m \cdot K - (1 + \epsilon) \cdot OPT(I_{MED}) \geq PB, \quad (15)
\]

\( I_{MCKP} \) is a YES instance, and \( A_c(I_{MED}) \) can give the answer in polynomial time, which implies \( P = NP \).

Hence, PTAS does not exist for MED-CC. Proof ends. \( \square \)

Based on the above complexity analysis, we conclude that MED-CC is NP-complete and non-approximable, which rules out the existence of any polynomial-time optimal or approximate solution. Therefore, we shall focus on the design of a heuristic approach to this optimization problem.

## 6 Algorithm Design

Our work targets compute-intensive workflows in a single datacenter cloud, where the data transfer time is assumed negligible since: i) we consider aggregate workflows where modules have been bundled into groups using a clustering technique such that the inter-module data transfer is minimized, and ii) the time for data transfers in most compute-intensive scientific workflows constitutes less than 10% of the overall workflow execution time [21]. For the same reason, widely adopted cloud computing models such as those in CloudSim [12] use high-bandwidth network connections by default within the same datacenter, and only inter-datacenter data transfers are considered.

### 6.1 Critical-Greedy(CG) Algorithm

We propose the Critical-Greedy algorithm as shown in Algorithm 1. CG first estimates cost boundaries based on the workflow and VM types, and generates an appropriate initial schedule based on the relative level of a given budget (Algorithm 2). It then refines the schedule by using the left cost to speed up the critical modules until no more rescheduling is possible (Algorithm 3). The time complexity of CG is \( O(m \cdot n) \), where \( m \) is the number of workflow modules and \( n \) is the number of VM types.

**Algorithm 1. Critical-Greedy(\( G_{m, \text{VT}, B} \))**

Input: a workflow graph \( G_m(V_m, E_m) \), a set \( \text{VT} \) of VM types, a fixed budget \( B \).

Output: the workflow’s end-to-end delay \( ED \).

1: for all \( w_i \in V_m \) do
2:     for all \( v_j \in \text{VT} \) do
3:         Calculate \( C_{i,j} \) and \( T_{i,j} \);  
4:         Find the minimal cost as \( C_{\text{min}} \);  
5:         if \( B < C_{\text{min}} \) then  
6:             Exit with an error.  
7:         Find the maximal cost as \( C_{\text{max}} \);  
8:         if \( B \geq C_{\text{max}} \) then  
9:             for all \( w_i \in V_m \) do
10:                Map \( w_i \) to VM type \( v_{\text{max}, i} \) with the minimum delay;  
11:                return the end-to-end delay \( ED \);  
12:                Calculate the global budget level \( gb_l = B - C_{\text{min}} \);  
13:                for all \( w_i \in V_m \) do
14:                    Map \( w_i \) to VM type \( v_i = \text{selectInitType}(w_i, \text{VT}, gb_l) \);  
15:                Compute the current end-to-end delay as \( ED \);  
16:                Compute the current cost as \( C_{\text{current}} \);  
17:                \( ED = \text{Critical-Allocate}(G_m, B - C_{\text{current}}, \text{VT}) \);  
18:                return \( ED \).
Algorithm 2. selectInitType($w_j, VT, gbl$)

**Input:** a module $w_j$, a set $VT$ of VM types, a global budget level $gbl$.

**Output:** a VM type $v_{t,j}$.

1: Select the VM type with the minimum delay as $v_{t,min}$, and the minimum cost as $v_{t,min}$.
2: Calculate the execution cost on $v_{t,min}$ as $C_{t,min}$, and the execution cost on $v_{t} = C_{t,min}$.
3: Set $targetcost = C_{t,min} + (C_{t,max} - C_{t,min}) \times gbl$.
4: Select VM type $v_{t,j} = \arg \min_{v_{t} \in VT} \{C_{t,J} - targetcost\}$, where $C_{t,J} \leq targetcost$.
5: return $v_{t,j}$.

