Workflow Scheduling in Multi-Tenant Cloud Computing Environments

Bhaskar Prasad Rimal, *Graduate Student Member, IEEE,* and Martin Maier, *Senior Member, IEEE*

Abstract—Multi-tenancy is one of the key features of cloud computing, which provides scalability and economic benefits to the end-users and service providers by sharing the same cloud platform and its underlying infrastructure with the isolation of shared network and compute resources. However, resource management in the context of multi-tenant cloud computing is becoming one of the most complex task due to the inherent heterogeneity and resource isolation. This paper proposes a novel cloud-based workflow scheduling (CWSA) policy for compute-intensive workflow applications in multi-tenant cloud computing environments, which helps minimize the overall workflow completion time, tardiness, cost of execution of the workflows, and utilize idle resources of cloud effectively. The proposed algorithm is compared with the state-of-the-art algorithms, i.e., First Come First Served (FCFS), EASY Backfilling, and Minimum Completion Time (MCT) scheduling policies to evaluate the performance. Further, a proof-of-concept experiment of real-world scientific workflow applications is performed to demonstrate the scalability of the CWSA, which verifies the effectiveness of the proposed solution. The simulation results show that the proposed scheduling policy improves the workflow performance and outperforms the aforementioned alternative scheduling policies under typical deployment scenarios.

Index Terms—Cloud computing, direct acyclic graph, multi-tenancy, resource management, scientific workflow applications.

1 Introduction

Cloud computing is one of the most promising contemporary technologies, on which the research community has recently embarked [1], [2]. It has been emerging as a powerful way to transform the IT and Telcos industries in order to build and deploy custom services and applications, e.g., healthcare, and scientific computations. "Cloud computing is a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet [3]."

Multi-tenancy is one of the key features of cloud computing. In a conventional single-tenant cloud architecture, providers offer a dedicated cloud service (instance of application and underlying infrastructure) to the tenants (customers), where no data is intermingled with other tenants. From the service provider's point of view, this model does not provide scalable cloud services and economics of scale. On the other hand, in multi-tenant cloud computing, infrastructures, applications, and database are shared among all tenants. At the downside, tenants may not be able to customize their use of cloud services in order to fit their specific needs. Moreover, multi-tenancy may be seen differently from a cloud service model perspective. For instance, as for Infrastructure-as-a-Service (IaaS), tenants have the ability to provision computing, storing, and networking resources. Thus, an IaaS provider must allow tenants for virtualization and resource sharing to achieve multi-tenancy. The virtual machine (VM) level multi-tenancy provides the following benefits: a) increased utilization of hardware resources and ease of maintenance (e.g., avoiding dedicated installation per tenant); b) scalability/elasticity, i.e., dynamic workload of large numbers of tenants is handled by scaling up/down resources; c) economics of scale, e.g., reduced service delivery cost for end-users and service providers by sharing the same instance of infrastructure based on an abstraction of isolation between tenants' data and VMs. In Software-as-a-Service (SaaS), a single instance of hosted applications is used by multiple tenants simultaneously, e.g., Force.com [4].

Even though multi-tenancy allows cloud service providers to better utilize computing resources, supporting the development of more flexible services based on economy of scale, and reducing infrastructural costs, how to effectively realize this is a fundamental question. For instance, multi-tenant cloud computing poses unique challenges such as scalability, resource provisioning (e.g., meeting the demand of large volumes of tenants per resource) and customization (per-tenant service customization) [5], [6]. In addition, multi-tenant applications need to be dynamic in nature, or polymorphic, to fulfill the individual expectations of various tenants and their users [4]. This paper investigates a resource management framework that consists of both architecture and scheduling policies with regards to multi-tenancy issues, especially scalability and shared resources for scheduling compute-intensive workflow applications. Further, a resource management framework is developed whose objective is to separate resource management policies from the control mechanisms required to implement them. Note that there are several detrimental consequences for both service providers and tenants such as unpredictable application performance, limited cloud applicability, and inefficiencies in datacenters and revenue [7]. However, these issues are beyond the scope of this paper.

Workflow scheduling is a process of mapping and managing the execution of inter-dependent tasks on distributed

• *B. P. Rimal and M. Maier are with the Optical Zeitgeist Laboratory, Institut National de la Recherche Scientifique (INRS), Montréal, QC, H5A 1K6 Canada. Email:{bhaskar.rimal, maier}@emt.inrs.ca.*

resources. It allocates suitable resources to workflow tasks such that the execution can be completed to satisfy objective functions imposed by users. The proper and efficient scheduling can have significant impact on the performance of the workflow system. In general, scheduling tasks for distributed services was shown to be an NP-hard problem [8]. Thus, there is no optimal solution within polynomial time. Heuristic algorithms were widely developed in order to achieve near optimal solutions. However, there is not any particular resource management framework for scheduling workflow in multi-tenant cloud computing environments.

This paper explores the idea of workflow scheduling in the context of multi-tenant cloud computing environments. In a multi-tenant cloud computing environment, designing a separate service layer for workflows on the top of IaaS or SaaS, giving rise to *Workflow-as-a-Service (WaaS)*. That provides fast turn-around times of the submitted tasks by means of automaticity of the centralized submission interface and scheduler. Given the vital importance of multi-tenancy (e.g., isolation and customization aspects in shared infrastructures), this paper proposes a resource management framework that consists of a novel four-layered architecture of workflow scheduling system and scheduling policy. The introduced architecture and CWSA scheduling policy provide multi-tenancy by unifying abstractions – workflows, resource, and control mechanism (scheduling) – that enable logically centralized policies. Further, the architecture enables a resource abstraction that unifies arbitrary resources, such as storage, network, CPU, and pools, enabling resource-agnostic policies. Moreover, the envisioned architecture provides a service and management environment to enable multiple tenants to run their compute-intensive workflow applications on a shared cloud infrastructure while taking advantage of the elasticity and pay-as-you-go billing model of cloud computing.

To deal with the resource management in the envisioned architecture, this paper proposes a novel cloud-based workflow scheduling (CWSA) policy for compute-intensive workflow applications. CWSA enforces the decisions of resource scheduling policies centrally without requiring explicit coordination. Indeed, CWSA takes advantage of the gaps between scheduled tasks. These idle periods can be used to schedule other tasks, thereby reducing the overall makespan. Further, CWSA helps minimize the overall workflow completion time, cost of execution of workflows, tardiness, and utilize idle resources of cloud effectively. Importantly, CWSA utilizes computational resources properly by reducing idle time of cloud resource nodes. In addition, CWSA policy exhibits less context switching and thus outperforms the state-of-the-art scheduling policies such as First Come First Served (FCFS), EASY (Extensible Argonne Scheduling system) Backfilling, and Minimum Completion Time (MCT). Furthermore, to evaluate the scalability performance of the proposed algorithm, a proof-of-concept experiment with real-world scientific workflow applications such as biology application (e.g., SIPHT) and earthquake science application (e.g., CyberShake) is performed, which verifies the effectiveness of the proposed solution.

The main contributions of this paper are as follows. First, from *an architectural perspective*, we propose a novel four-layered architecture of workflow scheduling system in a multi-tenant cloud computing environment, which represents a cost-effective solution for executing compute-intensive workflow applications. Second, from *a resource management perspective*, we propose a novel cloud workflow scheduling algorithm (CWSA) to deal with both structured and unstructured workflow scheduling in multi-tenant cloud computing environments, which maximizes cloud resource usage, minimizes the expected makespan (the overall completion time of the workflow), and also minimizes the scheduling execution time, workflow execution costs, and total expected tardiness (delay penalty, if a task in workflow is completed after its due-time). Third, we perform comprehensive *simulation-based studies* of the proposed scheduling policy and evaluate different set of performance metrics, including statistical analysis, to demonstrate the robustness of the proposed scheduling policy. We then compare the performance of the CWSA policy with the FCFS, EASY Backfilling, and MCT scheduling policies. Note that there is no consensus on widely accepted metrics to measure robustness. Thus, in this work, we propose to use the makespan standard deviation, skewness of the makespan, and the 99^{th} percentile distribution of the makespan to compare robustness metrics. Fourth, from *a proof-of-concept demonstration perspective*, CWSA scheduling policy is further implemented in real world scientific complex workflow applications (e.g., SIPHT and CyberShake – popular benchmark that have been widely used in workflow studies) to demonstrate the scalability and effectiveness of the proposed solution.

The remainder of the paper is structured as follows. Section 2 describes cloud workflow in a nutshell. In Section 3, we briefly review related work. Section 4 presents the proposed reference architecture of workflow scheduling and describes the proposed algorithm in greater detail. Section 5 describes the salient features of our implementation, followed by experimental results obtained by means of simulations, and provides a comparison of its performance with that of different alternative scheduling policies. Further, a proof-of-concept demonstration is presented and finally, we conclude and discuss future work in Section 6.

2 Workflow

Cloud workflow applications (e.g., high-end scalable media processing, scientific workflow applications, data analytics) can be defined as collections of resource intensive activities processed in a well defined order to achieve a certain goal, which could be executed in geographically distributed heterogeneous resources. Based on the structural complexity, cloud workflow applications can be sub-divided into two groups: balance structured, e.g., Electron Micrograph ANalysis (EMAN) – a Bio-imaging workflow and unbalanced structured, e.g., SIPHT and Montage workflows. The balanced structured workflow contains several parallel pipelines that require the same types of service to process different data sets. Conversely, unbalanced structured workflows are complex and require different services [9].

