J Internet Serv Appl (2011) 2:207–227
DOI 10.1007/s13174-011-0032-0

SI: CLOUD COMPUTING

HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds

Luiz Fernando Bittencourt ·
Edmundo Roberto Mauro Madeira

Received: 11 November 2010 / Accepted: 30 June 2011 / Published online: 3 August 2011

Abstract Workflows have been used to represent a variety of applications involving high processing and storage demands. As a solution to supply this necessity, the cloud computing paradigm has emerged as an on-demand resources provider. While public clouds charge users in a per-use basis, private clouds are owned by users and can be utilized with no charge. When a public cloud and a private cloud are merged, we have what we call a *hybrid cloud*. In a hybrid cloud, the user has elasticity provided by public cloud resources that can be aggregated to the private resources pool as necessary. One question faced by the users in such systems is: Which are the best resources to request from a public cloud based on the current demand and on resources costs? In this paper we deal with this problem, presenting HCOC: The Hybrid Cloud Optimized Cost scheduling algorithm. HCOC decides which resources should be leased from the public cloud and aggregated to the private cloud to provide sufficient processing power to execute a workflow within a given execution time. We present extensive experimental and simulation results which show that HCOC can reduce costs while achieving the established desired execution time.

Keywords Workflow · Scheduling · DAG · Cloud computing

L.F. Bittencourt (✉) · E.R.M. Madeira
Institute of Computing, University of Campinas, Av. Albert Einstein, 1251 Cidade Universitária, 13083-852 Campinas, São Paulo, Brazil
e-mail: bit@ic.unicamp.br

E.R.M. Madeira
e-mail: edmundo@ic.unicamp.br

1 Introduction

Cloud computing is nowadays being used to deliver on-demand storage and processing power. This environment allows the leasing of resources to improve the locally available computational capacity, supplying new computing resources when necessary. In a cloud, the user accesses computing resources as general utilities that can be leased and released [36]. The main benefits to the cloud users is the avoidance of up-front investment, the lowering of their operating cost, the maintenance cost reduction, and the scalability provided on demand. These cloud features provide *elasticity* to the user's computing environment, being able to adapt the computer system to the user needs.

Virtualization [31] is the process of presenting a logical grouping or subset of computing resources so that they can be accessed in abstract ways with benefits over the original configuration. The virtualization software abstracts the hardware by creating an interface to virtual machines (VMs), which represents virtualized resources such as CPUs, physical memory, network connections, and peripherals. Each virtual machine alone is an isolated execution environment independent from the others. Each VM can have its own operating system, applications, and network services. Virtualization allows server consolidation, hardware normalization, and application isolation in the same physical machine. Virtualization technologies, such as Xen [2], are an important part of a cloud computing environment, since the cloud provider can dynamically offer different hardware and software configurations to the cloud user. With this, the user can select a machine configuration and manage this machine with full privileges without interfering in cloud machines available to the others.

In the cloud computing paradigm, details are abstracted from the users. They do not need knowledge of, expertise in, or control over the technology infrastructure about the cloud they are using. It typically involves the provision of dynamically scalable and often virtualized resources as a service over the Internet [1]. The cloud computing characteristics are on-demand self-service, ubiquitous network access, independent resource location (reliability), rapid elasticity (scalability), and pay-per-use. The cloud computing allows the use of Service Oriented Computing (SOC) standards, permitting users to establish links between services, organizing them as workflows instead of building only traditional applications using programming languages.

Cloud computing delivers three defined models: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). In SaaS the consumer uses an application but does not control the host environment. Google Apps[1] and Salesforce.com[2] are examples of this model. In PaaS, the consumers use a hosting environment for their applications. The Google App Engine[3] and Amazon Web Services[4] are PaaS examples. In this model the platform is typically an application framework. In IaaS the consumer uses computing resources such as processing power and storage. In this model the consumer can control the environment including deployment of applications. Amazon Elastic Compute Cloud,[5] Globus Nimbus,[6] and Eucalyptus [24] are good examples of this model.

In terms of resources availability, we can classify IaaS clouds in three different types:

- *Public clouds*: Resource providers offer computing resources as services in a pay-per-use basis, leasing the use of machines to the user during the requested time.
- *Private clouds* or *internal clouds*: Clouds with resources that can be accessed and used by individuals inside an organization, having similarities with data farms or private grids.
- *Hybrid clouds*: Bring together public and private clouds, resulting in a combination of control over performance and security with elasticity.

The on-demand computing offered by the cloud allows the use of private systems (computers, clusters, and grids), aggregating the cloud resources as users need. However, this hybrid approach results in a system with new demands, notably in resource management.

Designing a hybrid cloud requires carefully determining the best split between public and private cloud components [36]. One of the problems to face when determining that is how to split a workflow, which is composed of dependent tasks, to execute in private resources and in public resources. This problem involves tasks execution times on heterogeneous resources, data transmission times over heterogeneous links, as well as monetary costs in the use of charged resources from public clouds. In this paper we introduce HCOC: the Hybrid Cloud Optimized Cost scheduling algorithm. HCOC decides whether a task from a workflow is to be executed in a private resource or in a public charged resource. We assume that the user stipulates a maximum desired execution time (or deadline) $\mathcal{D}$ for the workflow, and our proposed algorithm tries to optimize the monetary execution costs while maintaining the execution time lower than $\mathcal{D}$. Therefore, the objectives of our algorithm are: (i) minimize the monetary costs; (ii) minimize the schedule length (makespan); and (iii) reduce, over the time, the number of schedules with makespan higher than $\mathcal{D}$. With this, this paper contributes in the aggregation of the scheduling algorithm with free/paid multicore processing resources, monetary costs, and deadlines, which are characteristics that can be found in the emerging hybrid cloud systems, leveraging the discussion of the trade-off between local (free and slow) execution and remote (paid and fast) execution in hybrid clouds.

This paper is organized as follows. Section 2 introduces a hybrid cloud infrastructure deployed to perform experiments. In Sect. 3 we present the scheduling problem, providing notations, conceptual background, as well as describing workflow applications used in the simulations. After that we present the HCOC algorithm, in Sect. 4. Experimental and simulation results are discussed in Sect. 5. Works related to the scheduling problem are presented in Sect. 6, while the final remarks and conclusions are in Sect. 7.

2 The hybrid cloud infrastructure

On-demand computing requires services scalability, flexibility, and availability. To supply these requirements, it is important to the infrastructure to offer reconfiguration with the possibility of the deployment of new resources or updating the existing ones without stopping processes in execution. Our infrastructure combines a service-oriented grid, implemented using a dynamic service deployer that makes the underlying grid infrastructure to act as a private cloud, and public clouds.

In a hybrid system composed of a private cloud with the possibility of accessing a public cloud computing infrastructure, the workflow management must supply the requirements in some levels. First, it must provide facilities to the user to make submissions without the need of choosing or indicating the localization of the computational resources to be used. Inside the private cloud boundary, the

[1] http://www.google.com/apps/.

[2] http://www.salesforce.com/.

[3] http://code.google.com/intl/en/appengine/.

[4] http://aws.amazon.com/.

[5] http://aws.amazon.com/ec2/.

[6] http://workspace.globus.org/.

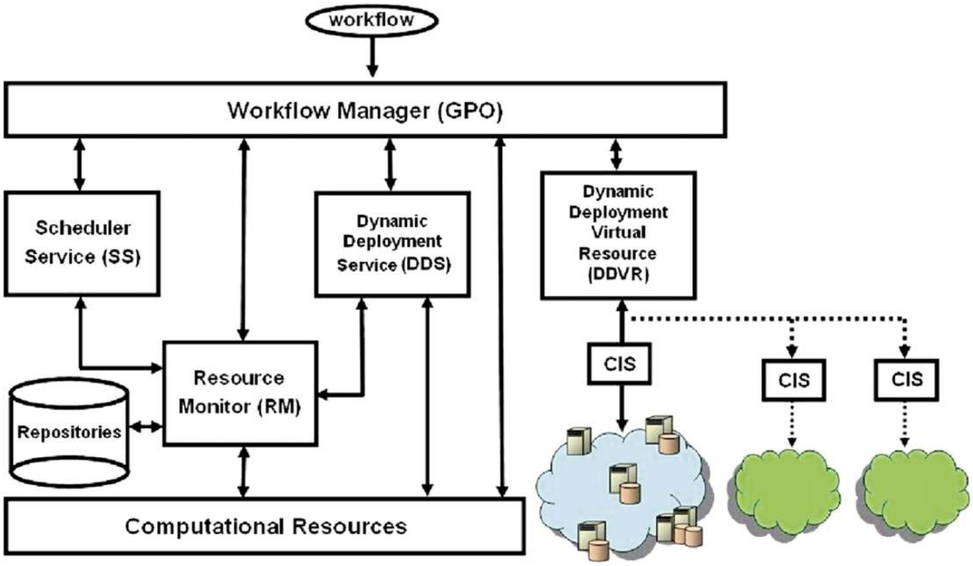


Fig. 1 The hybrid architecture

workflow manager must find the best resources available and, when necessary, it must make the dynamic deployment of services in these resources. On the other hand, inside the public cloud boundary, the infrastructure must be able to interact with the public cloud interfaces to obtain computational resources. After that, it must be able to prepare these resources according to workflow necessities, making the dynamic deployment of services in the resources inside the public cloud. This deployment can be made when local resources are not sufficient to the workflow necessities. This configuration increases the computational power of the private cloud without new infrastructure investment, using the on-demand computing advantages provided by the public cloud. By fulfilling these requirements, a hybrid cloud infrastructure can provide support to the decision making performed by the scheduling algorithm.

In this work we firstly present a small set of experiments using a real testbed, which was used to corroborate the results obtained through simulations shown in Sect. 5.

2.1 The testbed architecture

Similarities between private clouds and grids allow us to use a grid manager called Grid Process Orchestration inside our private cloud. To execute service workflows in hybrid systems we use our hybrid infrastructure [8], which provides dynamic instantiation of services. Thus, the private cloud deployed is capable of executing workflows of tasks as well as workflows of services.

The software infrastructure is composed of a set of services and the Grid Process and Orchestration Language (GPOL) [30], a simple workflow description language. The infrastructure provides dynamic service instantiation and deployment, automatic reference coordination (endpoint reference service). The set of services that compose our hybrid cloud infrastructure architecture is shown in Fig. 1. Its main components are a Workflow Manager (GPO), a Scheduler Service (SS), a service for publication of services (Dynamic Deployment Service—DDS), a service for dynamic publication of virtual resources in the cloud (Dynamic Deployment of Virtual Resources—DDVR), and repositories used by these components.

In a typical execution in our private cloud, the user submits a task workflow or a service workflow to the Grid Process Orchestration (GPO) [30], the workflow manager in the system. The GPO allows the creation and management of application flows, tasks, and services in grids. Its architecture is based on concepts presented in the OGSA (Globus Toolkit 4 implementation—GT4). GPO analyzes the workflow, consulting the scheduling service (SS), which identifies the best resource available for each service involved. If a new service needs to be published, the GPO makes the new service deployment through the Dynamic Deployment Service (DDS), which publishes the new service and returns its localization (endpoint reference) to GPO. Using the endpoint reference, the GPO creates the necessary instances for executing the workflow. When the execution is terminated, the GPO can automatically eliminate the instances created and added to the DDS, or it can leave the services for further use when necessary. If the workflow execution needs more resources, the GPO starts the Dynamic Deployment Virtual Resource (DDVR), which searches for resources in public clouds. The DDVR requests resources to the cloud informing its localization to the GPO. Through the DDVR, the GPO uses the public cloud resources in a similar manner to the use of local resources in the private cloud. It dynamically publishes and instantiates services if necessary, and it monitors the resources through the infrastructure services.

The resource monitor (RM) gathers information regarding computational resources in the hybrid system. It operates in a distributed manner, maintaining one instance in each computational resource. The workflow manager uses the RM to gather knowledge about which resources are available at a given time, and the scheduler uses it to obtain information about the current state of the resources. Besides that, the services repository has information about services currently available in the resources, and it stores the necessary files to the dynamic publication made by the DDS.

2.2 Clouds interconnection

The DDVR is used by the infrastructure when resources from public clouds are needed. It is composed of two groups of services. The first group is the DDVR itself, which communicates with the infrastructure, taking care of the functionalities. The second group, called Cloud Interface Service (CIS), makes the interface with public clouds. The CIS encapsulates particularities of each cloud type (Nimbus, Amazon, Eucalyptus, etc.), allowing transparent access to the public cloud resources in the same way that private cloud resources are accessed. For each resource, one instance of the couple DDVR/CIS is responsible for the binding between the workflow manager and the cloud resource. To use cloud resources, GPO communicates with the DDVR, which communicates with the cloud through CIS, and requests a resource. The cloud returns the localization of the VM obtained and the DDVR can start using it by initiating the GT4 and creating a DDVR instance in the cloud resource.