Algorithm 3. Critical-Allocate($G_w, CostToSpend, VT$)

**Input:** a workflow graph $G_w(V_w, E_w)$, an amount of budget $CostToSpend$, a set $VT$ of VM types.

**Output:** the workflow's end-to-end delay $ED$.

1: while $CostToSpend > 0$ do
2: Calculate the current critical path $CP$.
3: for all $w_k \in CP$ do
4: for all $v_{t,j} \in VT$ do
5: Calculate the time decrease $\Delta T_{i,j}$ and the cost increase $\Delta C_{i,j}$ from $v_{t,j}$ to the VM type $w_k$ is currently mapped to;
6: Record the VM type $v_{t,j}$ with the maximum $\Delta T_{i,j}$, where $\Delta T_{i,j} > 0$;
7: Set $\Delta(w_k) = \frac{\Delta T_{i,j}}{\Delta C_{i,j}}$;
8: Select module $w_k = \arg \max_{w_k \in CP} \Delta(w_k)$;
9: if no such $w_k$ is found then
10: break;
11: Map $w_k$ to $v_{t,j}$;
12: return $ED$.

In Algorithm 1, the global budget level $gbl = \frac{B - C_{t,max}}{C_{t,max} - C_{t,min}}$ represents where the given budget $B$ falls between the minimum cost possible $C_{t,min}$ and the maximum cost necessary $C_{t,max}$ of the entire workflow. In the initial schedule, each module $w_k$ is mapped to a VM type $v_{t,k}$ such that its local budget level $lbd = \frac{C_{t,J} - C_{t,min}}{C_{t,max} - C_{t,min}}$ is within and is the closest to the global $gbl$, as determined by Algorithm 2.

The initial schedule will not exceed the given budget. To optimize budget allocation, we could then reallocate such cost left to critical modules to further reduce the end-to-end delay of the workflow.

The ratio of the time and cost difference among different schedules has been adopted in many cost related workflow scheduling algorithms, such as the Cost Decrease Ratio (CDR) in [8], LossWeight, and GainWeight in [33], the Comparative Advantage (CA) in [43], and the rank value for deadline assignment in [27]. Since the effectiveness of this parameter has been established by many existing efforts, we also utilize it in the budget tuning phase of Critical-Greedy as a condition for VM type selection.

In Algorithm 3, we reallocate the left cost from Alg.2 to speed up the workflow execution. We always select the critical modules and upgrade their VM types to more powerful ones, with an attempt to maximize the impact of a local speed-up on the global end-to-end delay. At each iteration, only those modules on the CP are considered. For each critical module, we first calculate its local maximum ratio of the time decrease over the cost increase and record the corresponding VM type, and then select the module that achieves the global maximum ratio over the entire CP for rescheduling.

7 ALGORITHM IMPLEMENTATION AND PERFORMANCE EVALUATION

7.1 Simulation Settings

We evaluate the performance of Critical-Greedy using CloudSim [12], which has been widely adopted for the modeling and evaluation of cloud-based solutions. Particularly, we extended CloudSim to support DAG-structured workflows and scheduling algorithms.

We implement Critical-Greedy in CloudSim and evaluate its performance in comparison with two most recently published algorithms, i.e. ScaleStar [43] and HBCS [9]. Both ScaleStar and HBCS process workflow modules in a decreasing order of $l$-level, i.e. the length of the longest path from the current module to the end module, and then select a VM/processor for each module based on a certain criteria: the Comparative Advantage in ScaleStar and the worthiness in HBCS. Note that CA can be computed based on the module and VM type statically, while worthiness is adjusted by the overall budget consumption status in the current step. We choose ScaleStar for comparison because it outperforms both LOSS and GAIN algorithms [33], and we choose HBCS for comparison because it outperforms LOSS algorithms [33], BHEFT [44], and GreedyTime_CD [42].

To evaluate the algorithm performance, we consider different problem sizes from small to large scales. The problem size is defined as a three-tuple $(m, |E_w|, n)$, where $m$ is the number of workflow modules, $|E_w|$ is the number of workflow links, and $n$ is the number of available VM types. We generate workflow instances with different scales in a random manner as follows: (i) lay out all $m$ modules sequentially along a pipeline, each of which is assigned a workload randomly generated within range $[5,500]$, which represents total number of instructions; (ii) for each module, add an input edge from a randomly selected preceding module and add an output edge to a randomly selected succeeding module (note that the first module only needs output and the last module only needs input); (iii) randomly pick up two modules from the pipeline and add a directed edge between them (from left to right) until we reach the given number of edges.