Scientific workflow applications (see Fig. 1) have a complex structure and require heterogeneous services. The execution of such workflow applications faces several challenges such as scalability, quality of service (QoS), ensured reproducibility, computing resources, data storage,

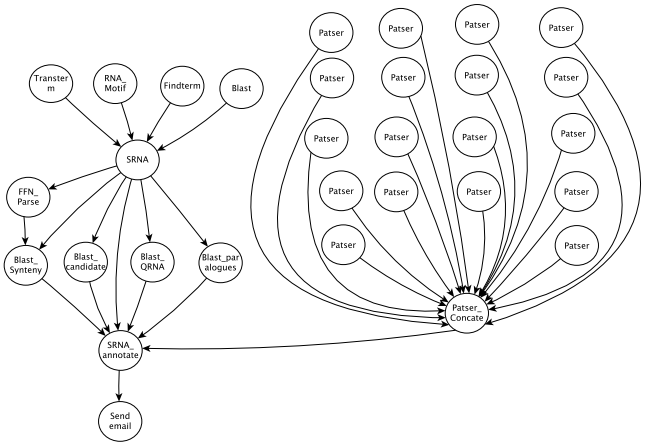

Fig. 1: SIPHT [12] – an example of scientific workflow.

as well as heterogeneous and distributed data management [10], [11]. Such workflow applications require a distributed high-end computing environment, leading to the recently emerged cloud computing paradigm. Cloud computing offers several distinct features for workflow applications compared to other computing environments such as: a) dynamic resources allocated/de-allocated, which enables the workflow to elastically scale up/down the cloud resources; b) a large shared pool of resources with scalable processing capability, storage, and network resources to execute compute-intensive workflow applications; c) cloud workflow applications can share application instances and their underlying cloud resources between multiple tenants, which allows a cloud service provider to maximize cloud resource utilization and thereby reduce service costs per tenant; d) manages secure isolation of such resources between multiple tenants through its multi-tenancy feature.

3 Related Work

The problem of scheduling tasks on multiple resources has been extensively studied in parallel and distributed systems, cluster and grid computing, and in recent years to a lesser extent in cloud computing. The methodology adopted varies according to the characteristics of the workload (e.g., batch workload or online workload, large/medium/small size, and frequency), characteristics of resources (e.g., physical/virtual resources, number of nodes, and networks), performance metrics of interest, and scheduling based on multi-agent systems, e.g., [13]. Further, most of these algorithms considered a stable infrastructure. We briefly review prior work on various aspects in the following.

Heuristic Algorithm: Several studies considered heuristics, e.g., list scheduling, clustering, and task duplication. Examples of list scheduling include Heterogeneous Earliest-Finish-Time (HEFT) [14] and Fast Critical Path (FCP) [15] scheduling for a single workflow. A list scheduling heuristic combined with multi-objective optimization was proposed in [16] for scheduling workflow in grids and clouds. Well-known examples of task duplication based algorithms include [17] and task duplication-based scheduling algorithm for network of heterogeneous systems (TANH) [18]. Clustering heuristics, e.g., [19] for task clustering and CASS-II for task clustering with no duplication [20] were studied in heterogeneous systems.

Cost- and time-based heuristic algorithm was proposed in [21] to minimize the execution, communication costs, and overall completion time for scheduling workflow tasks in cloud environments. The Whittle's index-based heuristic scheduling was proposed in [22] for executing parallel tasks on opportunistically available cloud resources. The opportunistic scheduling allocates low-priority tasks to intermittently available servers to minimize the cost of waiting and migration. A dynamic resource allocation heuristic was proposed in [23] to minimize skewness and improve the utilization of server resources. Notably, in [24], authors have shown that widely-used Best-Fit scheduling algorithm is not throughput-optimal. The work in [25] addressed the static resource-constrained multi-project scheduling problem (RCMPSP) by considering project and portfolio lateness. However, RCMPSP may not be suitable in the context of multi-tenant cloud environments due to multi-tenant interference, unfairness, as well as variable and unpredictable performance (e.g., throughput). Some tenants may pay for performance isolation and predictability, while other tenants may choose best-effort behavior [26].

Meta-Heuristic Algorithm: A particle swarm optimization (PSO) based scheduling algorithm was proposed in [27], [28] to minimize the execution cost of workflow applications in cloud computing. The market-oriented cloud workflow systems based on genetic algorithm, ant colony optimization, and PSO has proposed in [29]. In [30], authors proposed a pricing model and dynamic scheduling of single tasks in commercial multi-cloud environments and compared their approach with pareto-optimal solutions based on two classical multi-objective evolutionary algorithms, i.e., SPEA2 and NSGA-II.

Note that heuristic based scheduling algorithms fit only a particular type of problem (e.g., a workflow with a simple structure), while the meta-heuristic algorithm provides a general solution method for developing a specific heuristic to fit a particular kind of problem [9]. Most of the heuristic and meta-heuristic based scheduling algorithms were designed and optimized in the context of grid computing environments. Further, meta-heuristic algorithms such as PSO, SPEA2, and NSGA-II are very time consuming and thus not very effective for large workflow applications.

Scientific Workflows Execution: The performance and the cost of execution of scientific workflows in a cloud environment was studied in [31]. This study has shown that most of the resources provided by Amazon EC2 are less powerful for I/O-intensive applications like Montage than Abe (NASA HPC cluster) due to the lack of high-performance parallel file systems. A data locality driven task scheduling algorithm was proposed to improve the system performance on cloud computing environments [32]. However, the algorithm did not implement real-world cloud applications to verify its effectiveness (e.g., scalability, dynamic workload). Similarly, a matrix based k-means clustering strategy for data placement in scientific cloud workflows was presented in [33].

Deadline-aware Scheduling: Dynamic resource provisioning of adaptive applications in cloud environments was studied in [34] based on Q-learning reinforcement guided control theory to maximize the application-specific benefit function within given time and budget constraints for

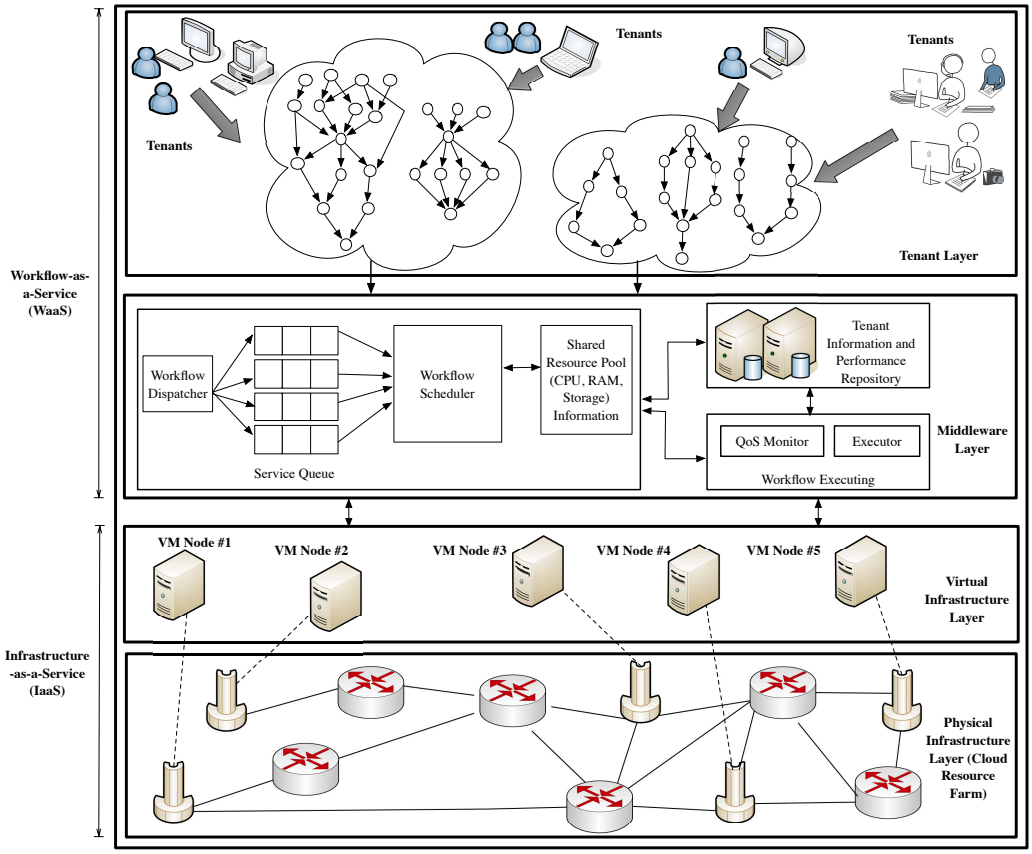

Fig. 2: Proposed reference architecture of workflow scheduling in a multi-tenant cloud environment.

a particular task. VGrADs [35], a virtual grid execution system– was studied for scheduling of a deadline sensitive weather forecast workflow supports. A resubmission heuristic based on the HEFT algorithm [14] was proposed in [36] to meet soft deadlines of scientific workflows in computational grids. A deadline constraint algorithm based on the HEFT was proposed in [37] for scheduling a single workflow instance on an IaaS cloud environment. However, none of these algorithms were designed in the context of multi-tenant cloud environments.

Multi-tenant SaaS Applications: There exist only few studies on multi-tenant SaaS applications, e.g., a cluster-based resource allocation algorithm [38], which includes lazy and pro-active duplications to achieve an improved system performance in a two-tier multi-tenant SaaS schedule architecture. Further, the capacity planning of multi-tenant applications using a method for determining the optimal allocation of application threads to physical nodes was studied in [39]. Recently, a resource allocation model for SaaS applications was proposed in [40]. However, none of these approaches dealt with the compute-intensive workflow scheduling in multi-tenant cloud computing.