The DDVR/CIS layout gives flexibility and scalability to the infrastructure, allowing the access to different clouds in an independent and simultaneous manner. Besides that, the transparent access to public cloud resources provided by DDVR/CIS allows, if necessary, the dynamic instantiation of services that are not available in the public cloud.

2.3 The hybrid cloud testbed configuration

Our infrastructure was deployed in a small system as a proof of concept, and it was used to execute a set of experiments. The hybrid cloud deployed was composed of a private cloud (resources shown in Table 1) and a public cloud (resources shown in Table 2). These tables show the available resource on each cloud along with their processing capacities (p_r) and cost of use per time unit. We assume that a private cloud already exists and is currently being used to process users jobs, thus the private cloud upkeep costs already exist. With this, the private cloud is free of charge and operation costs are not taken into consideration, thus the cost for using its resources is considered to be non-significative. In addition, with the number of resources in the private cloud being smaller than the ones available from the public cloud, it is possible to evaluate the algorithm when the private resources are not sufficient to execute the application within the desired deadline.

Table 1 Resources in the private cloud and their capacities

Name	Cores	RAM	p_r per core
Apolo	2	2.5 Gb	1
Cronos	2	4 Gb	2

Table 2 Resources in the public cloud, their capacities, and costs per time unit

Type	Cores	RAM	p_r per core	Price
Y	1	1 Gb	1.5	$0.25
Y	2	1 Gb	1.5	$0.36
Z	1	2 Gb	2	$0.3
Z	2	4 Gb	2	$0.4
Z	3	6 Gb	2	$0.5
Z	4	8 Gb	2	$0.6
Z	8	16 Gb	2	$0.9

3 Workflow scheduling

The Hybrid Cloud Optimized Cost (HCOC) scheduling algorithm has the objective of reducing the makespan to fit a desired execution time or deadline ($\mathcal{D}$) maintaining a reasonable cost. As a consequence, a reduction of the number of $\mathcal{D}$ violations is to be observed over the time. While executing every task of the workflow locally may delay the execution, on the other hand, executing all tasks in the public cloud may result in prohibitive costs. Therefore, the algorithm tries to balance the use of private resources with the ones available from the public cloud in a pay-per-use basis.

3.1 Background and definitions

A workflow is commonly represented by a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes (or tasks), where $t_a \in \mathcal{V}$ is a workflow task with an associated computation cost $w_{t_a} \in \mathbb{R}^+$, and $e_{a,b} \in \mathcal{E}$, $a \neq b$, is a dependency between t_a and t_b with an associated communication cost $c_{a,b} \in \mathbb{R}^+$. An example of workflow represented by a DAG is shown in Fig. 2. It shows a 14-node DAG, where node 1 is the first task to be executed, nodes 2 to 13 can only start their execution after task 1 finishes and sends the data, and node 14 is the last task and can only start its execution after all tasks before it finish and send their data. Nodes in the DAG are labeled with their computation cost (number of instructions, for instance), while edges are labeled with their communication cost (bytes to transmit, for instance).

A private cloud is a set of heterogeneous resources $\mathcal{R} = \{r_1, r_2, \ldots, r_k\}$, with associated processing capacities $p_{r_i} \in \mathbb{R}^+$, connected by network links. Each resource $r_i \in \mathcal{R}$ has a set of links $\mathcal{L}_{r_i} = \{l_{i,r_1}, l_{i,r_2}, \ldots, l_{i,r_m}\}$, $1 \leq m \leq k$, where $l_{i,j} \in \mathbb{R}^+$ is the available bandwidth in the link between

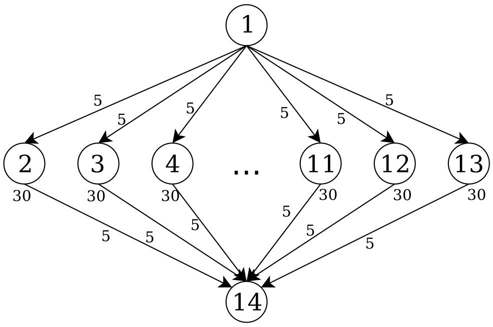


Fig. 2 Example of fork-join DAG with 14 nodes

resources r_i and r_j, with $l_{i,i} = \infty$. A public cloud is a set of charged virtualized resources $\mathcal{U} = \{u_1, u_2, \ldots, u_w\}$, available on demand, with associated processing capacities $p_{u_j} \in \mathbb{R}^+$, connected by network links. Each resource $u_j \in \mathcal{U}$ has a set of links $\mathcal{L}_{u_j} = \{l_{j,u_1}, l_{j,u_2}, \ldots, l_{j,u_q}\}$, $1 \leq q \leq w$, where $l_{j,k} \in \mathbb{R}^+$ is the available bandwidth in the link between resources u_j and u_k, with $l_{j,j} = \infty$. Additionally, each resource $u_j \in \mathcal{U}$ has a price per time unit $price_{u_j}$. A set $\mathcal{S}_u^t \subseteq \mathcal{U}$ is a set of resources leased from the public cloud $\mathcal{U}$ at a given time t.

A hybrid cloud is a set of resources

$$\mathcal{H} = \left(\bigcup_{a=1}^{b} \mathcal{R}_a\right) \cup \left(\bigcup_{s=1}^{t} \mathcal{S}_s\right),$$

with $b \geq 1$ and $t \geq 1$. Each cloud $C_z \in \mathcal{H}$ has a set of links $\mathcal{L}_{C_z} = \{l_{z,1}, l_{z,2}, \ldots, l_{z,y}\}$, $1 \leq y \leq |\mathcal{H}|$, where $l_{z,x} \in \mathbb{R}^+$ is the minimum available bandwidth in the path between C_z and C_x. Therefore, the bandwidth between two resources $h_i \in \mathcal{H}$ and $h_j \in \mathcal{H}$, $i \neq j$, is the minimum bandwidth in the path between h_i and h_j. Note that a public cloud has an unbounded number of resources. Consequently, the size of $\mathcal{H}$ (and $\mathcal{S}$), as a result of the elasticity, is bounded by the number of resources requested by the scheduling algorithm.

Task scheduling is a function $schedule_{DAG} : \mathcal{V} \mapsto \mathcal{H}$, where each task $t \in \mathcal{V}$ is mapped to a resource $r \in \mathcal{H}$.

3.2 Workflow applications

Nowadays many applications are represented by DAGs. Montage [13] is an example of workflow application that can have different sizes. It is an image application that makes mosaics from the sky for astronomy research. Its workflow size depends on the square degree size of the sky to be generated. For example, for a 1 square degree of the sky, a workflow with 232 jobs is executed. For 10 square degrees of the sky a 20,652 jobs workflow is executed, dealing with an amount of data near to 100GB. The full sky is around 400,000 square degrees [12]. Our infrastructure along with the proposed algorithm can handle this kind of application by requesting resources to public clouds when the private resources are not sufficient to execute the workflow.

Our evaluation comprises experiments and simulations with eight DAGs of real world applications (Fig. 3), namely Montage [13], AIRSN [38], CSTEM [14], LIGO-1 [28] and LIGO-2 [23], Chimera-1 and Chimera-2 [4], and the median filter image processing application, represented by the DAG in Fig. 2.

3.3 Initial schedule

When submitting a DAG to be executed, the user may want to have it finished before a certain time. Let $\mathcal{D}_G$ be the deadline (or a desired finish time) of a DAG G. The proposed algorithm makes an initial schedule using the Path Clustering Heuristic (PCH) algorithm [6]. This initial schedule considers only the private resources $r_i \in \mathcal{R}$ to check if they already satisfy the deadline. If the deadline is not satisfied, the algorithm starts the process of deciding which resources it will request to the public cloud. This decision is based on performance, cost, and the number of tasks to be scheduled in the public cloud. To make the initial schedule, the PCH algorithm uses some attributes computed for each DAG node:

- *Computation cost*:

$$w_{i,r} = \frac{instructions}{p_r}$$

$w_{i,r}$ represents the computation cost (execution time) of the node i in the resource r, and p_r is the processing capacity of resource r in instructions per second.

- *Communication cost*:

$$c_{i,j} = \frac{data_{i,j}}{l_{r,p}}$$

$c_{i,j}$ represents the communication cost (time to transfer data) between nodes n_i and n_j using the link l between resources r and p. If $r = p$, then $c_{i,j} = 0$.

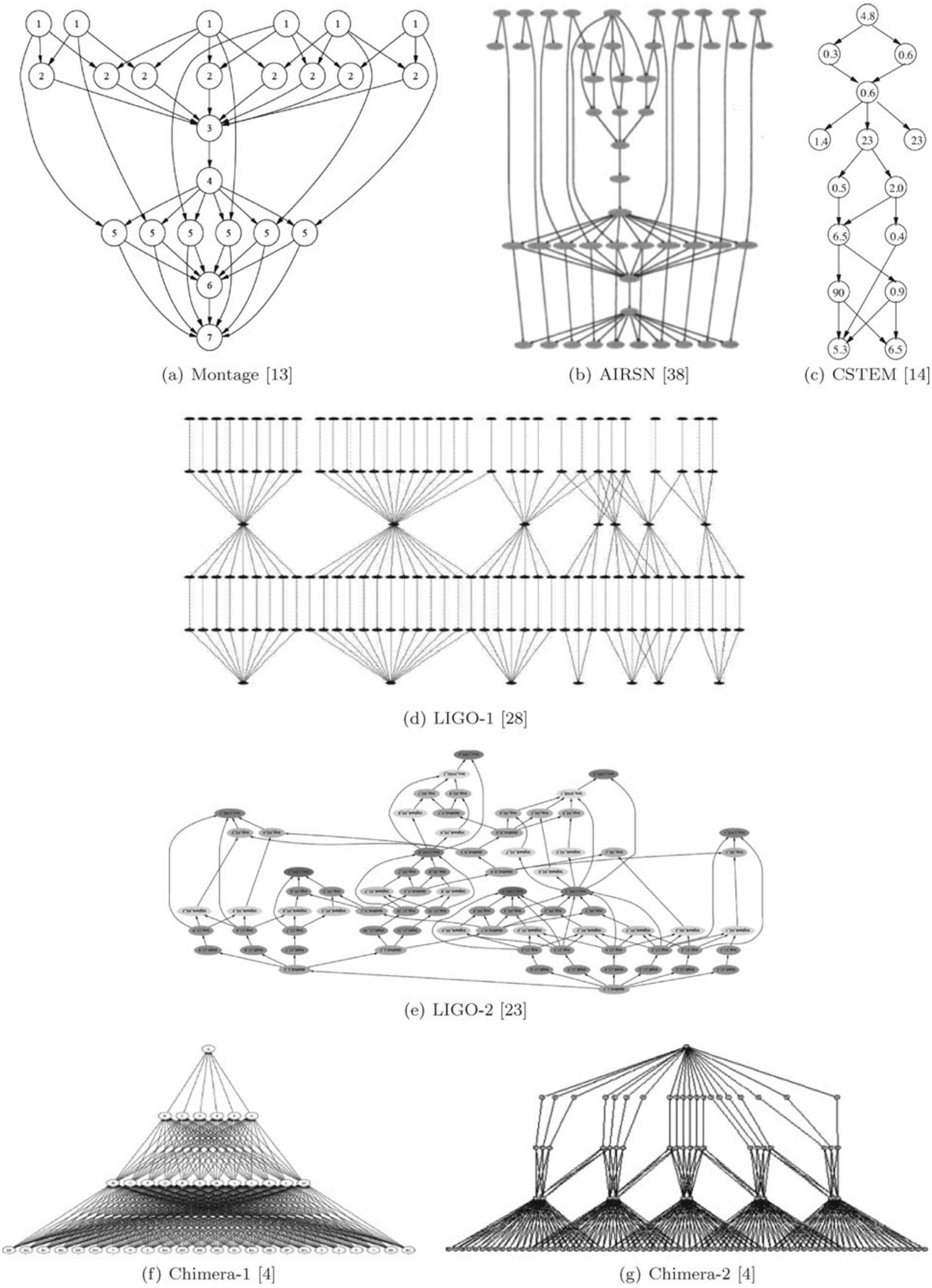
- $suc(n_i)$ is the set of immediate successors of node n_i in the DAG.
- $pred(n_i)$ is the set of immediate predecessors of n_i in the DAG.
- *Priority*:

$$\mathcal{P}_i = \begin{cases} w_{i,r}, & suc(n_i) = \emptyset \\ w_{i,r} + \max_{\forall\, n_j \in\, suc(n_i)} (c_{i,j} + \mathcal{P}_j), & \text{otherwise} \end{cases}$$

$\mathcal{P}_i$ is the priority level of node i at a given time instant during the scheduling process.