We use the pricing model for VMs according to real-world cloud service providers such as EC2. The price is a linear function of the number of processing units in the VM type. In our simulation, we set a base processing unit to be the processing power of 10 instructions per time unit, with a base price at 0.01 per time unit. The financial cost of each VM type is priced according to the number of base processing units. When generating a given number of VM types, type $v_{t,j}$ has $2j$ base processing units for $j > 0$, and $v_{t,0}$ has one base processing unit.

We compare our Critical-Greedy algorithm with ScaleStar [43] and HBCS [9] in terms of the total workflow execution time under the same budget constraint. The MED.
performance improvement of CG over the other algorithms in comparison is defined as:

$$Imp(\text{Other}) = \frac{MED_{\text{Other}} - MED_{CG}}{MED_{\text{Other}}} \times 100\%,$$

where $MED_{\text{Other}}$ is the MED achieved by the other algorithms, i.e., ScaleStar or HBCS, and $MED_{CG}$ is the MED achieved by Critical-Greedy.

7.2 Simulation Results

7.2.1 Comparison with Optimal Solutions

We first compare the performance of Critical-Greedy with optimal solutions in small-scale problems with 5, 6, 7, and 8 modules and available VM types. For each problem size, we randomly generate 100 problem instances with different module workload and DAG topology. For each problem instance, we calculate the corresponding least cost $C_{\text{min}}$ and minimum cost $C_{\text{max}}$, and then iterate through 20 budget levels at an equal interval of $\Delta C = (C_{\text{max}} - C_{\text{min}})/20$ in this range. We run all three algorithms on these instances and compare the MED results with the optimal ones computed by an exhaustive search approach. Fig. 5 shows the number of optimal results among 2,000 instances (100 workflow instances $\times$ 20 budget levels) produced by CG, HBCS, and ScaleStar with different problem sizes. We observe that CG is more likely to achieve the optimality than the other two algorithms in a statistical sense, which indicates the efficacy of CG.

7.2.2 Comparison with HBCS and ScaleStar

We further consider 20 problem sizes from small to large scales, indexed from 1 to 20. In each problem size, we randomly generate 50 problem instances, and for each of them, we run the scheduling simulation by iterating through 20 budget levels as earlier. We provide in Table 3 the average MED improvement percentages together with standard deviations achieved by CG over HBCS and ScaleStar (SS) across all the 20 problem sizes, which are further plotted in Fig. 7 for a visual comparison.

For a better illustration, we plot the overall performance improvement percentage of Critical-Greedy over HBCS in Fig. 8, where $x$ axis denotes the budget increment across 20 levels and $y$ axis denotes the index of 20 problem sizes from small to large scales. Each point $(x, y, imp)$ in the 3D plot represents the average performance improvement across all 50 problem instances of the same problem size under the same budget level (the actual budget values may be different in different instances). These performance results show

| Prb Idx | $(m, |E_m|, n)$ | Imp (HBCS) % | StdDev | Imp (SS) % | StdDev |
|---------|----------------|--------------|--------|------------|--------|
| 1       | (5, 6, 5)      | 6.57         | 10.85  | 3.90       | 5.97   |
| 2       | (10, 15, 6)    | 11.54        | 9.51   | 8.79       | 8.34   |
| 3       | (15, 60, 7)    | 9.26         | 6.63   | 8.71       | 9.88   |
| 4       | (20, 80, 8)    | 10.81        | 6.00   | 11.49      | 10.56  |
| 5       | (25, 200, 9)   | 11.53        | 6.41   | 13.98      | 11.58  |
| 6       | (30, 300, 10)  | 11.91        | 7.81   | 16.02      | 12.51  |
| 7       | (35, 500, 11)  | 11.87        | 9.03   | 17.73      | 13.34  |
| 8       | (40, 500, 12)  | 11.95        | 9.84   | 19.62      | 13.35  |
| 9       | (45, 500, 13)  | 11.64        | 9.45   | 21.22      | 13.43  |
| 10      | (50, 500, 14)  | 11.65        | 9.04   | 22.50      | 13.68  |
| 11      | (55, 500, 15)  | 11.42        | 9.19   | 23.78      | 13.61  |
| 12      | (60, 900, 16)  | 11.40        | 9.80   | 24.88      | 13.67  |
| 13      | (65, 950, 17)  | 11.34        | 10.21  | 25.86      | 13.75  |
| 14      | (70, 950, 18)  | 11.05        | 10.43  | 26.70      | 13.84  |
| 15      | (75, 1000, 19) | 10.90        | 10.66  | 27.60      | 13.62  |
| 16      | (80, 1200, 20) | 10.87        | 10.77  | 28.44      | 13.45  |
| 17      | (85, 1200, 21) | 10.87        | 10.89  | 29.18      | 13.36  |
| 18      | (90, 1600, 22) | 10.76        | 11.19  | 29.80      | 13.26  |
| 19      | (95, 1600, 23) | 10.63        | 11.49  | 30.41      | 13.20  |
| 20      | (100, 2000, 24)| 10.36        | 11.72  | 30.96      | 13.16  |