Overall, most of the aforementioned studies focused on the scheduling performance of a single workflow. Moreover, these approaches did not consider compute-intensive workflow applications in a multi-tenant cloud computing environment. In fact, there exist no particular scheduling policies to execute compute-intensive workflow applications in multi-tenant cloud environments. Unlike those approaches, in this paper we focus on the minimization of the makespan, cost of execution of the workflows, and tardiness, while maximizing the resource utilization within a given deadline in multi-tenant cloud computing environments.

4 Multi-tenant Cloud Environments for Workflow Scheduling

This section presents the proposed reference architecture of workflow scheduling in a multi-tenant cloud environments and describes the proposed cloud workflow scheduling algorithm in greater detail.

4.1 Architecture

There exists a plethora of workflow management systems, e.g., Pegasus [41]. However, many of their features are optimized for conventional grid and cluster computing to execute single/multiple job(s) or workflow and thus may not be able to obtain most of the key aspects of cloud computing, while such systems suffer from limited resource provisioning. Although there are few works addressing workflow scheduling on clouds, e.g., [33], they were not designed in the context of multi-tenancy. Given the emergence of diverse sets of scientific workflow applications each belonging to different domains, a multi-tenant aware and flexible workflow platform is needed to cost-effectively execute/deploy the workflow applications of multiple tenants. The envisioned architecture enables such workflow applications to share a single infrastructure while taking advantage of the elasticity and pay-as-you-go billing model of cloud computing.

Fig. 2 depicts a four-layered architecture of the proposed workflow scheduling system. The first layer (tenant) consists of workflow creator/composer. The second layer (middleware) consists of workflow dispatcher, service queue,

workflow scheduler, shared pool resource information, tenant information and performance repository, QoS monitor, and executor. The third layer is the virtual infrastructure layer, whereas the fourth layer consists of the physical infrastructure layer. Each layer is briefly described in the following.

Tenant Layer: Each tenant-specific cloud workflow is configured by acquiring the tenant preferences and QoS. Each tenant submits an individual workflow application to the workflow scheduling system. The workflow tasks are scheduled according to the available resources (virtual machines, datacenter, etc.) at the given deadline. The tenants submit the workflow applications according to a uniform or random distribution. According to the users' QoS requirements, the workflow scheduler checks the availability of services and resources and then applies the given scheduling policies to execute these workflow tasks.

Middleware Layer: In order to realize multi-tenancy, a suitable middleware is required to minimize the underlying complexity (e.g., configure, manage, and identify multiple tenants as well as tenant-specific customization of workflow applications). In fact, it decouples the tenant and infrastructure layers. Middleware layer comprises a number of components such as workflow scheduler, QoS monitoring component, and performance repository component. When the tenant workload increases, a pool of identical application instances is created in order to ensure scalability. These components are briefly described in the following.

Workflow Dispatcher: The main functionality of the workflow dispatcher is to aggregate the workload (workflow applications) and dispatch it to the service queue.

Service Queue: The service queue maintains a priority queue for all incoming workflow tasks and distributes them to the workflow scheduler.

Workflow Scheduler: The workflow scheduler is the core component of the middleware layer, which provides several features for storing task information, maintaining up-to-date cloud resource information, resource selection information (matching task requirements to resource space available on the cloud), QoS monitoring information, and performance information. The major functions of this component help select prioritized workflow tasks from the service queue, execute scheduling policies, send provisioning instructions to the virtual infrastructure layer for creating virtual resources and subsequently mapping the workflow tasks to virtual machines (VMs). Further, the workflow scheduler communicates with the performance repository and tenant information component in order to get information about the current status of the cloud resources and tenants, including which VMs are running on which physical machines in the cloud resource farm.

Shared Resource Pool: A resource pool is a logical abstraction that is needed for the flexible management of resources. It provides information about used, limit, available, and shared resources (e.g., CPU, RAM, storage, network, and software licenses).

Tenant Information and Performance Repository: This component stores tenant configuration files. More specifically, it saves configuration and customization metadata of all tenants. It also accumulates all the data related to QoS, e.g., deadline, availability, scalability, etc. Therefore, any changes of the configuration would affect the scope of a particular tenant. Further, the services that are to be composed should be selected based on such a configuration (i.e., tenant information).

QoS Monitoring: This component monitors the QoS data. Further, it oversees the performance of the service/workflow application instance. It reports on whether the service instance on its worker node is underloaded, overloaded, or in normal conditions based on the threshold. Such information is retrieved from the tenant information and performance repository component.

Executor: The executor notifies a task's completion status after finishing it successfully. During the workflow execution, the scheduler is responsible for handling task dependencies such as transferring dependent files. In addition, the middleware layer coordinates the virtual infrastructure layer by distributing workflow tasks to available resources.

Virtual Infrastructure Layer: This layer consists of a number of VM instances running on top of the cloud server farm. It allocates on-demand resources and maps the virtual resources to the physical resources.

Physical Infrastructure Layer: It contains a number of physical resources, i.e., cloud server farm that consists of physical servers for computing, storage, and network. It also provides provisioning and deprovisioning of resources to the virtual infrastructure layer.

4.2 Problem Formulation

4.2.1 Application and Resource Models

A cloud workflow can be formally modeled as a directed acyclic graph (DAG). It consists of computational tasks and transmission tasks. A DAG is a tuple $G = (V, E)$, where $V = \{\tau_i \mid i = 1, \ldots, v\}$ denotes the set of vertices representing tasks with $|V| = v$ and $E = \{e_{ij} \mid (i,j) \in \{1, \ldots, v\} \times \{1, \ldots, v\}, |E| = e$ is the set of communication edges representing precedence relation between two computational tasks. The labels on nodes denote computation costs and the labels on edges represent communication costs. The communication time between tasks in the workflow is determined by various factors, including bandwidth, number of tasks, and volume of data transferred. A critical path (CP) is the longest path in the workflow.

For illustration, an example workflow is shown in Fig. 3, where edge $e(i,j)$ from $(\tau_i - \tau_j)$ means that τ_j needs an intermediate result from τ_i such that $\tau_j \in \text{succ}(\tau_i)$, where $\text{succ}(\tau_i)$ is the set of all intermediate successor tasks of τ_i. A task without immediate predecessors is known as an entry-task and a task without immediate successors as an exit-task. The average communication cost between two tasks τ_i and τ_j is equal to $C(i,j) = \frac{data(i,j)}{B}$, where B denotes the average bandwidth and $data(i,j)$ represents the amount of data required to be transmitted between the τ_i and τ_j. If τ_i and τ_j are assigned to the same resource (i.e., VM), the communication cost is negligible and assumed to be zero. The computation cost is the execution costs of tasks on the resources. It is computed by dividing the total number of instructions required to execute that tasks by the processing capacity of resources (in instructions per second). The service queue is considered large enough to buffer all unprocessed tasks. The IaaS cloud may offer a

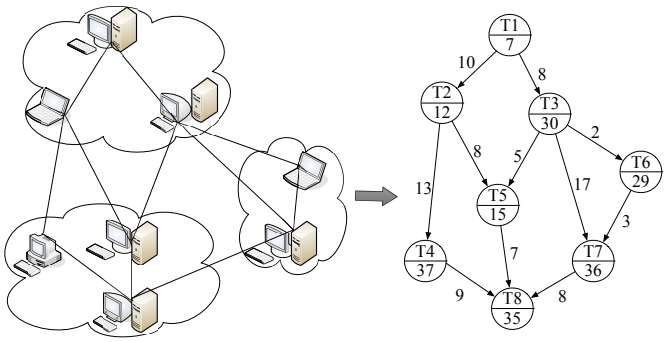

Fig. 3: Representation of a cloud workflow by a direct acyclic graph (DAG).

set of VM instance types with different processing power (CPU, RAM, storage) and prices. When a VM is leased, it needs a boot-up time to be properly initialized and be made available to the tenant (s). Thus, this time is considered in the scheduling and cost evaluation. Since we assume a single cloud datacenter model to execute workflow applications, where physical machines are interconnected by high-bandwidth links. Thus, we do not consider the cost incurred by data transfer. Further, the so-called *pay-per-use* billing model is considered, where partial utilization of the leased VM instance-hour is counted as a full time period.

4.2.2 Problem Definition

In our model, a given set of cloud workflows $\{\omega_l \mid l = 1, \ldots, n\}, \omega_l \in W$, consists of computing-intensive tasks $(\tau_1, \tau_2, \ldots \tau_v)$. The workflows need to be scheduled in the cloud resource farm R_k $(1 \leq k \leq m)$, provided that resource $R_k \in R_\phi$ is available for processing at time $P(t)$ such that $\sum_{R_k \in R_\phi} Q_{\tau_i \omega_j R_k} = 1$, where $Q_{\tau_i \omega_l R_k}$ determines to which resource a certain workflow is scheduled. Recall from Fig. 2 that each tenant's information is stored in the tenant information and performance repository.

$$Q_{\tau_i \omega_l R_k} = \begin{cases} 1 & \text{if task } \tau_i \text{ of workflow } \omega_l \text{ is allocated} \\ & \text{to resource } R_k \\ 0 & \text{otherwise.} \end{cases}$$

During the scheduling process, each tenant submit a service request, i.e., a workflow to the workflow scheduler with the resource requirements given by (ID, t_l, VM_l, D_l), where ID, t_l, VM_l represent the tenant ID, reservation time slot, number of VMs required for ω_l, and associated deadline of each workflow, respectively. Note that tenant ID is very important to create an isolation environment for tenants that separates one tenant context (tenant's information) from another. The objective of the workflow scheduler is to schedule a workflow ω_l to a cloud resource R_k $(1 \leq k \leq m)$ for a given time constraint (task τ_i of the workflow ω_l has to complete its execution before the deadline). No task is scheduled for any resources that are not available yet, i.e., $t_{\tau_i \omega_l} \geq p_k Q_{\tau_i \omega_l R_k}$, whereby $t_{\tau_i \omega_l}$ is the starting time of task τ_i of workflow ω_l and p_k represents the time when the resource becomes available, $t_{\tau_i \omega_l} \geq 0$.