- *Earliest start time*:

$$EST(n_i, r_k) = \begin{cases} Time(r_k), & \text{if } i = 1 \\ \max\{Time(r_k), ST_i\}, & \text{otherwise} \end{cases}$$



(a) Montage [13]

(b) AIRSN [38]

(c) CSTEM [14]

(d) LIGO-1 [28]

(e) LIGO-2 [23]

(f) Chimera-1 [4]

(g) Chimera-2 [4]

Fig. 3 Workflow applications used in the evaluation

where $ST_i = \max_{\forall n_h \in pred(n_i)}(EST(n_h, r_k) + w_{h,k} + c_{h,i})$. $EST(n_i, r_k)$ represents the earliest start time possible for node i in resource k at a given scheduling instant. $Time(r_k)$ is the time when resource k is available to execute node i.

- *Estimated finish time*:

$$EFT(n_i, r_k) = EST(n_i, r_k) + w_{i,k}$$

$EFT(n_i, r_k)$ represents the estimated finish time of node i in resource k.

After computing these attributes, the initial scheduling takes place by creating groups of nodes (*clusters of tasks*) that are in the same path in the DAG. Tasks in the same cluster are scheduled in the same resource. To create such clusters, the algorithm uses the priority attribute to select the first task to be added to the first cluster (clustering phase). The first task t_i selected to compose a cluster cls_k is the unscheduled node with the highest priority. It is added to cls_k, then the algorithm starts a depth-first search on the DAG starting on t_i, selecting $t_s \in succ(t_i)$ with the highest $P_s + EST_s$ and adding it to cls_k, until t_s has a non-scheduled predecessor. The resource selected for a given cluster of tasks is the one that minimizes the *EFT* of the cluster. For more details about PCH, please refer to [6].

4 The HCOC scheduling algorithm

The HCOC algorithm is based on the following general steps:

1. Initial schedule: schedule the workflow in the private cloud $\mathcal{R}$;
2. While the makespan is larger than the deadline:
   - Select tasks to reschedule;
   - Select resources from the public clouds to compose the hybrid cloud $\mathcal{H}$;
   - Reschedule the selected tasks in $\mathcal{H}$.

The two main steps of the algorithm are the selection of tasks to reschedule and the selection of resources from the public cloud to compose the hybrid cloud. While the former decides which tasks can have their execution time reduced by using more powerful resources from the public cloud, the latter determines the performance and costs involved in the new schedule.

In this paper we compare three policies to select which tasks will be rescheduled if the desired execution time is violated in the initial schedule:

- *Highest* P_i: select the node n_i which has the highest priority P_i in the scheduled workflow. This selects tasks to reschedule from the beginning of the DAG to the end, and it is the main policy evaluated throughout the paper;
- *Highest* EFT_i: select the node n_i which has the highest estimated finish time EFT_i in the scheduled workflow. This policy selects tasks from the end to the beginning of the DAG;
- *Highest* $P_i + EST_i$: select the node n_i which has the highest priority plus its estimated start time, $P_i + EST_i$, in the scheduled workflow. This is the same policy used by PCH to create clusters of tasks [9].

With the definition of which tasks will be rescheduled, the algorithm decides how many resources to request from the public cloud considering their prices and performance. Such resources will be used to compose the hybrid cloud. Our algorithm takes into account the number of clusters of tasks being rescheduled to select these resources, as well as their cost, number of cores, and performance. The benefits of considering the number of resource cores are twofold: (i) task dependencies costs are suppressed, since communication time between two tasks in the same processor is considered to be null; and (ii) in general, higher number of cores means lower cost per core. Note that this last benefit is only interesting when there are enough tasks to be executed in parallel, making use of most cores rented. This is why our algorithm considers the number of clusters of tasks being rescheduled.

After the initial schedule generated by PCH, the algorithm checks if resources from the public cloud are needed based on the deadline $\mathcal{D}$. If the makespan given by PCH is larger than $\mathcal{D}$, the algorithm selects the node i that is currently scheduled in the private cloud and has the largest $\mathcal{P}_i$ to be scheduled considering the public cloud as well. The algorithm repeats these steps until the deadline is met or a certain number of iterations is reached. This strategy is shown in Algorithm 1.

The second line of Algorithm 1 makes the initial schedule using the PCH and considering only resources in the private cloud. After that, while $\mathcal{D}$ is violated, the algorithm iterates until the deadline is met or the number of iterations is equal to the number of nodes in the DAG (line 3). Inside this iteration, the algorithm selects the node n_i such that $\mathcal{P}_i$ is maximum and n_i is not in the current rescheduling group $\mathcal{T}$ (line 5). Then, in line 7, node n_i is added to the set $\mathcal{T}$, which is composed of all nodes being rescheduled from the private cloud to the public cloud. In line 8, the algorithm computes the number of clusters of nodes in $\mathcal{T}$, which will determine how many resources (or cores) will be requested to the public cloud. After that, an iteration is repeated until the number of selected cores reaches the number of clusters in $\mathcal{T}$, always selecting the public cloud resource i which gives the smallest $\frac{price_i}{num\_cores_i \times p_i}$ with $num\_cores_i <= num\_clusters$. The selected resource is added to the public cloud resources pool $\mathcal{H}$ in line 11. With the resources selected, the algorithm schedules each node $n_i \in \mathcal{T}$ in the resource $r_n \in \mathcal{H}$ that results in the smallest EFT_i.

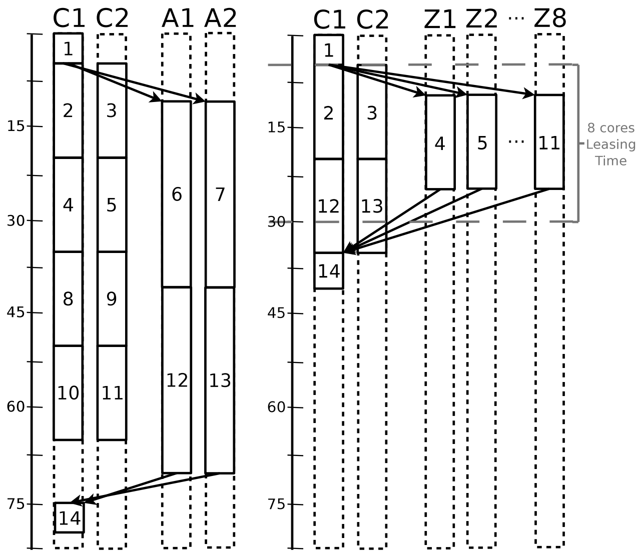
Algorithm 1 HCOC: the Hybrid Cloud Optimized Cost scheduling

```
$\mathcal{R}$ = all resources in the private cloud
Schedule $\mathcal{G}$ in the private cloud using PCH
while $makespan(\mathcal{G}) > \mathcal{D}$ AND $iteration < size(\mathcal{G})$ do
    $iteration = iteration + 1$
    select node $n_i$ such that $\mathcal{P}_i$ is maximum and $n_i \ni \mathcal{T}$
    $\mathcal{H} = \mathcal{R}$
    $\mathcal{T} = \mathcal{T} \cup t$
    $num\_clusters$ = number of clusters of nodes in $\mathcal{T}$
    while $num\_clusters > 0$ do
        Select resource $r_i$ from the public clouds such that $\frac{price_i}{num\_cores_i \times p_i}$ is minimum and $num\_cores_i <= num\_clusters$
        $\mathcal{H} = \mathcal{H} \cup \{r_i\}$
        $num\_clusters = num\_clusters - num\_cores_i$
    end while
    for all $n_i \in \mathcal{T}$ do
        Schedule $n_i$ in $h_j \in \mathcal{H}$ such that $EFT_i$ is minimum
        Recalculate $ESTs$ and $EFTs$
    end for
end while
```

Figure 4 shows an example considering the DAG of Fig. 2 and resources in Tables 1 and 2, with $\mathcal{D} = 45$. The initial schedule considering only the private cloud resources is on the left-hand side of Fig. 4. It allocates nodes in two cores of *Cronos* ($C1$ and $C2$) and two cores of *Apolo* ($A1$ and $A2$), resulting in *makespan* = 80 and *cost* = 0. The schedule in the hybrid system using HCOC is shown on the right-hand side. It uses two cores of *Cronos* and 8 cores of the public cloud ($Z1$ to $Z8$), with *makespan* = 40 and *cost* = $25 \times 0.9 = 22.5$, since the lease time is 25 time units and the leasing of 8 cores in *Zeus* costs \$0.9 per time unit. This schedule is achieved starting with the schedule on the left-hand side, and then rescheduling nodes 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 and 11, in this order.

Besides the private-only and HCOC scheduling, we consider another approach as comparison. We call *greedy* scheduling the approach where both resources from private and public clouds are considered, but resources from the public cloud are selected based solely on their performance no matter their costs. This approach follows the same algorithm as the scheduling in the private cloud only, but considering that any resource in any cloud can be used. For the example of Figure 4, the greedy approach would result in a schedule with *makespan* = 35 and *cost* = $(25 \times 0.3) \times 10 = 75$, using 10 leased resources with 1 core each during 25 time units.

Note that HCOC can be easily adapted to deal with budgets instead of $\mathcal{D}$. In this case, the user just has to stipulate a budget $\mathcal{B}$ to be obeyed, and the outer iteration can be adapted to be executed while the current execution cost is smaller than $\mathcal{B}$. This would also be useful in prepaid systems, where the user pay for resources. Additionally, other heuristics might be used instead of PCH. In fact, in order to evaluate the robustness of the proposed strategies, we also executed simulations of the proposed algorithm with the Heterogeneous Earliest Finish Time algorithm [32].



(a) Private cloud $\mathcal{R}$ (b) Hybrid cloud $\mathcal{H}$

Fig. 4 Example of scheduling in the private and hybrid clouds

5 Evaluation

The evaluation is separated in two parts: first, we present results for experiments conducted in our testbed [9], as described in Sect. 2. After that, we show simulations covering more types of workflows and different resource configurations. The metrics used are the average makespan and average cost of the schedules given by the algorithms, the percentage of times the algorithm did not reach the deadline, and the number of rescheduled tasks. Results show 95% confidence intervals.

5.1 Experimental results

We have set up a testbed, as described in Sect. 2, to start the evaluation of HCOC. In this evaluation scenario we deployed services on each resource to execute an image processing filter (*median filter* [22]). Applying filters on high resolution images can be expensive and may demand some computational power. On the other hand, it can be done parallelizing the filter application by segmenting the image into smaller blocks. Then, the filter is applied in a distributed way, and the resulting blocks are merged into the final resulting image.

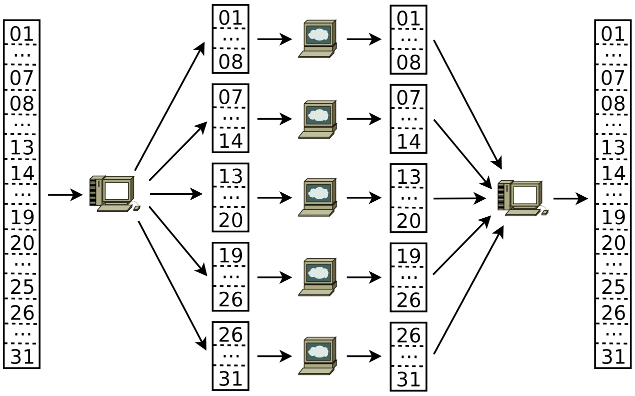


Fig. 5 The segmentation and merge operations

The median filter substitutes the value of a pixel by the median value from its neighborhood. It sorts the pixels in the neighborhood and picks the median pixel value as the new value for the pixel of interest. Thus, for a neighborhood of size 1 (3 × 3 filter), 9 elements of the matrix are sorted and the median value is selected. For a neighborhood of size 2 (5 × 5 filter), 25 elements are sorted, and so on. In our experiments we show results for 3 × 3 and 5 × 5 filters.

To apply the filter in a distributed way, we first split the image into smaller blocks, then applied the median filter on each block separately, and merged the blocks to generate the final image. Because the filter uses a median value of a neighborhood, there is data redundancy in the generated blocks. For instance, in the 3 × 3 filter the first and last lines on each block are duplicated, thus the first and last lines of the block can have the filter applied using their full neighborhood. Figure 5 illustrates the segmentation and merge operations. Note that application is essentially a fork-join DAG, as the one shown in Fig. 2. The execution and communication costs for the DAG were estimated from historical data from other executions of the filter.

We developed a service to execute the application which applies the filter on the matrix file. This service is used by the workflows and it provides several operations, such as splitting the file into blocks according to the desired number of blocks, applying the median filter on a file, generating another file with the resulting block/matrix, and merging the blocks to generate the resulting image. A set of workflows was created in GPOL (introduced in Sect. 2) to evaluate the scheduling performance.