Fig. 6. The average MED performance improvement percentages (percent) with standard deviations across 1,000 instances (20 different budget levels $\times$ 50 random workflow instances) for each problem size.
The Average MED Improvement Percentages (Percent) of CG over HBBCS and ScaleStar (SS) and the Corresponding Standard Deviations Across 1,000 Instances (20 Problem Sizes × 50 Random Workflow Instances) at Different Budget Levels

<table>
<thead>
<tr>
<th>Budget Level</th>
<th>Imp(HBBCS) %</th>
<th>StdDev</th>
<th>Imp(SS) %</th>
<th>StdDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27.38</td>
<td>11.31</td>
<td>0.25</td>
<td>3.35</td>
</tr>
<tr>
<td>2</td>
<td>31.89</td>
<td>10.30</td>
<td>0.45</td>
<td>4.47</td>
</tr>
<tr>
<td>3</td>
<td>26.59</td>
<td>6.89</td>
<td>1.94</td>
<td>3.49</td>
</tr>
<tr>
<td>4</td>
<td>14.70</td>
<td>7.90</td>
<td>6.23</td>
<td>7.01</td>
</tr>
<tr>
<td>5</td>
<td>6.58</td>
<td>8.73</td>
<td>10.53</td>
<td>9.23</td>
</tr>
<tr>
<td>7</td>
<td>3.80</td>
<td>6.29</td>
<td>13.91</td>
<td>10.53</td>
</tr>
<tr>
<td>7</td>
<td>3.54</td>
<td>3.84</td>
<td>16.79</td>
<td>11.20</td>
</tr>
<tr>
<td>8</td>
<td>4.39</td>
<td>4.05</td>
<td>19.57</td>
<td>13.33</td>
</tr>
<tr>
<td>9</td>
<td>5.52</td>
<td>3.61</td>
<td>21.64</td>
<td>11.54</td>
</tr>
<tr>
<td>10</td>
<td>6.62</td>
<td>2.80</td>
<td>23.65</td>
<td>11.40</td>
</tr>
<tr>
<td>11</td>
<td>7.38</td>
<td>2.68</td>
<td>25.50</td>
<td>11.01</td>
</tr>
<tr>
<td>12</td>
<td>7.98</td>
<td>2.31</td>
<td>26.85</td>
<td>11.04</td>
</tr>
<tr>
<td>13</td>
<td>8.78</td>
<td>2.19</td>
<td>28.34</td>
<td>10.49</td>
</tr>
<tr>
<td>14</td>
<td>9.22</td>
<td>2.84</td>
<td>29.56</td>
<td>10.23</td>
</tr>
<tr>
<td>15</td>
<td>9.81</td>
<td>2.53</td>
<td>30.49</td>
<td>10.29</td>
</tr>
<tr>
<td>16</td>
<td>10.13</td>
<td>2.81</td>
<td>31.36</td>
<td>10.08</td>
</tr>
<tr>
<td>17</td>
<td>10.63</td>
<td>2.78</td>
<td>32.38</td>
<td>9.44</td>
</tr>
<tr>
<td>18</td>
<td>11.08</td>
<td>2.89</td>
<td>33.30</td>
<td>8.85</td>
</tr>
<tr>
<td>19</td>
<td>11.46</td>
<td>2.85</td>
<td>34.05</td>
<td>8.49</td>
</tr>
<tr>
<td>20</td>
<td>0.79</td>
<td>3.00</td>
<td>34.76</td>
<td>8.01</td>
</tr>
</tbody>
</table>