We assume that a child task cannot be executed until all of its parent tasks are completed. In a multi-tenant environment, resources are virtualized and shared among many tenants. Minimizing the completion time and tardiness of the workflow application in such a shared environment poses a challenging problem while taking the resource utilization into account. The proposed Cloud Workflow Scheduling Algorithm (CWSA) aims at improving the performance of the metrics described below in Section 4.2 in more detail.

4.3 Cloud Workflow Scheduling Algorithm (CWSA)

4.3.1 Performance Metrics

We define the following performance metrics that are used to study the efficiency and robustness of our proposed workflow scheduling policy.

Makespan: The makespan or completion time is a measure of the throughput of the system. Clearly, the main design objective of an efficient scheduling policy is to minimize the makespan. The minimum makespan implies a high utilization of computing machines, leading to a higher throughput. The makespan is calculated for each scheduled workflow as the time from submission to completion of task, i.e., denoted by $makespan = SCT$. The workflow makespan C_W as the maximum completion time of all its tasks, is expressed as follows:

$$C_W = \max_{\tau_i \in \omega_l} \{SCT(\tau_i)\}. \tag{1}$$

Further, the optimization rate of the makespan (OMS) is given by:

$$OMS = (MMS - MS_{min})/MMS, \tag{2}$$

where MMS denotes the mean makespan and MS_{min} denotes the minimum makespan.

Tardiness: A delay penalty with weight is charged, if the task τ_i is completed after its due time. The delay penalty of a given task is defined as its tardiness. The tardiness of a workflow task indicates the time span of the completion time exceeding the projected due time in a fine-grained manner. The minimization of mean tardiness has been used as the primary objective in scheduling tasks in order to meet a given task's due time, which can be computed as follows:

$$\overline{td}_i = \sum_{i=1}^{N} [max(C_i - d_i, 0)]/v, \tag{3}$$

where $C_i - d_i \geq 0$ and the maximum tardiness is given by:

$$td_i{}^{max} = \max_{i=1 \ldots v} [max(C_i - d_i, 0)], \tag{4}$$

where C_i, d_i, and v denote the completion time of task τ_i, due time (i.e., sum of arrival time and total processing time of task), and number of tasks in a workflow, respectively.

Laxity: The laxity of a task is the measurement of its urgency. At time t the laxity of task τ_i is $(d_i - t - p)$, where d_i is the task's deadline and p is its remaining computing time requirement. When the laxity of a task is negative, the execution of the task can not meet its deadline. If the laxity of a given task is zero then the execution of the task should start immediately. In case of a positive laxity, the execution of the task can be delayed and thus placed in the queue.

Mean scheduling execution time: This metric accounts for the total time taken by the workflow scheduler to execute workflow tasks using a particular scheduling policy.

Resource utilization rate: It is defined as the percentage of time that a resource is busy. Therefore, a better performance can be achieved only through a higher utilization of resources, while meeting given deadlines and QoS requirements of workflow applications. The average resource usage is computed as follows:

$$\overline{RU} = \frac{\sum_{i=0}^{n-1}(RU_{t_i} * (t_{i+1} - t_i))}{t_n} \quad (5)$$

where,

$$RU_i = \frac{R_{act}}{min(R_{ava}, R_{req})}, \quad (6)$$

where n, R_{act}, R_{ava}, and R_{req} denote the simulation length, number of active resources, number of available resources, and required resources for the workflows, respectively.

Makespan standard deviation (MSD): It is used to measure the robustness of scheduling policies. The MSD is calculated as follows:

$$\sigma_m = \sqrt{\frac{1}{\mathcal{N}-1}\sum_{a=1}^{\mathcal{N}}\left(X_a - \overline{X}\right)^2}, \quad (7)$$

where σ_m, X_a, $\mathcal{N}$, and $\overline{X}$ denote the sample standard deviation of the makespan, data set of the makespan, size of the sample, and average value of the makespan, respectively. Note that smaller values of MSD are more likely to result in a stable system performance.

Skewness of makespan: It measures the symmetry of the makespan distribution and can be calculated by using the adjusted Fisher-Pearson standardized moment coefficient, as follows:

$$\gamma(X) = \frac{\mathcal{N}}{(\mathcal{N}-1)(\mathcal{N}-2)}\sum_{a=1}^{\mathcal{N}}\left(\frac{X_a - \bar{X}}{\sigma_m}\right)^3. \quad (8)$$

A negative value of skewness indicates that a tail can be found on the left-hand side of the makespan distribution, whereas a positive value of skewness means the opposite.

4.3.2 Proposed Algorithm

The pseudocode of our proposed CWSA algorithm is shown in Fig. 4. The algorithm works as follows. Initially, the resource nodes are sorted in descending order based on their computational speeds. The objective to do so is to select the lowest execution cost for ready workflow tasks (lines 3-6). The scheduler checks the task dependency at scheduling time to verify which tasks can be scheduled one after another. This is done through a depth-first search. When the workflow is submitted to the scheduler, the workflow tasks will be inserted into a service queue (lines 6-7). The ready workflow tasks are sorted according to a deadline priority. Next, an appropriate $schedulegap$ is calculated (see also Eq. 9), i.e., a suitable time-slot of resource nodes for every ready workflow task (lines 8-11).

The $schedulegap$ is the time period of an idle CPU. It occurs when the number of currently available CPUs is more powerful with respect to the existing schedule than the number of CPUs requested by the given workflow applications in the given time period. When a new task arrives, a better schedule position is selected according to the current schedule position. For the set of workflows $\omega_l \in W$ with deadlines D_l, $l = 1, 2, \ldots, n$, the necessary and sufficient condition for the feasibility of a workflow schedule with schedule utilization $SU(t)$ is defined as the utilization by the current task invocations at time t, which can be expressed as follows:

$$SU(t) = \sum_{TI_i \in CI(t)} (C_i/T_i) \leq 1, \quad (9)$$

where TI_i is the task invocation, $CI(t)$ denotes the set of current invocations, and C_i represents the worst-case computing time (i.e., largest time between release and termination). Further, T_i denotes the period of workflow task and (C_i/T_i) denotes the fraction of processor time spent on the execution of the workflow tasks. The counter keeps track of the schedulable utilization at run-time. The counter is set to zero $(SU = 0)$ at the initial stage of each schedule gap. When a new workflow task is invoked, it is incremented by (C_i/T_i), i.e., if $SU + C_i/T_i \leq 1$ then the request for schedule is admitted at current time plus its deadline and is decremented by (C_i/T_i), if the deadline of the current workflow task's invocation is reached. The algorithm searches for $schedulegaps$ on every resource.

The workflow scheduler is able to execute different scheduling policies. If the $schedulegap$ is not found, then the scheduler can schedule the workflows using other policies like FIFO, EASY Backfilling, and MCT (lines 12-20). For Instance, if $SchedulingPolicy == 1$ then schedule of the workflow is determined by using the FCFS scheduling policy. With FCFS, incoming workflows are sorted in the scheduler queue in their arriving order. Based on this order, if the required resources are available to execute the first workflow it is immediately scheduled. Otherwise, the workflows wait until the resources become available for them.

If $SchedulingPolicy == 2$ the workflow is scheduled by using the EASY Backfilling scheduling policy. In the EASY Backfilling policy workflows are sorted based on their deadline and are placed in the queue accordingly. The EASY Backfilling policy works similarly to FCFS. However, if the first workflow can not be scheduled due to the lack of required resources, the scheduler calculates the earliest start time based on the running workflow. Subsequently, it makes a reservation for the first workflow. Once the resources become available the workflow is scheduled. If a workflow in the backlog can be executed on time without delaying others, it can be moved forward in the queue and be executed earlier. Thus, ideally resources are backfilled with suitable workflows in order to achieve a higher resource utilization.

In the case of $SchedulingPolicy == 3$, the workflow is scheduled by using the MCT scheduling policy. With MCT, the workflow scheduler assigns the workflows in an arbitrary order to the available cloud resources such that the workflow will have the minimum completion time. Toward this end, the scheduler firstly chooses a workflow arbitrarily from the service queue. Secondly, it finds the cloud resource that gives the minimum completion time for the chosen workflow. Thirdly, the schedule maps the workflow to the chosen cloud resource and removes the workflow from the service queue. Then, the available time of

Algorithm 1 Cloud Workflow Scheduling Algorithm (CWSA).

Require: A Workflow W is defined in a workflow repository. $NewSchedule = 0$.
Ensure: Workflow schedule within the deadline, if the deadline exists.
1: **repeat**
2: $W \leftarrow$ unscheduled workflow in the list
3: **for all** $r_k \in R$ **do**
4: insert r_k into $InitialSchedule(IS)[]$ in descending order based on their computational speed.
5: **end for**
6: **for all** $\omega_l \in W, r_k \in R$ **do**
7: Insert $\forall$ ready workflow $\omega_l \in$ W into service queue then traverse the workflow by using a depth-first search.
8: **for all** $r_k \in R$ **do**
9: $schedulegap \leftarrow$ find $schedulegap(IS)$
10: Calculate a $schedulegap$ as described in Eq.(9).
11: **end for**
12: **if** There does not exist a $schedulegap$ on the resource nodes $r_k \in R$ and $\exists$ $SchedulingPolicy$, $\forall$ available scheduling policies, Select $SchedulingPolicy$ **then**
13: **switch** ($SchedulingPolicy$)
14: **case** 1**:**
15: $NewSchedule \leftarrow$ Schedule workflow task $\tau_i \in$ T on the resource node $r_k \in$ R using the FCFS scheduling policy.
16: **case** 2**:**
17: $NewSchedule \leftarrow$ Schedule workflow task $\tau_i \in$ T on the resource node $r_k \in$ R using the EASY Backfilling scheduling policy.
18: **case** 3**:**
19: $NewSchedule \leftarrow$ Schedule workflow task $\tau_i \in$ T on the resource node $r_k \in$ R using the MCT scheduling policy.
20: **end switch**
21: **else if** ($\exists schedulegap$ on the resource nodes $r_k \in R$ & & case = = 4) **then**
22: Call **function** CWSA()
23: **end if**
24: **end for**
25: **until** all the workflows have been scheduled.