We evaluated HCOC in the resources shown in Tables 1 and 2 with the median filter workflow using a matrix of size 7500 × 7500 with different number of slices and filter sizes. We measured the execution time (makespan) and cost for executing the workflows, computing the average over 3 executions. Note that the cost for executing workflows in the private cloud is 0, thus it is not shown in the graphs.

We considered $\mathcal{D} = 35$ for 3 × 3 filter and $\mathcal{D} = 50$ for 5 × 5 filter. Figure 6 shows the average makespan and average cost for the median filter with 4, 8, and 12 slices. Using only locally available resources from the private cloud cannot meet the deadline, resulting in the highest makespan among the three scenarios. The greedy scheduling can meet the deadline with a very low makespan; however, the cost for the execution is high. The execution in the hybrid system with HCOC can meet the deadline with costs 2 to 5 times lower than greedy average, as we can observe in Fig. 6(b).

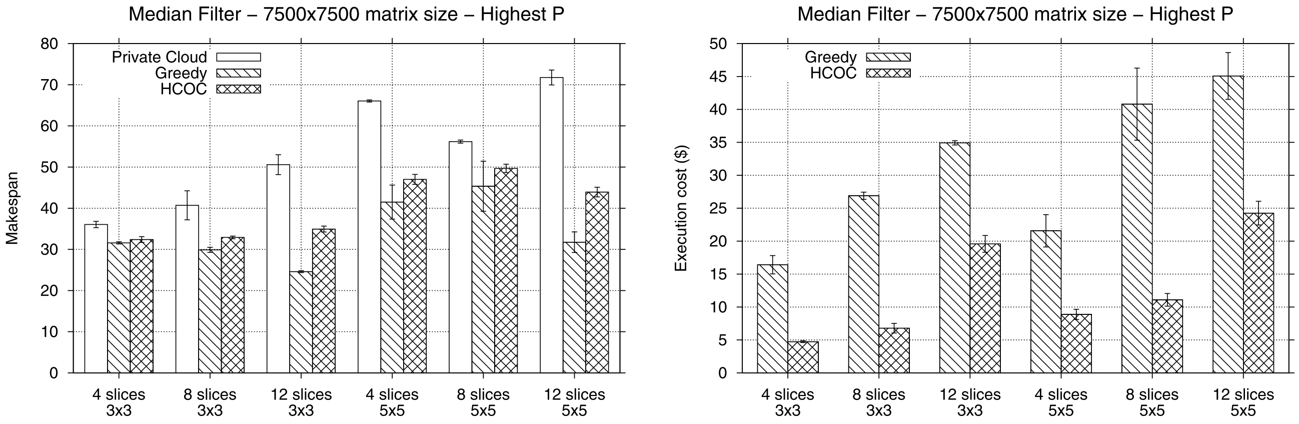
From this initial evaluation we observe that the proposed algorithm can reduce the makespan when compared to the private cloud execution, as well as the cost when compared to the execution in the public cloud. The HCOC algorithm is capable of decreasing the makespan achieved in a local execution without prohibitively increasing the execution cost, as well measures by which extent such variations occur. We can note that in the private cloud, the higher the number of slices, the higher the execution time. This is because the slice and merge operations take more time to split the files, but there are no parallel processors available to execute all the slices. On the other hand, executing most tasks of the workflow in the public cloud can reduce the execution time, since there are more processors available and more slices can be executed in parallel. In the hybrid execution, the algorithm stops requesting public cloud resources when the deadline is met, thus there are fewer slices executing in parallel than in the public cloud execution, but more than in the private cloud execution.

5.2 Simulation results

In the simulations we evaluate the algorithms using different cloud sizes, workflows, and deadlines. The evaluation of the proposed algorithm, the Hybrid Cloud Optimized Cost (labeled as *HCOC* in the graphs), is done using the three different task selection policies for rescheduling described in Sect. 4 (highest P, highest $P + EST$, and highest EFT). The proposed algorithm is compared with two different approaches: *Private Cloud* and *Greedy*. The *Private Cloud* approach considers only resources available in the private cloud when scheduling, which actually is the initial schedule. The *Greedy* approach considers both resources from private and public clouds, but resources from the public cloud are selected based solely on their performance no matter their costs. In addition, we present results that compare the communication costs involved when using each scheduling algorithm.

5.2.1 Simulation scenarios and configurations

The simulation environment was set up with one private cloud and public cloud resources that could be requested by the scheduler to be part of the hybrid infrastructure. Each simulation was repeated 1000 times.



(a) Makespan

(b) Cost

Fig. 6 Results for median filter DAGs

In each simulation we generated a private cloud $\mathcal{R}$ with 2 to 10 resources. Each resource had its processing power randomly taken from the interval (10, 100): $\forall r_i \in \mathcal{R}$, $p_{r_i} = random(10, 100)$. Links between resources in the private cloud had their bandwidth randomly taken from the set $\{40, 50, 60, 70, 80\}$. The external link of the private cloud (i.e., the link that connects the private cloud to the public clouds) had its bandwidth randomly taken from the set $\{20, 30, 40, 50\}$. These resource and link configurations aim to cover a wide range of processor capacities and bandwidth, combining high processing power with low bandwidth and vice versa. Additionally, the external bandwidth is usually lower than the internal bandwidth, which is a common configuration in real environments.

On the other hand, public cloud resources could be requested by the scheduler as it needed, without a limit in the number of resources that could be leased. The number of cores and costs per time unit for using public cloud resources followed what is shown in Table 2. Processing power from public cloud's virtualized resources were randomly taken from the interval (50, 150): $\forall u_i \in \mathcal{U}$, $p_{u_i} = random(50, 150)$, while bandwidth between any two resources in the same public cloud was set to 100.

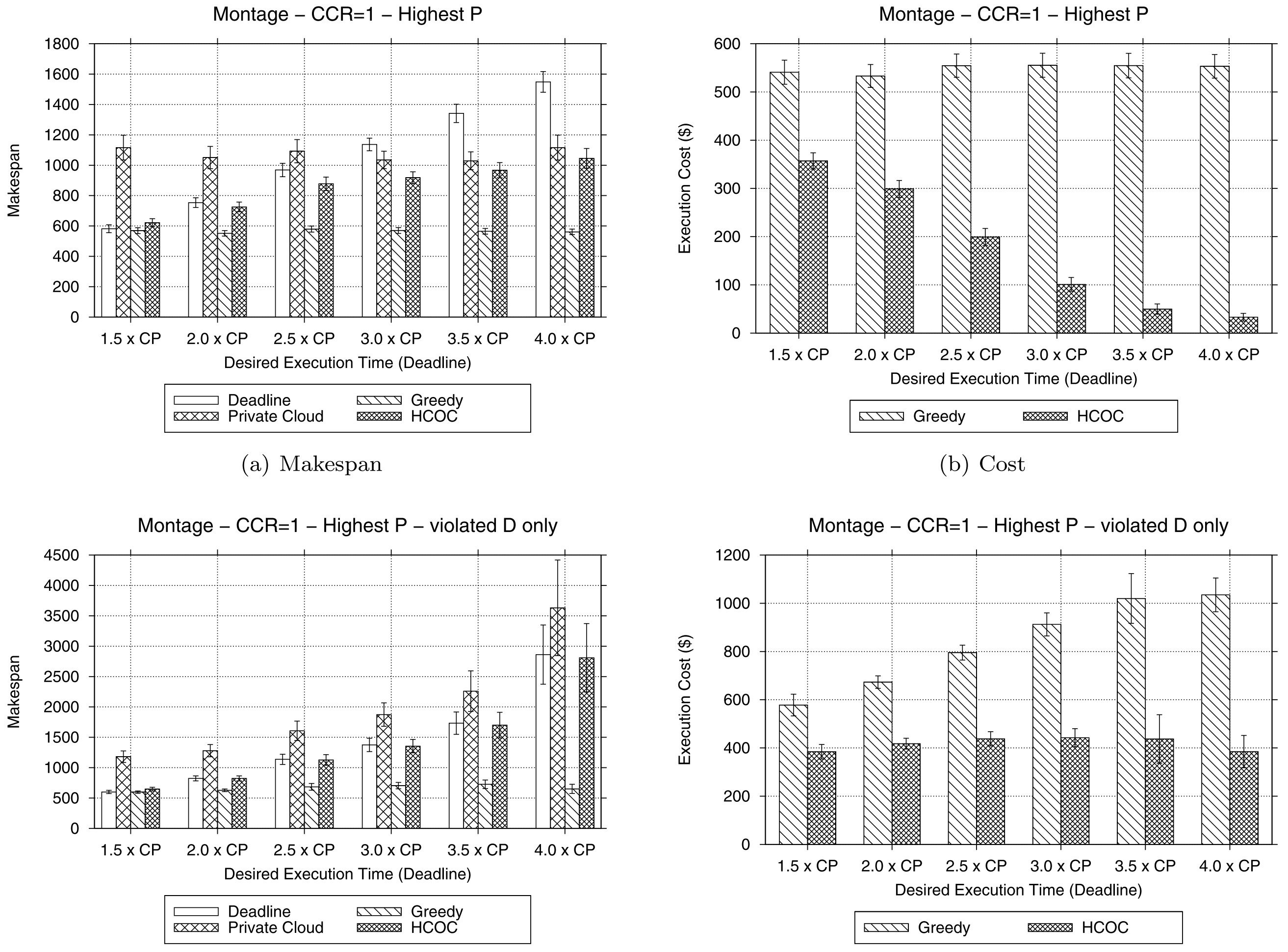
The main metrics evaluated are the average makespan and average cost of the schedules given throughout the simulations, as well as the percentage of times each algorithm did not achieve a makespan lower than $\mathcal{D}$. To compute the cost of a given schedule, we considered all time intervals that each resource was being used for processing a task or transmitting data. Therefore, gaps in the schedule were not computed as resource usage time. In addition, for each scenario and each DAG, we present two graphs for each metric: one that averages over all simulations and another one that averages only over simulations where the private cloud scheduling did not achieve a makespan lower than $\mathcal{D}$.

We used the DAGs shown in Sect. 3.2 to run our simulations. In particular, the Montage application was generalized, generating DAGs with size from 5 to 100. Besides that, random DAGs with size from 5 to 100 were generated using the procedure described in [29]. Each task of the generated DAGs had its computing cost generated from the interval (500, 4000). For each type of DAG, we performed simulations using different communication to computation ratios (CCRs): low communication (CCR = 0.5); medium communication (CCR = 1.0); high communication (CCR = 2.0). CCR is defined as the ratio between the amount of communication and the amount of computation in the workflow.

This first set of simulations used the Highest P policy for selecting tasks to reschedule and medium communication. Thus, this set represents results for Algorithm 1 without any modification. Each result graph shows six sets of bars. Each set presents results for different $\mathcal{D}$ values, varying from $1.5 \times CP$ to $4.0 \times CP$, where CP is the execution time in the best resource available of all tasks in critical path of the currently scheduled DAG. For makespan results, there are four bars in each set: average $\mathcal{D}$, average makespan for *private cloud* scheduling, average makespan for *greedy* scheduling, and average makespan for HCOC scheduling.