**Fig. 7.** The average performance improvement percentages with standard deviations across 1,000 instances (20 different problem sizes × 50 random workflow instances) for each budget level.

that Critical-Greedy achieves an average of 10 percent performance improvement over HBBCS. Similarly, we plot the overall performance improvement percentage of Critical-Greedy over ScaleStar in Fig. 9. These performance results show that Critical-Greedy achieves an average of 25 percent performance improvement over ScaleStar. Such performance improvements are considered significant for workflow execution in large-scale scientific applications.

### 7.3 Experimental Results

#### 7.3.1 WRF Workflow

Our workflow experiments are based on the Weather Research and Forecasting (WRF) model [35], which has been widely adopted for regional to continental scale weather forecast. The WRF model [5] generates two large classes of simulations either with an ideal initialization or utilizing real data. In our experiments, the simulations are generated from real data, which usually requires preprocessing from the WPS package [6] to provide each atmospheric and static field with fidelity appropriate to the chosen grid resolution for the model.

The structure of a general WRF workflow is illustrated in Fig. 10, where the WPS consists of three independent programs: geogrid.exe, ungrib.exe, and metgrid.exe. The geogrid program defines the simulation domains and interpolates various terrestrial datasets to the model grids. The user can specify information in the namelist file of WPS to define simulation domains. The ungrib program “degrin” the data and stores the results in a simple intermediate format. The metgrid program horizontally interpolates the intermediate-format meteorological data that are extracted by the ungrib program onto the simulation domains defined by the geogrid program. The interpolated metgrid output can then be ingested by the WRF package, which contains an initialization program real.exe for real data and a numerical integration program wrf.exe. The postprocessing model consists of ARWpost and GrADS. ARWpost reads in WRF-ARW model data and creates output files for display by GrADS.
7.3.2 Experiment Settings

Nimbus [7] is an open-source toolkit focused on providing IaaS capabilities to the scientific community. The basic structure of a Nimbus cloud consists of a central controller node and multiple virtual machine monitor (VMM) nodes. VM images are uploaded through a client program to the storage repository on the controller node, and VMs are provisioned upon client requests on available VMM nodes. We set up a local Nimbus cloud testbed with Xen virtualization tools [11]. This private cloud contains five physical servers: one serves as the cloud controller and VM images repository, and the others are VMM nodes where VMs are deployed. We execute the WRF workflow using our prototype workflow system in this cloud testbed.

We duplicate three WRF pipelines each from ungrb1.exe to ARWpost.exe, and group these programs into different aggregate modules to simulate real-life workflow clustering and provide various module parallelism, as shown in Figs. 11 and 12. Fig. 12 is a high-level view of grouped workflow in Fig. 11, where \( w_0 \) and \( w_7 \) are the start and end modules.

Based on the physical resources, we provide three available VM types with sufficient RAM size and disk space for WRF execution and different CPU speeds, as shown in Table 4, and assign pricing linearly according to the processing power. Since the VMs in the experiment have the same disk space, they have a nearly identical VM startup time. Hence, we can always launch the VMs in advance before actually running workflow modules.

7.3.3 Performance Comparison

We store the entire workflow as well as the input data on each VM image. To obtain the execution time of each module, we create a VM of each available VM type and run each module on it for multiple times. We observe that the module execution time remains stable on the same type of VM, and construct the execution time matrix \( T_{E} \) in unit of seconds in Table 5 and execution cost matrix \( C_{E} \) in Table 6.