Function 1 Function CWSA ()

1: **function** CWSA ()
2: $NewSchedule \leftarrow InitialSchedule$
3: **if** $totalweight > 0$ (as described in Eq.(10)) **then**
4: **for all** $\omega_l \in W, r_k \in R$ **do**
5: $NewSchedule \leftarrow$ Move workflow tasks into found $schedulegap$. That means in the appropriate scheduling position, i.e., a virtual machine already started.
6: **end for**
7: **else**
8: remove the workflow from schedule
9: **end if**
10: **end function**

the resource is updated. The scheduler repeats this process until all workflows have been scheduled and assigned to the resources.

Afterwards, in the case that a $schedulegap$ is found, the workflow scheduler executes the CWSA scheduling policy (see lines 21-22 and Function 1). All suitable resource nodes are tested whether a suitable $schedulegap$ for new workflow tasks exists in their schedules. At regular intervals, if the task queue is not empty and idle resource(s) exist(s) in the cloud resource pool, the workflow scheduler tries to find a suitable resource for the workflow tasks in the resource pool. In Function 1, line 4, the value of the total weight is checked. Note that the $totalweight$ denotes the highest value of the weight for each assignment of a new workflow task, which is set to the ideal CPU gap according to Eqs. (10-12). Thus, the $totalweight$ can be defined as follows [42]:

$$totalweight = weight_{makespan} + weight_{deadline}, \quad (10)$$

where,

$$weight_{makespan} = \frac{(makespan_{old} - makespan_{new})}{makespan_{old}}, \quad (11)$$

$$weight_{deadline} = \frac{(nondelayed_{new} - nondelayed_{old})}{nondelayed_{old}}, \quad (12)$$

whereby $makespan_{old}$ and $makespan_{new}$ represent the expected makespan of the current schedule of workflow tasks and the makespan of the new schedule of workflow tasks, respectively. Further, $nondelayed_{old}$ and $nondelayed_{new}$ denote the number of workflow tasks executed within the deadline before and after the workflow task assignment, respectively. The proposed algorithm searches for all gaps on every available resource and selects the best gap. If $totalweight > 0$ (see Function 1, line 3), the current mapping is taken as the best schedule. If there is no proper schedule in the beginning of the schedule, the algorithm searches for the gap between the task on the current position, τ_i, and the next task τ_{i+1}. If all the schedule positions are tested and there no better performance has been found, the workflow returns to the initial position. Note that $NewSchedule$ has a lower number of tardy tasks, if the value of $weight_{deadline}$ is positive.

Fig. 4 depicts an illustrative example of scheduling multiple workflows. Two workflows, W_A and W_B, are to be scheduled on two resources, $R1$ and $R2$. For workflow W_A, the scheduler first assigns $\{\tau_{A1}, \tau_{A2}, \tau_{A3}, \tau_{A6}\}$, $\{\tau_{A5}, \tau_{A7}\}$, and $\{\tau_{A4}, \tau_{A8}\}$, respectively. The scheduler schedules $\{\tau_{B1}, \tau_{B2}, \tau_{B4}\}$ on the gap found before τ_{A5}. Similarly, tasks $\{\tau_{B3}, \tau_{B5}, \tau_{B6}\}$ are scheduled on the gap between τ_{A4} and τ_{A8}. Task τ_{A5} starts at time 50, when its predecessor has already finished its execution. The time complexity to find the resource, which has minimum execution time, is $O(mW)$, where W is the number of workflows and m denotes the number of allocated resources.

4.3.3 Advantages of CWSA Scheduling Policy

In the following, we discuss the superiority of CWSA policy over the alternative scheduling schemes under consideration. FCFS uses the time instance when a workflow arrives at the cloud scheduler to define the priority for all requests associated with the workflow. This policy is inefficient as for

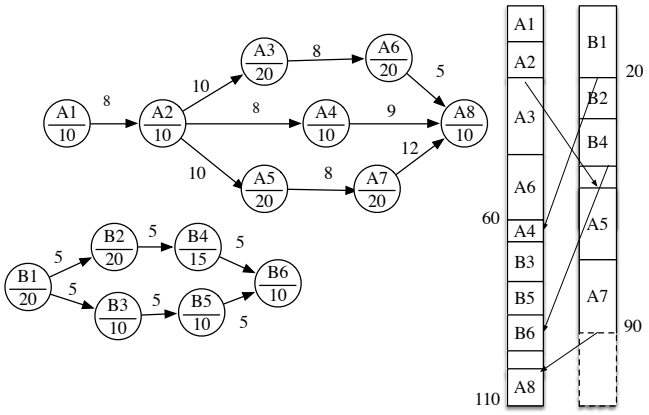

Fig. 4: Strategies for scheduling multiple workflows.

increasing workload many workflows waiting for execution may experience unnecessary idle time of some resources. The EASY Backfilling policy is an example of production batch schedulers, which is similar to FCFS but enables backfilling in order to reduce resource fragmentation. It processes the first task in the queue and reserves the earliest possible time slot. In principle, it is able to run under the FCFS policy, but other tasks in the queue are scheduled opportunistically as nodes become available as long as they do not interfere with the reservation of the first task. However, re-computing the tasks' priorities and sorting them at each scheduling event may cause tasks to be delayed. On the other hand, MCT assigns tasks in an arbitrary order to the worker node based on their minimum completion time. This may cause some tasks to be assigned to resources that do not provide the minimum execution time.

In contrast to these policies, our proposed CWSA policy exhibits less context switching and thus outperforms the former ones. Context switching corresponds to the time period needed for switching between two tasks, i.e., bringing a waiting task into execution and sending an execution task into terminate/waiting state. If the total time of execution of all tasks is assumed to be M, then the context switching time equals $M-$[sum of all tasks (waiting time $+$ execution time)]. We executed each scheduling policy and recorded the context switching information with our script using Linux system call $schedule()$; *schedule()* calls *context-switch*, which is responsible for switching from one task to another one when the new task has been selected by *schedule().context-switch()*. Alternatively, lmbench and Perf could be used to trace context switching information.

The main advantage of the CWSA scheduling policy is its ability to allocate resources at a higher speed for increased schedule lengths of workflows. If shorter schedule lengths are required, it assigns the workflows to the resources at lower speed. CWSA takes advantage of the gaps between scheduled tasks. These idle periods can be used to schedule other tasks, thereby reducing the overall makespan. Further, the execution time of CWSA is shorter than that of the other considered policies, as discussed next.

TABLE 1
Resource types and prices used (based on on-demand instances offered by Amazon EC2, US East (N. Virginia)).

Type	Core	ECU[1]	RAM(GB)	Storage(GB)	Price($/hr)
m1.small	1	1	1.7	1×160	0.044
m1.medium	1	2	3.75	1×410	0.087
m1.large	2	4	7.5	2×420	0.175
m1.xlarge	4	8	15	2×840	0.35

TABLE 2
Workflow parameters and default values.

Parameters	Value
Total number of workflow	1-20
Number of node per workflow	3,000
Bandwidth	100-1000 Mbit/s
Baud rate	10,000 Kbit/s

5 Implementation and Performance Evaluation

This section describes the experimental setup and simulation results. Further, a proof-of-concept experiment is presented to validate our solution using real-world scientific workflow applications, e.g., SIPHT and CyberShake.

5.1 Experimental Setup

The performance of the proposed scheduling policy is thoroughly evaluated using a discrete event cloud simulator based on CloudSim framework [43]. CloudSim supports the modeling and simulations of large scale cloud computing environments on a single computing node, including service brokers, resource provisioning, datacenters, and allocation policies. Interested readers may refer to [43] for further information about CloudSim. We extended CloudSim to support the components shown in the proposed architecture in Fig. 2 and scheduling algorithms. Since workloads have different characteristics, no single elasticity (auto-scaling, i.e., resource over-provisioning and under-provisioning) algorithm is suitable for all workloads. There are mainly reactive (rule-based) and predictive (proactive) auto-scaling techniques classified in the literature. In this work, we implemented rule-based auto-scaling in the QoS monitor component of Fig. 2. Note that most of the cloud service providers (e.g., Amazon EC2 and Rightscale) also apply rule-based mechanisms to scale up/down VMs. For instance, rules like: monitor CPU utilization (U) every 2 min, **[scale-up]** If $U \geq 70\%$ for 10 min, then add 1 VM of small size, wait 4 consecutive 1 min intervals; **[scale-down]** If $U \leq 30\%$ for 12 min, remove 1 VM of small size, wait 5 min consecutive 1 min intervals. For VM reconfiguration, we used an approach similar to the rule-based approach described in [44], we designed rules for auto-scaling VMs. The behavior of auto-scaling can be controlled by changing the configuration file, where rules are defined. The rules specify the upper and lower bounds of the number of VMs

1. One EC2 Compute Unit (ECU) is defined as the CPU power of a 1.0-1.2 GHz of a 2007 Opteron or 2007 Xeon processor, as specified in Amazon EC2 documentation. At the peak performance, one ECU equals 4.4 gigaflops per second (GFLOPS) (see [28] and references herein).