5.2.2 *Highest priority*

Figure 7 shows average cost and makespan for the Montage DAG (Fig. 3(a)) with CCR = 1.0 and number of nodes from 5 to 100. We can observe that the hybrid cloud optimized cost approach is very efficient for $\mathcal{D} = 1.5 \times CP$, resulting in an average makespan close to both the average $\mathcal{D}$ and the average makespan given by the greedy algorithm. As we increase $\mathcal{D}$, the HCOC algorithm presents an increase in the makespan, following the increase in $\mathcal{D}$,



(a) Makespan

(b) Cost

(c) Makespan - violated $\mathcal{D}$ only

(d) Cost - violated $\mathcal{D}$ only

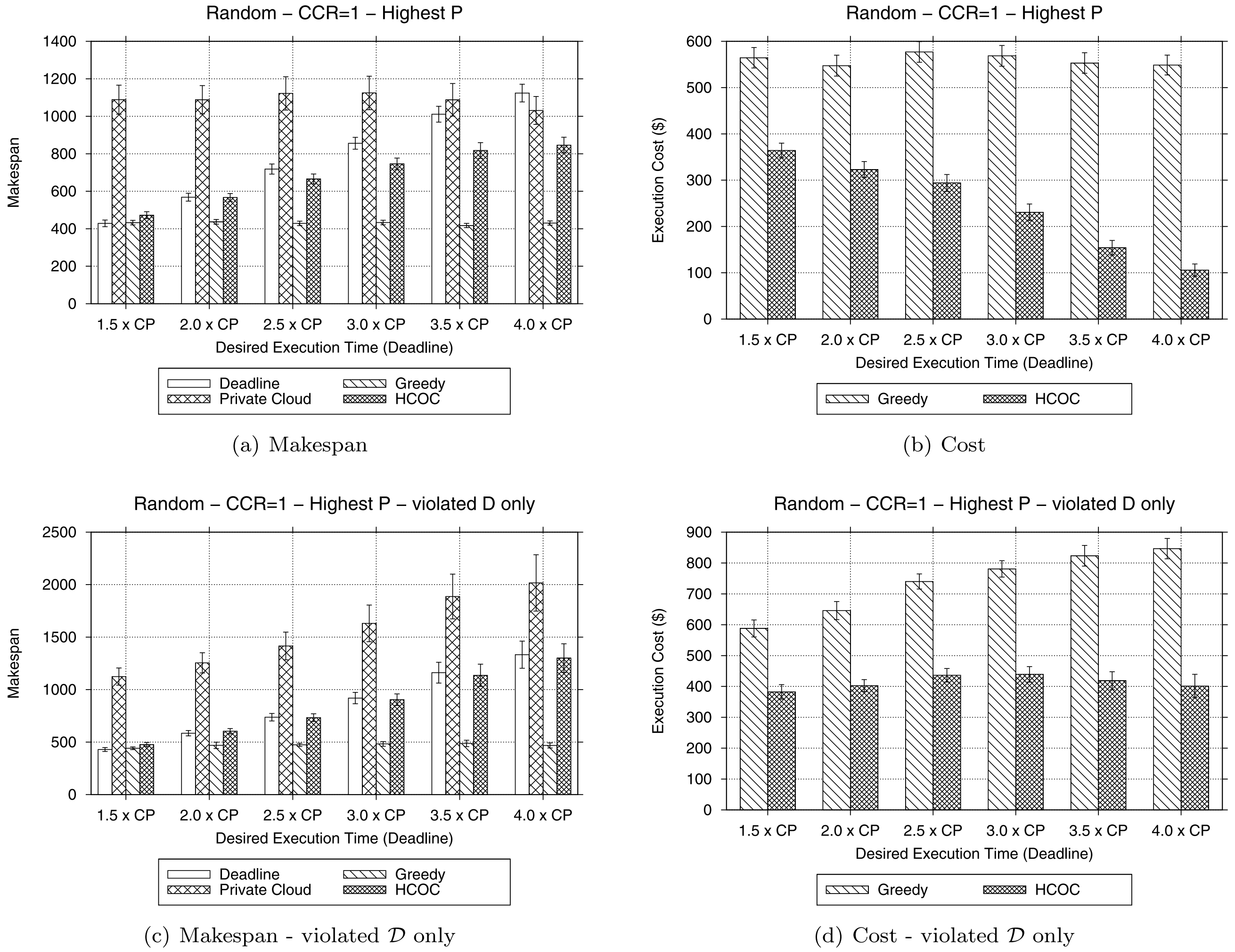
Fig. 7 Results for Montage DAGs with size from 5 to 100 with medium communication (CCR = 1.0)

while the greedy and private cloud averages present negligible changes. This is the desired behavior, and the efficiency of HCOC is corroborated by the average costs. Even presenting an average makespan really close to the greedy approach when $\mathcal{D} = 1.5 \times CP$, HCOC generated makespans with an average cost 34% lower. The higher the $\mathcal{D}$, the lower the cost from HCOC schedules. This cost efficiency when $\mathcal{D} = 1.5 \times CP$ can be explained by the multicore and cost aware scheduling. While HCOC leases resources from the public cloud according to the number of clusters of tasks to be scheduled, the greedy approach tends to lease single core resources for each cluster, since it considers only one cluster of tasks at a time. Additionally, using multicore resources can minimize communication costs, contributing to the minimization of the makespan.

Observing the results for Montage considering only executions where the private cloud approach did not achieve the desired makespan $\mathcal{D}$, we can note that HCOC makespans closely follow the average $\mathcal{D}$. This means that the proposed algorithm can efficiently reduce a makespan larger than $\mathcal{D}$ to a makespan lower but close to $\mathcal{D}$. As a consequence, average costs for those executions are lower than the average given by the greedy algorithm. Note that the average costs in this case have a different behavior when $\mathcal{D}$ increases: while HCOC maintains a nearly constant average cost, the greedy worsens it with higher $\mathcal{D}$s. This is expected when we average only over initial schedules higher than $\mathcal{D}$, since the higher is $\mathcal{D}$, also higher is the makespan that did not meet it. However, this also means that HCOC is efficient in avoiding high costs when the initial schedule makespan is high. Complementing these results, Table 3 shows the percentage of schedules that did not meet the desired execution time for each algorithm. We can observe that HCOC, besides efficiently achieving the objective of reducing costs, can also produce schedules that obey $\mathcal{D}$ in more simulations than both greedy and private cloud approaches. HCOC's multicore awareness contributes to this, since selecting multicore

Table 3 Percentage of schedules that did not meet the deadline for Montage DAGs

Algorithm	Deadline					
	$\mathcal{D} = 1.5 \times CP$	$\mathcal{D} = 2.0 \times CP$	$\mathcal{D} = 2.5 \times CP$	$\mathcal{D} = 3.0 \times CP$	$\mathcal{D} = 3.5 \times CP$	$\mathcal{D} = 4.0 \times CP$
Private cloud	92.7	71.2	45.1	22.6	11.1	8.5
Greedy	47.5	22.7	14.7	6.1	3.6	1.5
HCOC	31	12.9	4.8	1.5	0.3	0



(a) Makespan

(b) Cost

(c) Makespan - violated $\mathcal{D}$ only

(d) Cost - violated $\mathcal{D}$ only

Fig. 8 Results for random DAGs with size from 5 to 100 generated using the procedure described in [29] and medium communication (CCR = 1.0)

resources can minimize communication costs, reducing the makespan.

Results for random DAGs with size from 5 to 100 and $CCR = 1.0$ are shown in Fig. 8 and Table 4. We can observe a quite similar behavior to the Montage results. This behavior indicates that HCOC can maintain a good performance with DAGs with different topologies.

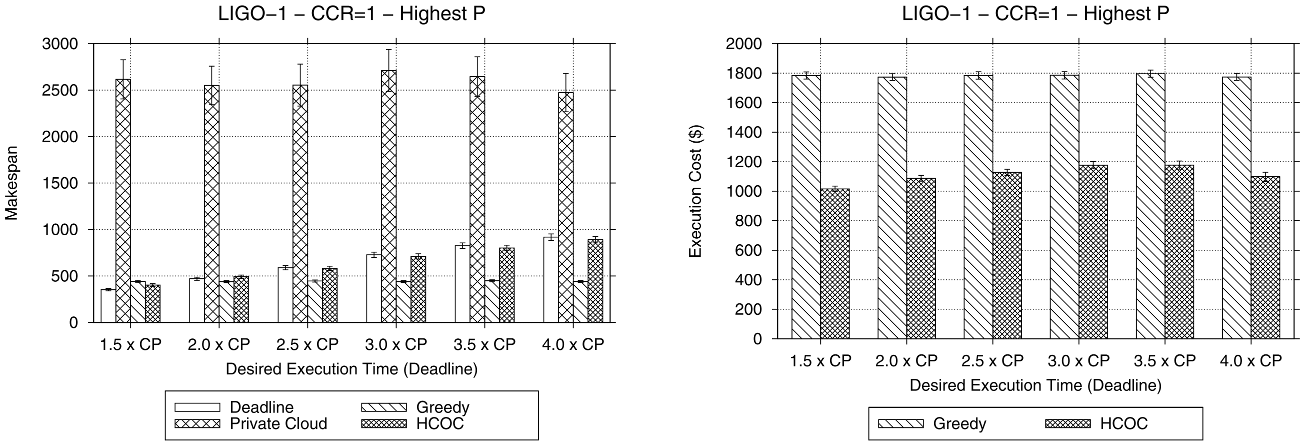
After Montage and random DAGs results, which represent simulations using DAGs of varying sizes, hereon we present results for DAGs with fixed number of nodes. In the case of LIGO, results show that HCOC remains efficiently generating schedules. Figure 9 shows that, when averaging over all simulations, makespans generated by HCOC tend to follow $\mathcal{D}$. On the other hand, the private cloud average makespan is as much as 350% higher than HCOC's average. Because of that, and in consequence of LIGO having much more tasks than the number of resources available in the private cloud, HCOC needs to reschedule most tasks from the initial schedule to the public cloud, even when $\mathcal{D} = 4.0 \times CP$. However, for all $\mathcal{D}$ values, HCOC gener-

Table 4 Percentage of schedules that did not meet the deadline for random DAGs

Algorithm	Deadline					
	$\mathcal{D} = 1.5 \times CP$	$\mathcal{D} = 2.0 \times CP$	$\mathcal{D} = 2.5 \times CP$	$\mathcal{D} = 3.0 \times CP$	$\mathcal{D} = 3.5 \times CP$	$\mathcal{D} = 4.0 \times CP$
Private cloud	95.2	80.1	66.8	52.1	36.1	26.1
Greedy	50.5	28.4	15.7	11.3	4.7	3.3
HCOC	33	24.5	10.5	5.3	2.3	1

Table 5 Percentage of schedules that did not meet the deadline for LIGO-1 DAGs

Algorithm	Deadline					
	$\mathcal{D} = 1.5 \times CP$	$\mathcal{D} = 2.0 \times CP$	$\mathcal{D} = 2.5 \times CP$	$\mathcal{D} = 3.0 \times CP$	$\mathcal{D} = 3.5 \times CP$	$\mathcal{D} = 4.0 \times CP$
Private cloud	100	100	100	100	99.7	97.8
Greedy	75.9	44.7	28.9	16.8	12.4	6.7
HCOC	33.4	28.1	19	11.8	8.2	4.9



(a) Makespan

(b) Cost

Fig. 9 Results for LIGO-1 DAGs (Fig. 3(d)) and medium communication (CCR = 1.0)

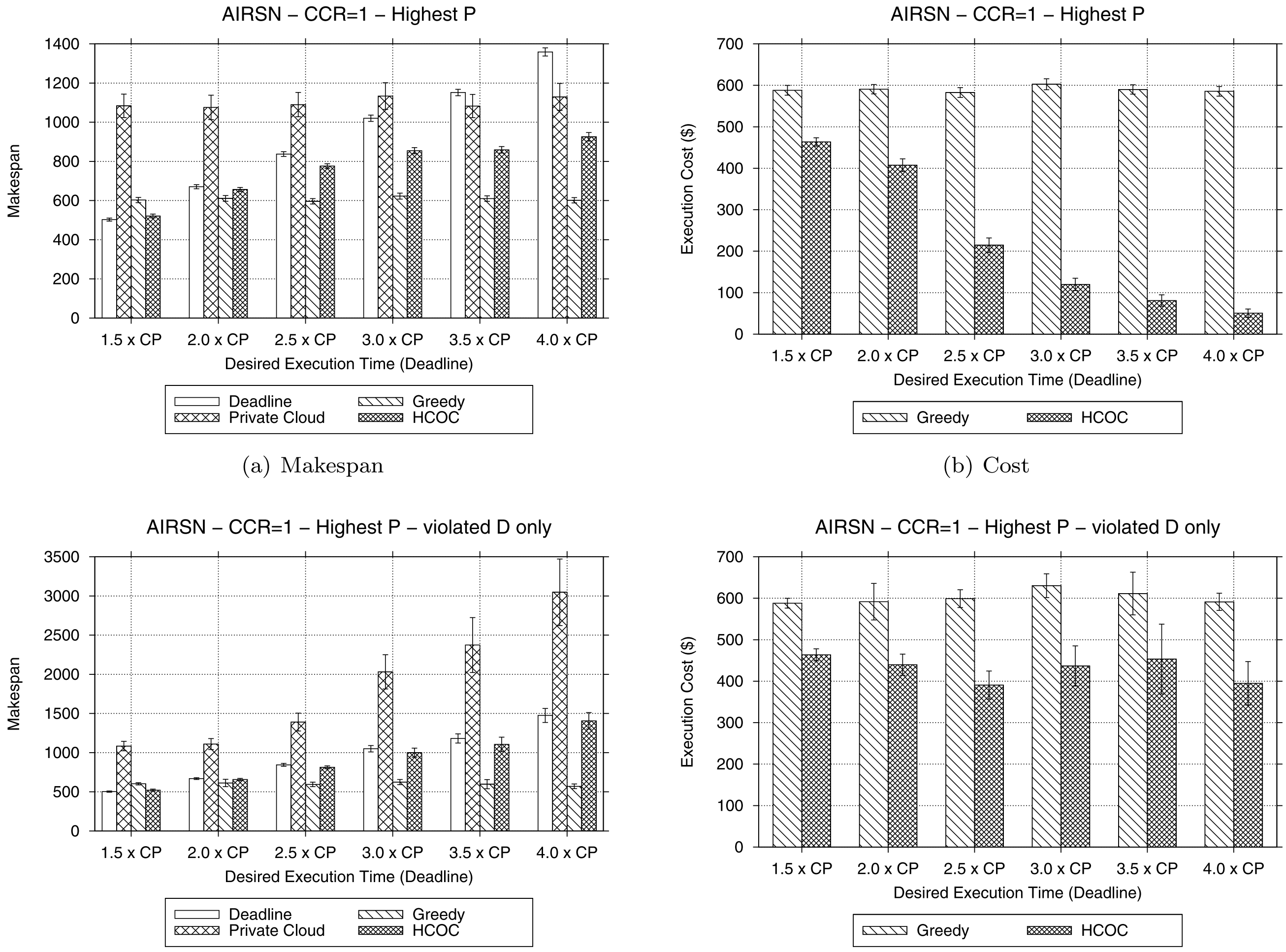
ates schedules with costs around 45% less than the greedy algorithm.