From \( T_{E} \) and the pricing of VM types, we calculate \( C_{min} = 125.6 \) and \( C_{max} = 196.2 \), and then run four scheduling algorithms, i.e. Critical-Greedy, HBCS, ScaleStar, and optimal solution, with 20 budget levels at an equal interval, indexed from 1 to 20, within the range of \( (C_{min}, C_{max}) \). Since adjacent modules along an execution path in the workflow always have execution precedence constraints (i.e. execution dependencies), their execution times do not overlap. Therefore, if such adjacent modules are mapped to the same type of VM, in the workflow experiment, all of them can be deployed on the same VM for execution but one at a time. This way, we are able to save the cost of instance initialization for a new VM. The schedules generated by these algorithms and their corresponding MED measurements with different budget levels are tabulated in Table 7 and further plotted in Fig. 13 for comparison. These performance
measurements show that the proposed CG algorithm consistently outperforms the other two heuristic algorithms in comparison in all the test cases and approaches the optimum results in most cases.

8 CONCLUSION

We developed a prototype generic workflow system by leveraging existing workflow technologies and formulated a budget-constrained workflow scheduling problem for delay minimization in cloud computing environments, referred to as MED-CC, which was proved to be NP-complete and non-appoximable. We proposed a heuristic algorithm, Critical-Greedy, to MED-CC and demonstrated its performance superiority over existing methods through extensive simulations and real-life workflow experiments using the prototype workflow system in a cloud testbed.

It would be of our future interest to refine and generalize the mathematical models to achieve a higher level of accuracy for workflow execution time measurement in real-world cloud environments. We also plan to incorporate the cost of inter-cloud data movement into workflow scheduling in multi-cloud environments. Such data transfer may pose some restrictions on VM provisioning as we need to consider VMs’ connectivity to support inter-module communication based on the available bandwidth in the cloud infrastructure.

ACKNOWLEDGMENTS

This research was sponsored by U.S. Department of Energy’s Office of Science under Grant No. DE-SC0010641 with University of Memphis and Grant DE-AC02-98CH10886 with Brookhaven National Laboratory (BNL). This research is also sponsored by BNL via Laboratory Directed Research and Development (LDRD-13-017).

REFERENCES


Chase Qishi Wu received the BS degree in remote sensing from Zhejiang University, P.R. China, in 1985, the MS degree in geomatics from Purdue University in 2000, and the PhD degree in computer science from Louisiana State University in 2003. He was a research fellow in the Computer Science and Mathematics Division at Oak Ridge National Laboratory during 2003-2006. He is currently an associate professor with the Department of Computer Science, University of Memphis. His research interests include big data, distributed computing, computer networks, scientific visualization, and cyber security. He is a member of the IEEE.

Xiangyu Lin received the BEng degree in electronics and information engineering from Huazhong University of Science and Technology in 2010. She is currently a doctoral student in the Department of Computer Science, University of Memphis, and works in the High-Performance Networking and Computing Group. Her research interests include workflow optimization, cloud computing, and parallel and distributed computing. She is a student member of the IEEE.

Dantong Yu received the BS degree in computer science from Beijing University, China, in 1995, and the PhD degree in computer science from State University of New York at Buffalo, in 2001. He joined Brookhaven National Lab (BNL) in 2001, and coordinated the Grid computing group at BNL. He currently leads the computer science group at the Computational Science Center. His research interests include high-speed network, information retrieval, data mining, and high performance data storage. He has published papers in leading technical journal and conferences. He has been leading BNL’s software-defined networking (SDN) effort since 2004. He is a member of the IEEE.

Wei Xu received the BS and MS degree in computer science from Zhejiang University, China, in 2004, and the PhD degree in computer science from Stony Brook University in 2012. She joined Brookhaven National Lab (BNL) in 2013 and is currently an Advanced Applications engineer in the Computational Science Center collaborating with HEP X-ray Nanoprobe (HXN) and Full Field X-ray Imaging (FFI) beamline groups of National Synchrotron Light Source II (NSLS-II). Her research interests include medical imaging, tomography, visualization, visual analytics, high performance computing with GPGPUs and multi-core clusters, imaging processing, machine learning, and workflow systems. She is a member of the IEEE.

Li Li received the BS degree in physics from Beijing University, China, in 2005, and the PhD degree in materials science and engineering from Columbia University in 2011. He joined Brookhaven National Lab (BNL) in 2011 as a research associate. His research areas include x-ray/neutron dynamical diffraction, x-ray/neutron powder diffraction and imaging reconstruction techniques. He is also interested in scientific software development and high performance computing. He is a member of the IEEE.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.