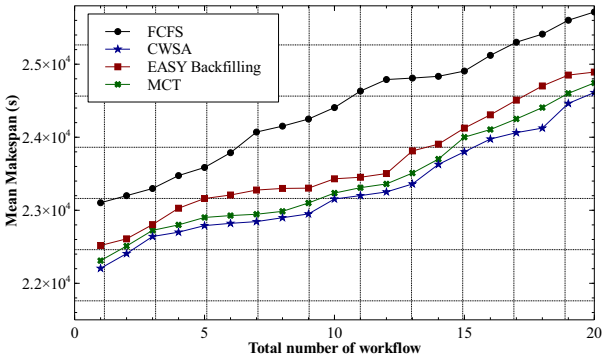

Fig. 5: Mean makespan vs. total number of workflow.

and the conditions to triggers scaling. The infrastructure level (physical and virtual) of Fig. 2 is modeled by the core layer representing the original CloudSim datacenter, which encapsulates sets of computing hosts that can be homogeneous/heterogeneous with respect to the configuration of their hardware, i.e., CPU cores, storage, memory, and bandwidth. A single datacenter is considered in our cloud model and the datacenter is assumed to have sufficient resources. The VMs/cloud resources are modeled Amazon AWS EC2[2] standard instance types and the parameters relevant for the experiment are shown in Table 1. Each newly provisioned VM needs several minutes to be booted-up. Therefore, a boot-up time of 97 seconds is considered for each instance as in [45].

Similar to the deadline assignment in [46], the deadlines are calculated by identifying the smallest amount of time required to execute a single workflow $[D_{min} = minCP(\omega_i)]$, which is the length of the critical path for the workflow with the shortest critical path and the time required to execute all the workflows equals $[D_{max} = \sum CP(\omega_i)]$. The range $[D_{min}, D_{max}]$ is then divided into equal intervals. We assume that each workflow has a D_{max}, in $2CP$, $3CP$ and $4CP$, where CP is the execution time of tasks in the critical path of the DAG at the best available resource.

We used a VM billing interval of 1 hour (i.e., cost per instance per hour). Thus, the usage is rounded up to the nearest hour and any partial hours are counted as full hours (e.g., 1.1 hours is rounded up to 2 hours). For illustration, the pricing is designed based on Amazon's `m1.small` instance type of EC2 US East region (i.e, 1.7 GB of memory, 1 virtual core, 160 Gb of instance storage, and instance price of $0.044 per hour). Other simulation parameters and their default values are listed in Table 2.

5.2 Simulation Results

This section presents the simulation results and discusses the obtained findings. We focus on finding the effective makespan, execution time, resource utilization, and total tardiness.

5.2.1 Makespan

This experiment compares the makespan of different scheduling policies. The mean makespan is depicted in Fig. 5. They are admitted to the workflow scheduler, which utilizes the CPU gap to minimize idle time and thus improve throughput. The highest makespan improvement was obtained for CWSA. On average, CWSA policy is 57% better than FCFS, 30% better than EASY Backfilling, and 16% better than MCT. Our proposed CWSA policy is able to efficiently utilize the resources, thereby enabling that more tasks are completed in a shorter time. The trend of the figure indicates that the makespan is slightly increasing for a growing number of workflows. In fact, with FCFS the makespan increases significantly, while with CWSA the increase is less pronounced compared with the FCFS, EASY Backfilling, and MCT policies.

2. Amazon Elastic Compute Cloud (Amazon EC2), https://aws.amazon.com/ec2.

5.2.2 Skewness of makespan

The skewness of makespan measures the degree of asymmetry in the makespan data set. We calculated the skewness of makespan and list the obtained results in Table 3. We observe that the makespan has an asymmetric distribution. The workloads are skewed, i.e., they experience abnormally high and abnormally low values as they always change, which in turn may degrade the performance.

TABLE 3
Average makespan, skewness of the makespan and standard deviation of the makespan for different scheduling policies.

Scheduling policies	Average makespan (s)	Skewness of the makespan	Makespan standard deviation
FCFS	24422.62	-0.1297	819.50
CWSA	23294.57	0.4475	686.67
EASY Backfilling	23634.90	0.3884	724.20
MCT	23421.51	0.4509	714.68

5.2.3 Network Impact

Next, let us evaluate the network impact of the scheduling policies. To do so, we calculate the average makespan, skewness of the makespan, and standard deviation of the makespan for different scheduling policies. These values are listed in Table 3. We observe that the average makespan time is shorter with CWSA and longer with FCFS. The higher value of the standard deviation stipulates that the communication costs vary significantly among the tasks of the workflow.

5.2.4 Performance distribution using cumulative distribution function (CDF)

Next, we study the distribution of the performance variation for different scheduling policies using the CDF of makespan. The CDF captures the entire performance distribution variation. Fig. 6 depicts the CDF of the makespan for the FCFS, CWSA, EASY Backfilling, and MCT policies. As we can see, CWSA outperforms the other policies in terms of makespan and it represents the most reliable scheduling policy, i.e., the distribution of makespan time of the CWSA policy tends to have a shorter makespan than the others, as shown in Fig. 6. The FCFS and EASY Backfilling policies, however, have a steep tail. Conversely, the FCFS policy has 25% of all makespans being over 24900 seconds.

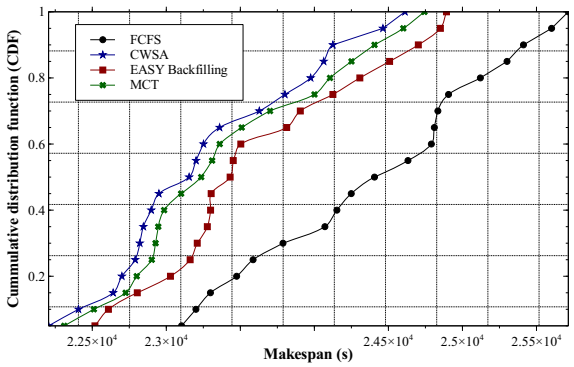

Fig. 6: Cumulative distribution function (CDF) of makespan for different scheduling policies.

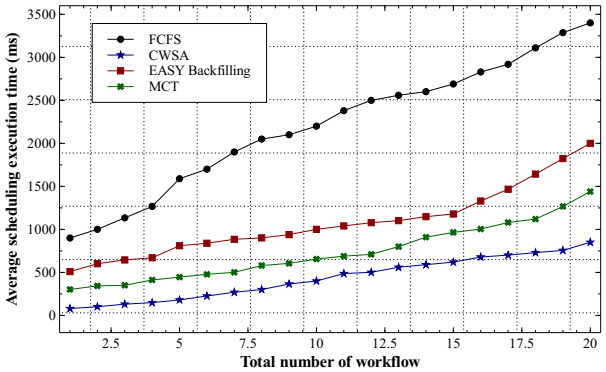

Fig. 7: Average execution time of scheduling policies.

5.2.5 Execution Time

This experiment evaluates the scalability of scheduling policies. The average execution time to execute the scheduling policies averaged over 100 runs with number of workflows. The performance result of each scheduling policy with 20 workflows is depicted in Fig. 7. The execution time of the CWSA policy is smaller than that of the FCFS, EASY Backfilling, and MCT policies. On average, the CWSA scheduling policy is 91% faster than FCFS, 82% faster than EASY Backfilling, and 70% faster than MCT. The execution time appears to grow linearly with the number of workflows. We observe that in the case of CWSA policy, the execution time is lesser and the scalability is better with the increasing number of workflows among others as it efficiently schedules the workflow. Note that the time complexity measures the amount of time required to execute an algorithm, which is given by the upper bound of the amount of workflow tasks performed.

5.2.6 Resource Usage

In this experiment, we evaluate the resource usage. Fig. 8 depicts the percentage of resource utilization of each scheduling policy. A resource node can be idle when it does not support all the hardware/software requirements required by a queued task. The CWSA policy exhibits a better resource utilization. On average, in the case of the CWSA scheduling policy, the resource utilization is 19% better than that of FCFS, 9% better than that of EASY Backfilling, and 7% better than that of MCT. This is due to the fact that the CWSA policy is able to schedule the task in such a way that it keeps the system resources always busy and efficiently utilizes the system's computational

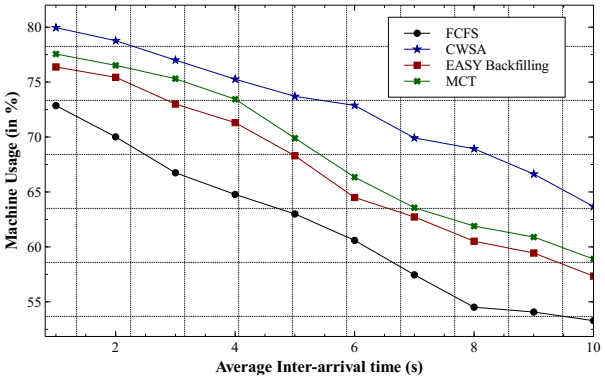

Fig. 8: Resource utilization.

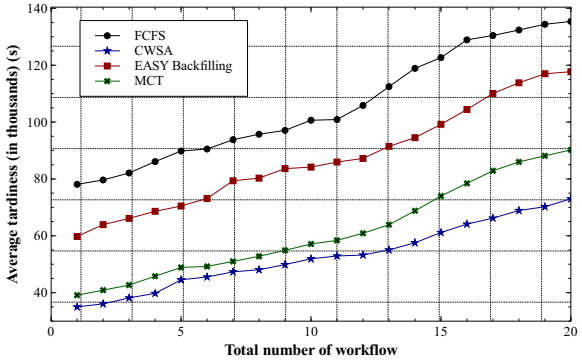

Fig. 9: Average tardiness vs. number of workflows.

resources. Further, the CWSA policy utilizes free resources by executing unscheduled tasks in advance in order to minimize costs. This indicates that maximizing the resource utilization is an effective solution to minimize computing units via an imposed deadline.