Considering only executions where the private cloud approach did not achieve the desired makespan, LIGO results are very close to the ones achieved for the average over all simulations, therefore we omit such graphs. This happens because nearly 100% of the simulations resulted in the private cloud scheduling do not achieving the desired makespan $\mathcal{D}$, as shown in Table 5. From this table we can also observe that HCOC can meet the deadline more often than greedy.

Figure 10 and Table 6 present results for the AIRSN workflow. These graphs are similar to Montage and random DAGs results. It is worth noting, however, that for $\mathcal{D} = 1.5 \times CP$ the proposed algorithm can reach an average makespan lower than the one given by the greedy algorithm. This happens because the greedy algorithm did not reach the desired makespan on average, and HCOC takes advantage of its multicore policy to reduce the makespan to as low as $\mathcal{D}$. Such policy, along with HCOC's cost-awareness, is also responsible for this achievement being possible even with a lower average cost than the one presented by the greedy.

For Chimera-1 simulations (Fig. 11 and Table 7), results show an intermediate behavior between Montage and LIGO: while the makespan increases with $\mathcal{D}$, the cost stills showing slight changes (as LIGO does) up to $\mathcal{D} = 2.5 \times CP$. Above this value, the average cost starts to get lower (as in Montage graphs). We observed that this change of behavior happens when the desired makespan reaches a value where an entire DAG level may be left in the private cloud, lowering the public cloud leasing costs.

Chimera-2 DAG results are similar to the ones observed for LIGO-1 DAGs. Therefore, we also omit additional graphs for Chimera-2. Simulations using CSTEM DAGs



(a) Makespan

(b) Cost

(c) Makespan - violated $\mathcal{D}$ only

(d) Cost - violated $\mathcal{D}$ only

Fig. 10 Results for AIRSN DAGs (Fig. 3(b)) and medium communication (CCR = 1.0)

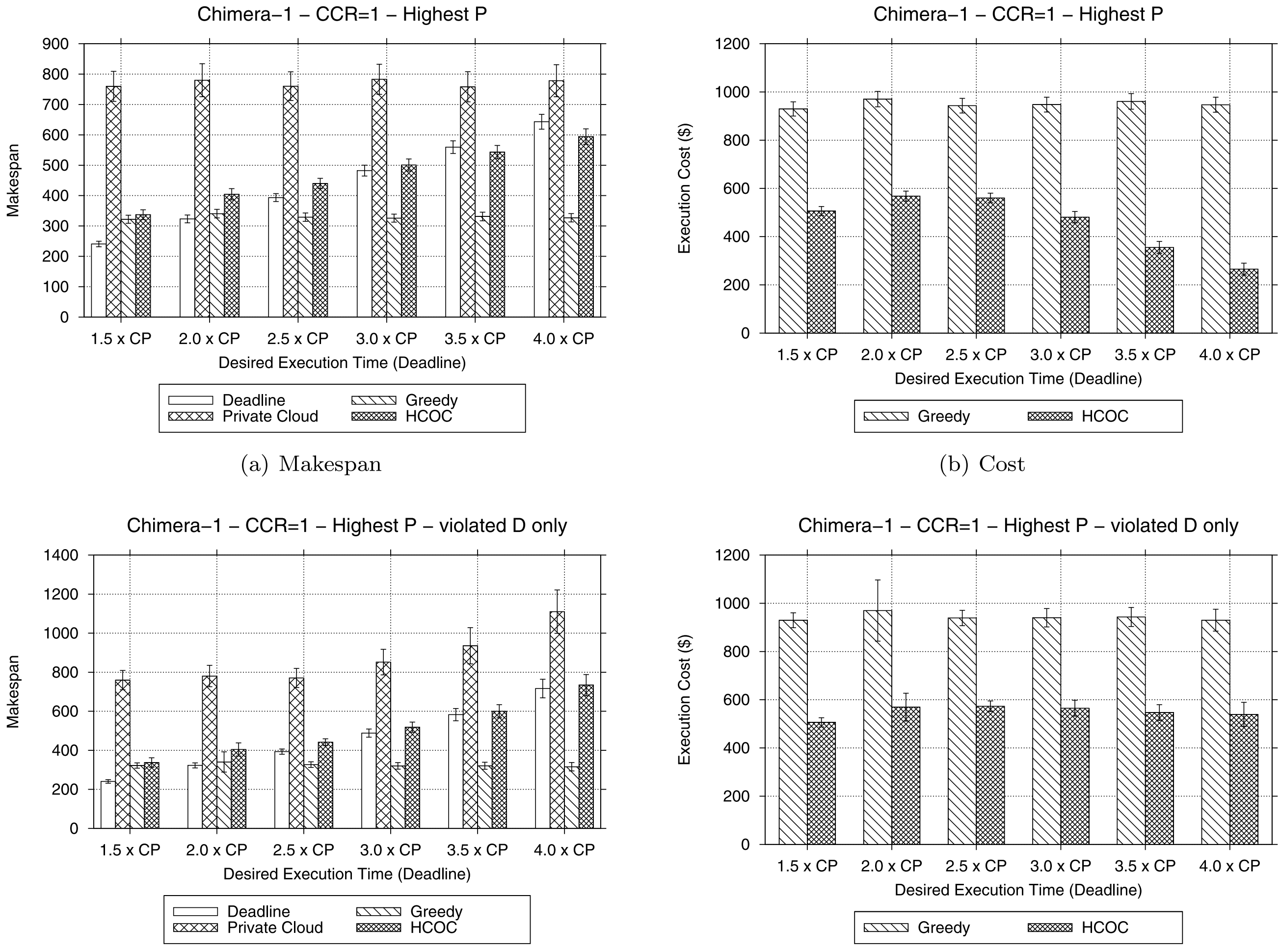
Table 6 Percentage of schedules that did not meet the deadline for AIRSN DAGs

Algorithm	Deadline					
	$\mathcal{D} = 1.5 \times CP$	$\mathcal{D} = 2.0 \times CP$	$\mathcal{D} = 2.5 \times CP$	$\mathcal{D} = 3.0 \times CP$	$\mathcal{D} = 3.5 \times CP$	$\mathcal{D} = 4.0 \times CP$
Private cloud	100	92.6	54.3	27.2	17.8	12.5
Greedy	60.4	35.4	17.6	10.7	5	2.3
HCOC	29.4	14.2	3.2	1.2	0.3	0

showed that the initial schedule was capable of satisfying $\mathcal{D}$ in nearly all simulations. LIGO-2 results are omitted, since they showed similar behavior to Montage and random DAGs, corroborating the results.

Table 8 presents how many tasks were rescheduled on average by HCOC, showing averages for simulations with CCR = 1 and highest P rescheduling policy. Results from this table also support the conclusions presented in Sect. 5.2.2. For example, we observe in Table 8 that Chimera-1 DAG has a sudden lowering in the number of rescheduled tasks when $\mathcal{D}$ is increased from $3.0 \times CP$ to $3.5 \times CP$. This behavior is one of the explanations for the pattern shown in Fig. 11(b), where the average cost starts dropping when $\mathcal{D} = 3.0 \times CP$.

In addition, we can see from the results that, depending on the relation among deadline and critical path, private resources can meet the deadline in most executions (Tables 3 to 5, for instance). However, the HCOC is still useful in such cases, since it can produce schedules that only use public resources when needed, thus reducing costs.



(a) Makespan

(b) Cost

(c) Makespan - violated $\mathcal{D}$ only

(d) Cost - violated $\mathcal{D}$ only

Fig. 11 Results for Chimera-1 DAGs (Fig. 3(f)) and medium communication (CCR = 1.0)

Table 7 Percentage of schedules that did not meet the deadline for Chimera-1 DAGs

Algorithm	Deadline					
	$\mathcal{D} = 1.5 \times CP$	$\mathcal{D} = 2.0 \times CP$	$\mathcal{D} = 2.5 \times CP$	$\mathcal{D} = 3.0 \times CP$	$\mathcal{D} = 3.5 \times CP$	$\mathcal{D} = 4.0 \times CP$
Private cloud	100	99.8	97.4	86.4	63.8	49
Greedy	63.4	38.5	31.2	21.7	16.7	14.4
HCOC	40.2	35.3	30.4	19.7	12	8.2

Communication overhead In order to measure the impact of distributing tasks among the hybrid cloud resources, we evaluate the communication costs in the schedules. Given a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we define

$$\mathcal{C} = \sum_{e_{a,b} \in \mathcal{E}} c_{a,b}$$

as the total communication costs at the end of the scheduling process. Table 9 shows the quotient of $\mathcal{C}_{HCOC}$ with $\mathcal{C}_{Private}$ and $\mathcal{C}_{Greedy}$. We can note that, when compared to the private cloud scheduling, HCOC is able to reduce the total communication with lower deadlines. For example, when $\mathcal{D} = 1.5 \times CP$, the communication costs for HCOC schedules of Montage DAGs were around half of the communication given by the private cloud scheduling (quotient of 0.5498). As $\mathcal{D}$ increases, this difference becomes smaller. This occurs because smaller deadlines force HCOC to concentrate tasks on the public cloud, which has higher link bandwidth as well as multicore resources. On the other hand, when the deadline is higher, most of the tasks are sched-

Table 8 Average number of tasks rescheduled by HCOC

Deadline	DAG (DAG size)						
	Montage (5–100)	Random (5–100)	LIGO-1 (168)	LIGO-2 (77)	AIRSN (53)	Chimera-1 (44)	Chimera-2 (124)
$\mathcal{D} = 1.5 \times CP$	43.72	52.45	164.7	73.66	45.23	43.47	122.88
$\mathcal{D} = 2.0 \times CP$	31.18	44.09	158.89	55.74	33.36	41.9	119.97
$\mathcal{D} = 2.5 \times CP$	17.28	35.9	150.17	35.45	15.59	37.5	114.72
$\mathcal{D} = 3.0 \times CP$	7.29	25.45	139.21	22.1	8	28.38	107.19
$\mathcal{D} = 3.5 \times CP$	3.15	15.28	128.96	13.5	5.21	18.73	96.13
$\mathcal{D} = 4.0 \times CP$	1.66	9.96	114.46	9.11	3.28	12.43	82.78

Table 9 Relation of total communication in schedules given by HCOC algorithm against private and greedy approaches

DAG	$\frac{\mathcal{C}_{HCOC}}{\mathcal{C}_{Private}}$						$\frac{\mathcal{C}_{HCOC}}{\mathcal{C}_{Greedy}}$					
	$1.5XCP$	$2.0XCP$	$2.5XCP$	$3.0XCP$	$3.5XCP$	$4.0XCP$	$1.5XCP$	$2.0XCP$	$2.5XCP$	$3.0XCP$	$3.5XCP$	$4.0XCP$
Montage	0.5498	0.6638	0.7764	0.9158	0.9445	0.9857	1.1598	1.4100	1.748	2.2210	2.2370	2.2100
Random	0.6030	0.8135	0.9387	1.0623	1.0397	1.0395	1.0084	1.2937	1.5473	1.7020	1.6673	1.7902
LIGO-1	0.3761	0.5054	0.6215	0.7090	0.8171	0.9386	1.0645	1.4387	1.7479	2.0202	2.3538	2.5385
LIGO-2	0.9225	0.9491	0.9777	1.0495	1.0474	1.0574	1.7700	1.7433	1.9067	2.0093	2.1221	2.1239
AIRSN	0.6653	0.8855	0.9748	0.9622	0.9888	0.9857	1.1833	1.6695	1.6068	1.8493	1.8968	1.7894
Chimera-1	0.6058	0.6828	0.8115	0.9559	1.0635	1.0761	0.8409	0.9601	1.2003	1.3634	1.5159	1.5300
Chimera-2	0.5644	0.6404	0.6257	0.7164	0.8073	0.9382	0.9908	1.0913	1.1354	1.2089	1.4780	1.6188

uled in the private cloud, approximating the relation to 1. Thus, we conclude that HCOC does not incur in communication overheads when compared to the execution in the private cloud only, actually reducing the overall communication costs in most cases.