5.2.7 Tardiness

This experiment evaluates the tardiness of the workflows. Fig. 9 shows the average tardiness for different scheduling policies. In comparison with FCFS, EASY Backfilling, and MCT scheduling policies, the average tardiness values are smaller in the case of CWSA policy. On average, the CWSA policy improves the tardiness by 48% compared to FCFS, 37% compared to EASY Backfilling, and 10% compared to MCT policies. The average tardiness depends on the number of workflows to be executed on the available machines. The trend of the graph shows that tardiness slightly increases for an increasing number of workloads. This clearly indicates that the total tardiness strongly depends on the number of available instances of virtual machines and workload. Intuitively, as the load increases, more workflows are subjected to become tardy and may also experience a longer period of tardiness. Such a situation may cause a time-critical workflow application to have an inopportune behavior. However, notice that since the CWSA scheduling policy has a better resource utilization (see Fig. 8), more tasks are able to be completed in a shorter time (as execution time is faster than that of others, see Fig. 7). Moreover, CWSA policy is able to utilize the schedule gap more effectively and therefore exhibits the smallest tardiness.

In overall, the proposed scheduling policy has shown its effectiveness to improve the makespan and tardiness. On the other hand, the execution time of CWSA scheduling

policy is notably smaller than that of other scheduling policies. The overall cloud resource utilization is important from a resource owner's point of view to optimize his or her profit. As illustrated in Fig. 8, a better resource utilization is achieved with the proposed CWSA policy.

5.3 Proof-of-Concept Experiment

This section presents the proof-of-concept experiment. We evaluate the performance of the proposed CWSA scheduling policy with real-world scientific workflow applications, such as SIPHT (a typical structure of SIPHT workflow is illustrated in Fig. 1) and CyberShake workflows. Note that these workflows are widely considered as a benchmark in the literature. Importantly, they represent a wide range of application domains and have a diverse resource requirements. For instance, SIPHT workflow, from the bioinformatics project at Harvard, is used to automate the process of searching for sRNA encoding-genes for all bacterial replicons in the National Center for Biotechnology Information (NCBI) database [12]. CyberShake workflow is used the Southern California Earthquake Center to characterize earthquake hazards by generating synthetic seismograms. CyberShake can be also classified as a data-intensive workflow with large memory and CPU requirements (see Table 4). Interested readers may refer to [47] for further details on these workflows.

5.3.1 Experimental Setup

In this experiment, we used the experimental setup, as described in Section 5.1. The additional parameters used in the experiment such as the characteristics and categorization of the considered scientific workflow applications are shown in Table 4. The medium and large size of workflow applications are more compute-intensive. The structure of these workflows comprise several components such as pipeline, data distribution, data aggregation, and data redistribution. The analysis and resource management for such applications are complex due to the nature of the unstructured (asymmetric) workflow hierarchy and the fact that they cover a wide range of application domains and have intensive resource requirements. We have generated these workflows with the Pegaus workflow system [47]. It generates the DAX (Directed Acyclic Graph in XML format) of these workflow applications for a given number of tasks. The DAX file for workflow contains a list of tasks and the dependencies between them as well as the computation time, and input/output data size of each task.

5.3.2 Results

We have run each scheduling policy 100 times for each workflow application. We evaluated the performance using the 99^{th} percentile distribution of the makespan to represent statistical measures of the scheduling policies for the SIPHT and CyberShake workflow applications. The 99^{th} percentile distribution better characterizes the makespan distribution. Figs. 10 and 11 compare the 99^{th} percentile makespan of the different scheduling policies for the SIPHT and CyberShake workflow across a range of application sizes, respectively. Both figures show that at small workloads, i.e., the size of tasks is small, there exists no significant difference of

TABLE 4
Characteristics and categorization of scientific workflow applications.

Tenant Workload Type	Workflow Name	Number of Tasks	Type (I/O Read/Write (GB), Peak Memory (MB), CPU (hours))
Small	CyberShake	30, 50	High, High, High
	SIPHT	30, 60	Low, Medium, Low
Medium	CyberShake	100	High, High, High
	SIPHT	100	Low, Medium, Low
Large	CyberShake	1000	High, High, High
	SIPHT	1000	Low, Medium, Low

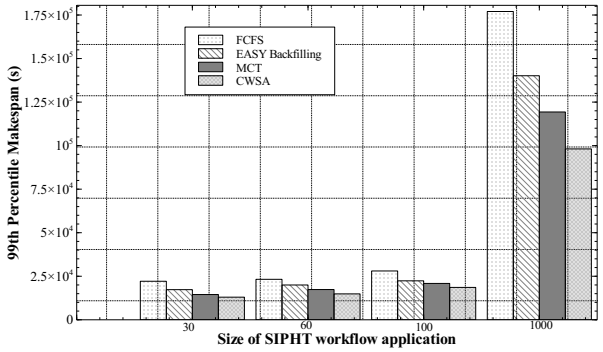

Fig. 10: 99^{th} percentile makespan of different scheduling policies for the SIPHT workflow application.

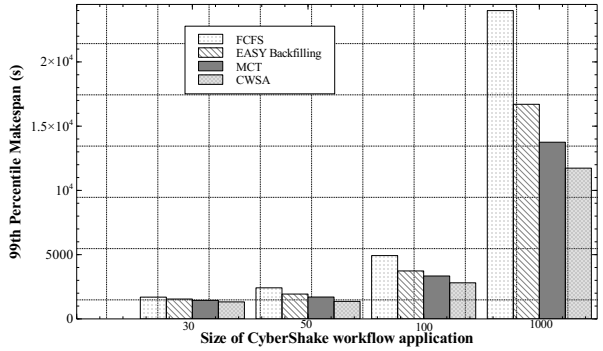

Fig. 11: 99^{th} percentile makespan of different scheduling policies for the CyberShake workflow application.

the makespan between the scheduling policies. This is because when the number of small tasks of CyberShake and SIPHT decreases, the number of allocated computation instances also decreases. As the size of workflow application increases, the makespan quickly approaches the 99^{th} percentile value. In both cases, CWSA performs much better than the other policies for large sizes of workflow. Overall, the CWSA scheduling policy outperforms the other policies for both workflow applications.

5.3.3 Cost Evaluation

The total cost of the execution of a workflow is defined as the product of the total execution time (also includes VM overhead/boot-up time) of a workflow and per hour cost of an instance type. The fractional consumption hours are rounded up. Note that the cost of the execution of workflows not only depends on the scheduling policy but also on the choice of VM instance types (see also Table 1) and the

TABLE 5
Average cost ($) of scientific workflow execution.

Workflow		Scheduling policies			
		FCFS	EASY	MTC	CWSA
CyberShake	small	22.73	22.74	11.37	11.36
	medium	45.46	45.46	22.73	22.73
	large	136.36	113.64	91.94	90.91
SIPHT	small	159.09	113.64	91.91	90.90
	medium	181.82	159.09	136.40	136.36
	large	1136.36	886.36	772.73	636.36

financial budge of a tenant. That means cheaper resources may be attractive, even though they might degrade the performance. As a result, the overall costs may be higher. Note also that the charge of instances is not necessarily proportional to its computing power[3].

The average execution cost obtained for each scientific workflow is shown in Table 5. From the results, we observe that the proposed CWSA scheduling performs better than others in terms of cost by generating much cheaper schedules. The CWSA algorithm shows a considerably lower cost for CyberShake application while having a lower makespan. On the other hand, as expected, SIPHT is a little costly than CyberShake in every workload type. However, SIPHT large workflow experienced significant performance gains compared to other scheduling policies. This shows that the cost of the execution of workflow varies depending on the application and on the size of the workflow as well as the structure of the workflow. Overall, we noticed that the proposed solution is not only a cost-effective but also a faster solution for the considered configurations. It is worthwhile noting that even though we considered an on-demand billing model, for the long-term usage, reserved VM instances are much cheaper than on-demand model (see also Amazon EC2 pricing).

6 Conclusion and Outlook

Cloud computing has been widely recognized as an essential computing paradigm to execute compute- and data-intensive business process workflow (e.g., media processing, analytics pipelines, orchestration of services, coordinating resources, people, information, and systems) and scientific workflow applications for processing of large sets of scientific data, as witnessed by the recent work on Amazon SWF (Simple Workflow Service).

In this paper, we introduced a four-layered workflow scheduling system. A novel CWSA scheduling policy was proposed for scheduling workflow applications in a multi-tenant cloud computing environment. An analysis of different performance metrics was carried out. An extensive simulations was performed to evaluate the performance of the proposed scheduling policy. The performance of the CWSA was then compared with different scheduling policies to highlight the performance and robustness of the proposed solution. The obtained results show that our CWSA outperforms other scheduling policies. Importantly, CWSA was shown to utilize computational resources properly by reducing idle time of cloud resource nodes. Further, we conducted proof-of-concept experiments by employing real-world scientific workflow applications. The proof-of-concept experiment indicates that the proposed CWSA scheduling policy offers significant improvements for larger workflow applications. Importantly, a key lesson learned from this study is that multi-tenancy helps improve the utilization of resources.

Although we have demonstrated the advantages of multi-tenant cloud environments for scheduling workflow applications, there are several potential directions for future work, including the development of a complex model of scheduling policies by considering resource failures and complex reservation scenarios for multi-tier application scaling, where scaling may affect different applications. For the future, we intend to further investigate the optimization of the CWSA scheduling and apply it in the context of mobile cloud computing.