In general terms, the same reasoning is valid for the relation with the greedy approach. However, since the greedy approach tends to send most tasks to the public cloud, concentrating tasks on fewer resources, the relation becomes larger as higher is the deadline. For example, HCOC schedules for Montage DAGs can generate around twice communication (quotient of 2.2370 for $\mathcal{D} = 3.5$) when compared to the greedy approach. Therefore, the greedy approach reduces the overall communication costs, but it increases the execution costs.

5.2.3 *Additional results*

Hereon, in this second set of simulation results, we present additional graphs for Montage DAGs with number of nodes from 5 to 100 to corroborate the conclusions outlined hitherto. We have chosen to present Montage results because many other DAGs results showed the same pattern.

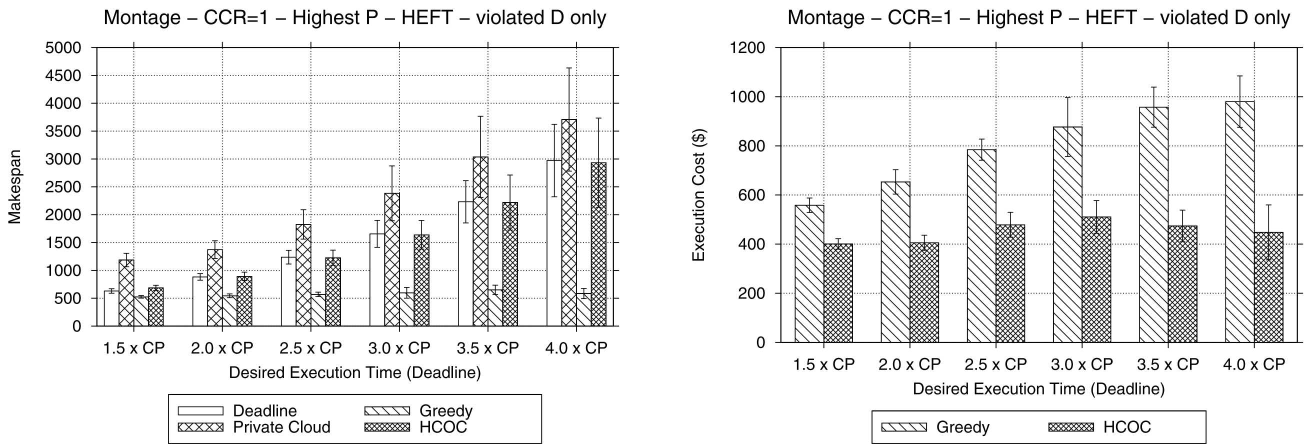
HEFT To evaluate the robustness of the proposed algorithm, we have run simulations using another heuristic to make the initial schedule. The Heterogeneous Earliest Finish Time (HEFT) [32] is a well-known heuristic to schedule DAGs in heterogeneous systems. We observe no significant changes in the general behavior of the results for Montage DAGs, as shown in Figure 12.

Different CCRs We also conducted simulations to evaluate the HCOC algorithm using PCH in DAGs with CCR = 0.5 (Fig. 13) and CCR = 2.0 (Fig. 14). These results show similar patterns to the ones found in simulations with CCR = 1.0.

Amazon EC2 Configuration We conducted simulations using different resources configuration. This new configuration mimics some resources processing power and prices offered by Amazon Elastic Compute Cloud (Amazon EC2) for its on-demand instances, shown in Table 10. Note that we generated a random processing capacity in the interval $(10, 70)$ to represent the small instance power, and the other instance types have their processing power derived from that. This follows the Amazon EC2 instance types,[7] which are based in *EC2 Compute Units* and varying number of virtual cores. According to Amazon, "one EC2 Compute Unit provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor".[8]

[7] http://aws.amazon.com/ec2/instance-types/.

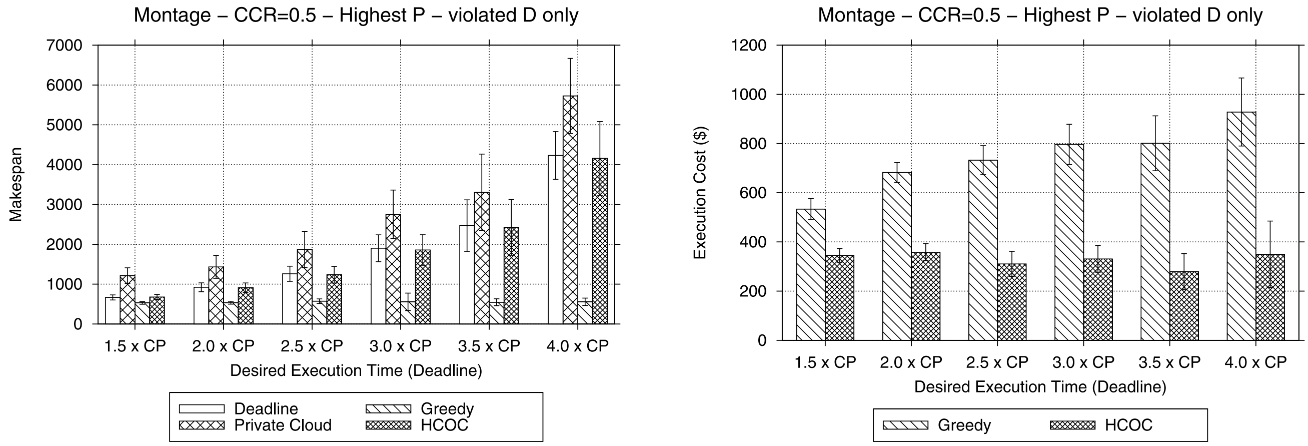
[8] http://aws.amazon.com/ec2/faqs/.



(a) Makespan - violated $\mathcal{D}$ only

(b) Cost - violated $\mathcal{D}$ only

Fig. 12 Results using HEFT for Montage DAGs (Fig. 3(a)) with medium communication (CCR = 1.0)



(a) Makespan - violated $\mathcal{D}$ only

(b) Cost - violated $\mathcal{D}$ only

Fig. 13 Results for Montage DAGs with size from 5 to 100 with low communication (CCR = 0.5)

Table 10 Resources configurations from the Amazon EC2 on-demand instances

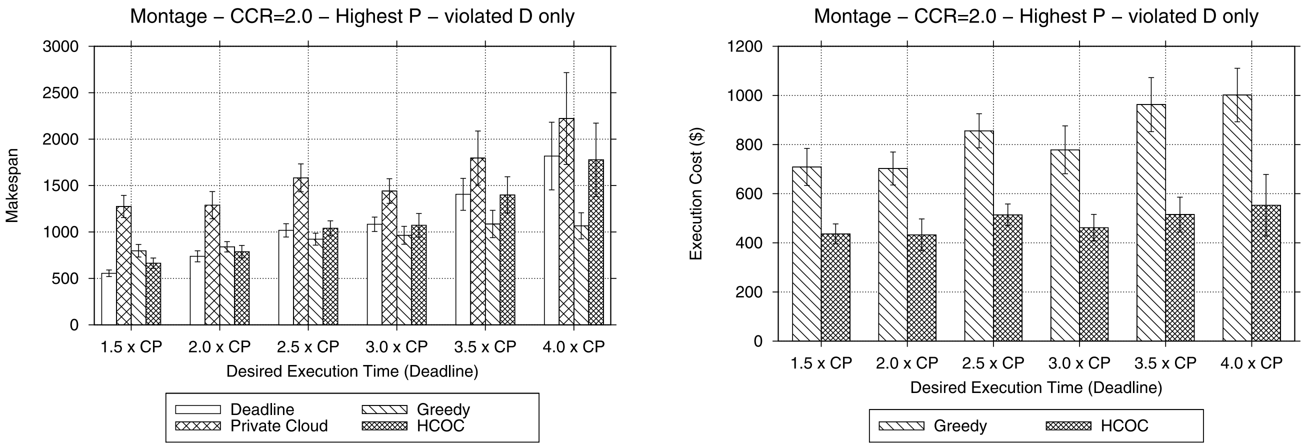
Type	Cores	p_r per core	Price
Small	1	$random(10, 70)$	\$0.085
Large	2	$2 \times p_{Small}$	\$0.34
Extra Large	4	$2 \times p_{Small}$	\$0.68
Extra Large High CPU	8	$2.5 \times p_{Small}$	\$0.68

Average makespan and average costs for Montage DAGs using the Amazon resources configuration shown in Table 10 are presented in Fig. 15. We observe no significant differences in the graph behavior, which indicates that the algorithm is robust enough to deal with changes in the resources capacities and pricing.

Highest $P + EST$ Figure 16 shows graphs for results with the Montage DAGs but using the highest $P + EST$ rescheduling policy. No significant changes are observed when compared to the highest P policy (Fig. 7).

Highest EFT Figure 17 shows graphs for results with the Montage DAGs but using the highest *EFT* rescheduling policy. Again, no significant changes to the highest P policy are observed.

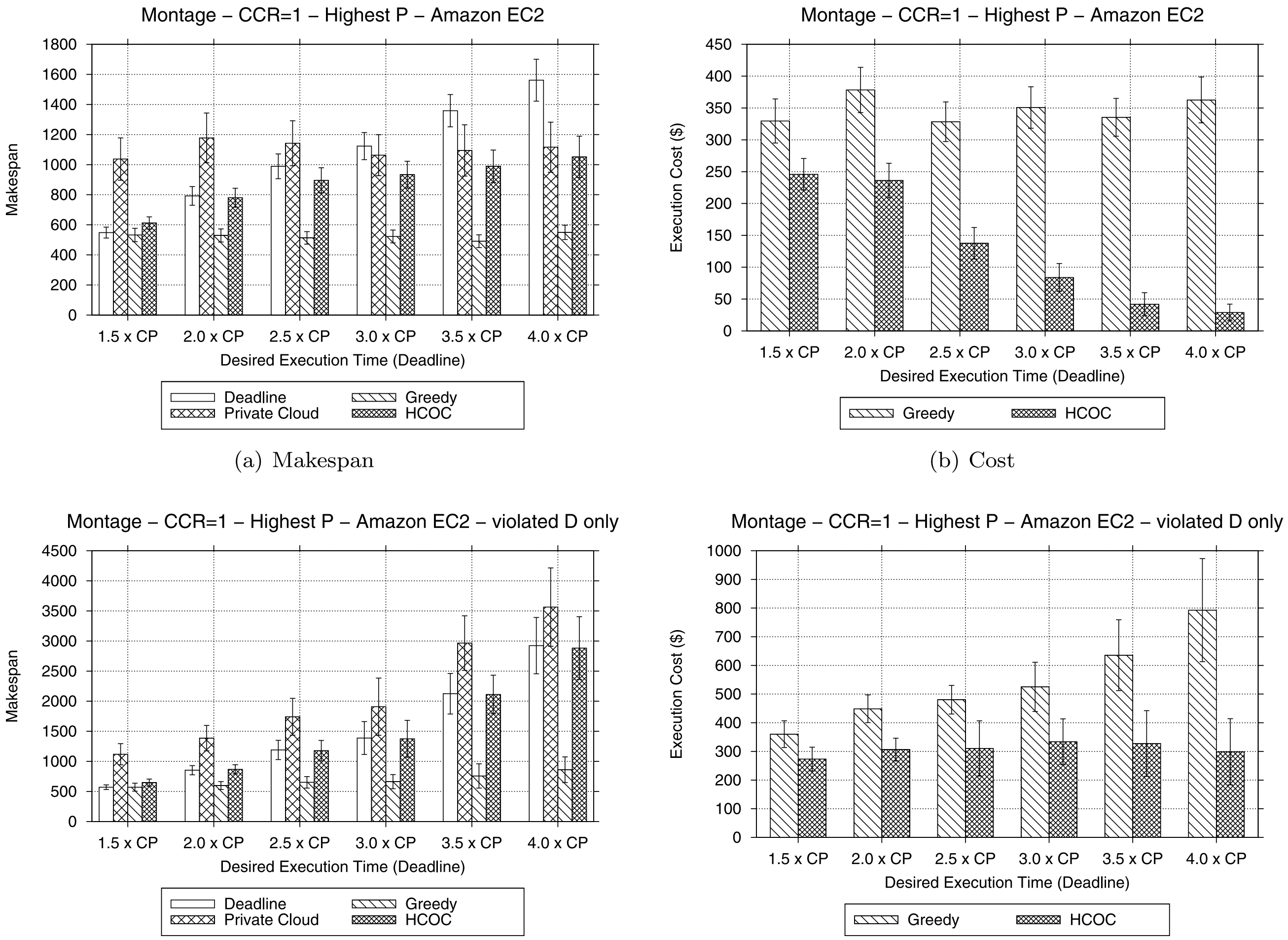
The Highest P policy selects the non-scheduled task t which has the largest path from the beginning of t to the end of the workflow. Therefore, when rescheduling tasks, it starts on the first task of the workflow. On the other hand, highest *EFT* selects the non-scheduled task t which has the largest path from the beginning of the workflow to the end of t. Highest P+EST combines both, trying to select non-scheduled tasks from longest path of the DAG. The iterative