3. Amazon EC2 Pricing: https://aws.amazon.com/ec2/pricing/

Acknowledgments

This research was supported by the Fonds de recherche du Québec - Nature et Technologies (FRQNT) MERIT Doctoral Research Scholarship Program.

References

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Jan. 2009.

[2] B. P. Rimal and E. Choi, "A service-oriented taxonomical spectrum, cloudy challenges and opportunities of cloud computing," *Int. J. Commun. Syst.*, vol. 25, no. 6, pp. 796–819, June 2012.

[3] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc., IEEE Grid Comput. Environments Wksp*, Nov. 2008, pp. 1–10.

[4] C. D. Weissman and S. Bobrowski, "The design of the force.com multitenant internet application development platform," in *Proc., ACM SIGMOD*, June 2009, pp. 889–896.

[5] B. P. Rimal and M. A. El-Refaey, "A framework of scientific workflow management systems for multi-tenant cloud orchestration environment," in *Proc., IEEE WETICE*, June 2010, pp. 88–93.

[6] Y. H. Z. H. W. C. J. Guo, W. Sun and B. Gao, "A framework for native multi-tenancy application development and management," in *Proc., IEEE CEC/EEE*, July 2007, pp. 551–558.

[7] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc., ACM SIGCOMM*, Aug. 2011, pp. 242–253.

[8] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Trans. Software Eng.*, vol. 15, no. 11, pp. 1427–1436, Nov. 1989.

[9] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing," in *Metaheuristics for Scheduling in Distrib. Comput. Environments*, 2008, vol. 146, pp. 173–214.

[10] Y. Zhao, X. Fei, I. Raicu, and S. Lu, "Opportunities and challenges in running scientific workflows on the cloud," in *Proc., IEEE Cyber-Enabled Distrib. Comput. and Knowledge Discovery*, Oct. 2011, pp. 455–462.

[11] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examining the challenges of scientific workflows," *Computer*, vol. 40, no. 12, pp. 24–32, Dec. 2007.

[12] J. Livny, H. Teonadi, M. Livny, and M. K. Waldor, "High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas," *PLoS ONE*, vol. 3, no. 9, p. e3197, Sept. 2008.

[13] F. S. Hsieh and J. B. Lin, "A dynamic scheme for scheduling complex tasks in manufacturing systems based on collaboration of agents," *Applied Intelligence*, vol. 41, no. 2, pp. 366–382, Sept. 2014.

[14] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel and Distrib. Sys.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[15] A. Rădulescu and A. J. C. van Gemund, "On the complexity of list scheduling algorithms for distributed-memory systems," in *Proc., ACM Supercomput.*, June 1999, pp. 68–75.

[16] H. M. Fard, R. Prodan, J. J. D. Barrionuevo, and T. Fahringer, "A multi-objective approach for workflow scheduling in heterogeneous environments," in *Proc., IEEE/ACM CCGrid*, May 2012, pp. 300–309.

[17] S. Darbha and D. P. Agrawal, "Optimal scheduling algorithm for distributed-memory machines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 1, pp. 87–95, Jan. 1998.

[18] R. Bajaj and D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *IEEE Trans. Parallel and Distrib. Sys.*, vol. 15, no. 2, pp. 107–118, Feb. 2004.

[19] A. Gerasoulis and T. Yang, "A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors," *J. Parallel and Distrib. Comput.*, vol. 16, no. 4, pp. 276 – 291, Dec. 1992.

[20] J.-C. Liou and M. A. Palis, "An efficient task clustering heuristic for scheduling DAGs on multiprocessors," in *Proc., Resource Management, Symp. of Parallel and Distrib. Processing*, 1996, pp. 152–156.

[21] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan, "Bi-criteria workflow tasks allocation and scheduling in cloud computing environments," in *Proc., IEEE CLOUD*, June 2012, pp. 638–645.

[22] T. He, S. Chen, H. Kim, L. Tong, and K.-W. Lee, "Scheduling parallel tasks onto opportunistically available cloud resources," in *Proc., IEEE CLOUD*, June 2012, pp. 180–187.

[23] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel and Distrib. Sys.*, vol. 24, no. 6, pp. 1107–1117, June 2013.

[24] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *Proc., IEEE INFOCOM*, Mar. 2012, pp. 702–710.

[25] T. R. Browning and A. A. Yassine, "Resource-constrained multi-project scheduling: Priority rule performance revisited," *Int. Journal of Production Economics*, vol. 126, no. 2, pp. 212–228, Mar. 2010.

[26] D. Shue, M. J. Freedman, and A. Shaikh, "Performance isolation and fairness for multi-tenant cloud storage," in *Proc., USENIX OSDI*, Oct. 2012, pp. 349–362.

[27] S. Pandey, L. Wu, S. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Proc., IEEE Advanced Information Networking and Applications*, Apr. 2010, pp. 400–407.

[28] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, Apr. 2014.

[29] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy incloud workflow systems," *J. Supercomputing*, vol. 63, no. 1, pp. 256–293, Jan. 2013.

[30] H. M. Fard, R. Prodan, and T. Fahringer, "A truthful dynamic workflow scheduling mechanism for commercial multicloud environments," *IEEE Trans Parallel and Distrib. Syst.*, vol. 24, no. 6, pp. 1203–1212, June 2013.

[31] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. Berman, and P. Maechling, "Scientific workflow applications on amazon ec2," in *Proc., IEEE E-Science Wksp*, Dec. 2009, pp. 59–66.

[32] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, "Bar: An efficient data locality driven task scheduling algorithm for cloud computing," in *Proc., IEEE/ACM CCGrid*, May 2011, pp. 295–304.

[33] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Gener. Comput. Syst.*, vol. 26, no. 8, pp. 1200–1214, Oct. 2010.

[34] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Trans. Services Comput.*, vol. 5, no. 4, pp. 497–511, Oct./Dec. 2012.

[35] L. Ramakrishnan, C. Koelbel, Y. suk Kee, R. Wolski, D. Nurmi, D. Gannon, G. Obertelli, A. YarKhan, A. Mandal, T. Huang, K. Thyagaraja, and D. Zagorodnov, "Vgrads: enabling e-science workflows on grids and clouds with fault tolerance," in *Proc., IEEE High Performance Comput. Networking, Storage and Analysis*, Nov. 2009, pp. 1–12.

[36] K. Plankensteiner and R. Prodan, "Meeting soft deadlines in scientific workflows using resubmission impact," *IEEE Trans. Parallel and Distrib. Sys.*, vol. 23, no. 5, pp. 890–901, May 2012.

[37] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 158–169, Jan. 2013.

[38] W. Tsai, X. Sun, Q. Shao, and G. Qi, "Two-tier multi-tenancy scaling and load balancing," in *Proc., IEEE ICEBE*, Nov. 2010, pp. 484–489.

[39] T. Kwok and A. Mohindra, "Resource calculations with constraints, and placement of tenants and instances for multi-tenant saas applications," in *Proc., Int. Conf. on Service-Oriented Comput.*, Dec. 2008, pp. 633–648.

[40] E. Javier, M. Arturo, J. Guillermo, M. Martn, R. Ral, and C. David, "A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures," *Future Gener. Compt. Syst.*, vol. 29, no. 1, pp. 273–286, Jan. 2013.

[41] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, "Mapping abstract complex workflows onto grid environments," *J. Grid Comput.*, vol. 1, no. 1, pp. 25–39, 2003.

[42] D. Klusek, H. Rudov, R. Baraglia, M. Pasquali, and G. Capannini, "Comparison of multi-criteria scheduling techniques," in *Grid Computing*. Springer US, 2008, pp. 173–184.

[43] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[44] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc., IEEE/ACM SC*, Nov. 2011, pp. 1–12.

[45] ——, "A performance study on the vm startup time in the cloud," in *Proc., IEEE CLOUD*, June 2012, pp. 423–430.

[46] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proc., IEEE High Performance Comput., Networking, Storage and Analysis*, Nov. 2012, pp. 1–11.

[47] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Gener. Compt. Syst.*, vol. 29, no. 3, pp. 682–692, Mar. 2013.

Bhaskar Prasad Rimal received the M.Sc. degree in Information Systems from Kookmin University, Seoul, Korea. He is currently pursuing the Ph.D. degree in telecommunications at the Optical Zeitgeist Laboratory, Institut National de la Recherche Scientifique (INRS), Montréal, Québec, Canada. Mr. Rimal is the recipient of the Doctoral Research Scholarship from the Québec Merit Scholarship Program for foreign students of Fonds de Recherche du Québec-Nature et Technologies (FRQNT), the Korean Government Information Technology (IT) Fellowship, the Kookmin University IT Scholarship, and the Kookmin Excellence Award as an Excellent Role Model Fellow. His current research interests include cloud computing, mobile-edge computing (MEC), fiber-wireless (FiWi) enhanced networks, Tactile Internet, Internet of Things, and game theory.

Martin Maier is a full professor with the Institut National de la Recherche Scientifique (INRS), Montréal, Canada. He was educated at the Technical University of Berlin, Germany, and received M.Sc. and Ph.D. degrees (both with distinctions) in 1998 and 2003, respectively. In the summer of 2003, he was a Post-Doctoral Fellow at the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA. He was a visiting professor at Stanford University, Stanford, CA, USA, from October 2006 to March 2007. Dr. Maier is a co-recipient of the 2009 IEEE Communications Society Best Tutorial Paper Award and Best Paper Award presented at The International Society of Optical Engineers (SPIE) Photonics East 2000-Terabit Optical Networking Conference. He is the founder and creative director of the Optical Zeitgeist Laboratory (www.zeitgeistlab.ca). He is the author of the book "Optical Switching Networks" (Cambridge University Press, 2008), which was translated into Japanese in 2009 and the lead author of the book "FiWi Access Networks" (Cambridge University Press, 2012).