(a) Makespan - violated $\mathcal{D}$ only

(b) Cost - violated $\mathcal{D}$ only

Fig. 14 Results for Montage DAGs with size from 5 to 100 with high communication (CCR = 2.0)



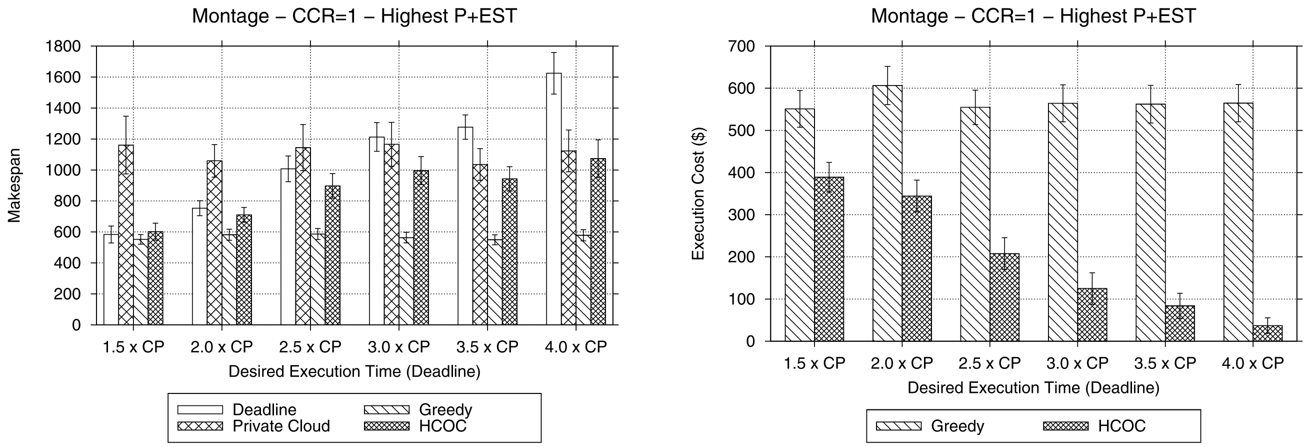
(a) Makespan

(b) Cost

(c) Makespan - violated $\mathcal{D}$ only

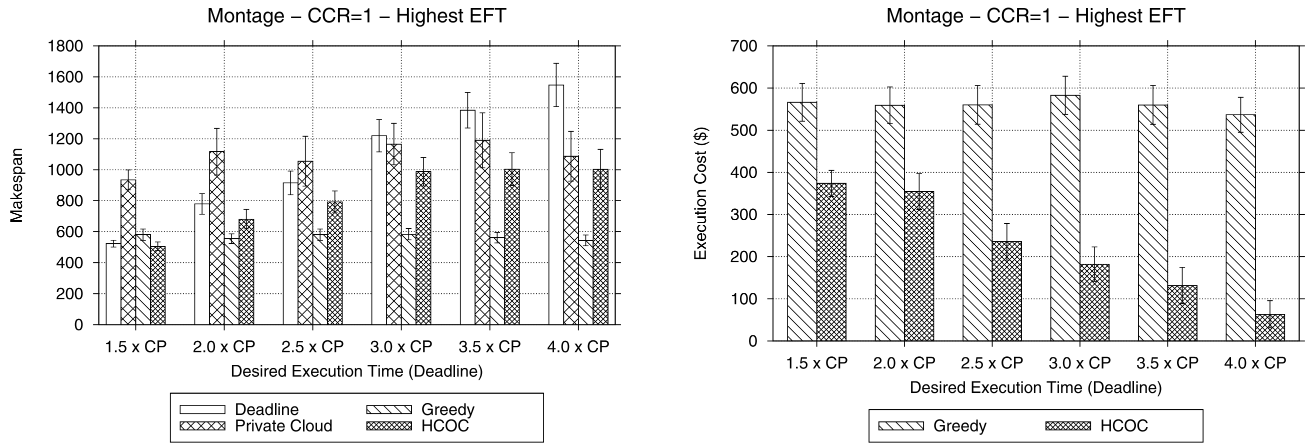
(d) Cost - violated $\mathcal{D}$ only

Fig. 15 Results using Amazon EC2 Cloud configurations for Montage DAGs (Fig. 3(a)) with medium communication (CCR = 1.0)



(a) Makespan (b) Cost

Fig. 16 Results for Montage DAGs with size from 5 to 100, medium communication (CCR = 1.0) and highest $P + EST$ rescheduling policy



(a) Makespan (b) Cost

Fig. 17 Results for Montage DAGs with size from 5 to 100, medium communication (CCR = 1.0) and highest EFT rescheduling policy

rescheduling provided by HCOC handles these differences by scheduling tasks on the public cloud until the deadline $\mathcal{D}$ is met. With this, regardless which tasks are scheduled on the public cloud, the average makespan is close to $\mathcal{D}$.

6 Related works

Due to its importance in reducing application execution times, the scheduling problem is well studied for a variety of problems. In particular, DAG scheduling studies comprise both homogeneous [3, 20, 21, 33] and heterogeneous [7, 27, 29, 32] systems. Task scheduling is an NP-Complete problem [26], thus its optimization version is NP-Hard. Therefore, most efforts in scheduling algorithms are concentrated on heuristics and meta-heuristics [11, 17, 18, 29, 32]. There are some works which use approximation algorithms [25, 16] and combinatoric approaches [5]. Extensive descriptions about scheduling in computational grids are presented in [15] and [19].

In distributed systems scheduling, the most common objective found in the literature is the makespan minimization. However, works that consider costs and deadlines also exist. Yu and Buyya present in [34] a study showing how to schedule scientific workflow applications with budget and deadline constraints onto computational grids using basic genetic algorithms. The proposed approach obtains a variable performance depending on the structure of the DAG being scheduled. Moreover, the algorithm does not consider multicore resources during the resource selection, what could make the genetic algorithm convergence time higher in hybrid clouds. In another work [35], Yu et al. focus on tasks deadline to try to meet the whole workflow deadline. Additionally, a rescheduling policy is presented to try to avoid resource load peaks that could affect the achievement of a deadline. These works do not consider multicore nor the dynamic provision/leasing of resources, available in a hybrid cloud. However, the rescheduling policy proposed could complement our work if used in the private cloud to manage unexpected loads on shared resources.

In [10], Van den Bossche et al. deal with the scheduling of independent tasks on hybrid clouds. They propose a binary integer program to decide which cloud provider to choose when outsourcing a task that cannot be currently executed in the private cloud. Workflow scheduling in conjunction with service composition in grids is discussed by Zhang et al. in [37]. An algorithm to schedule composed services is proposed. The composed services are modeled as a workflow, and then this workflow is scheduled using a technique called *list scheduling*. Fujimoto and Hagihara present an approximation algorithm for task scheduling in [16]. The algorithm uses the resource utilization as the performance criterion, not focusing on makespan minimization, costs, or deadlines. Since approximation algorithms are hard to be developed, the authors make some assumptions to make the problem more specific, like considering that all tasks of the workflow have the same weight.

The scheduling of DAGs in hybrid clouds is a challenge that has been becoming important. As hybrid clouds become popular, aggregating public clouds and private clouds or grids, deciding where to execute tasks aiming performance and cost reduction turns highly relevant. This problem stills in its infancy and in this paper we investigate it proposing the HCOC algorithm.

7 Conclusion

Hybrid clouds are being used to execute different kinds of applications. Among them, workflows have an important role in processes of many fundamental science fields, such as physics, chemistry, biology, and computer science. To speed up science advancements, it is important to provide efficient resource utilization as well as to make application executions affordable.

In this paper we present HCOC: The Hybrid Cloud Optimized Cost scheduling algorithm. HCOC is an algorithm to speed up the execution of workflows obeying a desired execution time, but also reducing costs when compared to the greedy approach. The extensive evaluation carried out in this work provides sufficient data to support the conclusion that the HCOC algorithm can provide efficient scheduling in a hybrid cloud scenario. Its multicore awareness, along with the cost knowledge, can provide makespans as low as the user needs. As a consequence, the user is able to control costs by adjusting the desired workflow execution time $\mathcal{D}$ according to his willingness. In general, the proposed algorithm has the ability of reducing the execution costs in the public cloud with the increase of the workflow desired execution time. Besides that, in some cases where the desired execution time is too low, HCOC finds better schedules than the greedy approach by taking advantage of multicore resources, reducing the number of violations of $\mathcal{D}$. HCOC had its robustness attested by simulations using different rescheduling criteria and heuristics to produce the initial schedule.

Furthermore, HCOC can be easily adapted to handle budgets instead of deadlines, what makes it flexible to the use with different requirements. As future work, to deal with multiple workflows is an important issue. Besides that, considering the potential presence of external load in private resources could improve the scheduling decisions. Another important point to be approached is how to estimate the available bandwidth between two public clouds. Such estimation is essential for the scheduling algorithm to decide when dependent tasks could be put in different public clouds.

Acknowledgements The authors would like to thank FAPESP (09/15008-1), and CNPq (142574/2007-4) for the financial support.

References

1. The NIST definition of cloud computing 15. Tech rep, National Institute of Standards and Technology (NIST) (2009). http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc
2. Abels T, Dhawan P, Chandrasekaran B An overview of xen virtualization (2009). http://www.dell.com/downloads/global/power/ps3q05-20050191-Abels.pdf
3. Amir Y, Awerbuch B, Barak A, Borgstrom RS, Keren A (2000) An opportunity cost approach for job assignment in a scalable computing cluster. IEEE Trans Parallel Distrib Syst 11(7):760–768
4. Annis J, Zhao Y, Voeckler J, Wilde M, Kent S, Foster I (2002) Applying Chimera virtual data concepts to cluster finding in the Sloan Sky Survey. In: Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on supercomputing. IEEE Computer Society Press, Los Alamitos, pp 1–14
5. Batista DM, da Fonseca NLS, Miyazawa FK (2007) A set of schedulers for grid networks. In: SAC '07: Proceedings of the ACM symposium on applied computing. ACM, Seoul pp 209–213
6. Bittencourt LF, Madeira ERM (2008) A performance-oriented adaptive scheduler for dependent tasks on grids. Concurr Comput 20(9):1029–1049
7. Bittencourt LF, Madeira ERM (2010) Towards the scheduling of multiple workflows on computational grids. J Grid Comput 8:419–441. doi:10.1007/s10723-009-9144-1
8. Bittencourt LF, Senna CR, Madeira ERM (2010) Enabling execution of service workflows in grid/cloud hybrid systems. In: International workshop on cloud management (Cloudman). IEEE Computer Society Press, Osaka
9. Bittencourt LF, Senna CR, Madeira ERM (2010) Scheduling service workflows for cost optimization in hybrid clouds. In: 6th international conference on network and service management (CNSM). Toronto, Canada, 2010

10. Van den Bossche R, Vanmechelen K, Broeckhove J (2010) Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In: 3rd international conference on cloud computing, pp 228–235
11. Corrêa RC, Ferreira A, Rebreyend P (1996) Integrating list heuristics into genetic algorithms for multiprocessor scheduling. In: SPDP '96: Proceedings of the 8th IEEE symposium on parallel and distributed processing (SPDP '96). IEEE Computer Society, Washington, p 462
12. Deelman E (2008) Clouds: an opportunity for scientific applications? (keynote in the 2008 Cracow Grid Workshops)
13. Deelman E, Singh G, Su MH, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J, Laity A, Jacob JC, Katz DS (2005) Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci Program 13(3):219–237
14. Dogan A, Özgüner F (2005) Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. Comput J 48(3):300–314
15. Dong F, Akl SG (2006) Scheduling algorithms for grid computing: state of the art and open problems. Tech rep, Queen's University School of Computing, Kingston, Canada
16. Fujimoto N, Hagihara K (2003) Near-optimal dynamic task scheduling of precedence constrained coarse-grained tasks onto a computational grid. In: 2nd intl symp on parallel and distributed computing, Slovenia. IEEE Press, New York, pp 80–87
17. Grajcar M (1999) Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system. In: DAC '99: Proceedings of the 36th ACM/IEEE conference on design automation. ACM Press, New York, pp 280–285
18. Hou ESH, Ansari N, Ren H (1994) A genetic algorithm for multiprocessor scheduling. IEEE Trans Parallel Distrib Syst 5(2):113–120
19. Yu J, Buyya R, Ramamohanarao K (2008) Workflow scheduling algorithms for grid computing. Stud Comput Intell 146:173–214
20. Kwok YK, Ahmad I (1996) Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. IEEE Trans Parallel Distrib Syst 7(5):506–521
21. Kwok YK, Ahmad I (1999) Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput Surv 31(4):406–471
22. Lindley CA (1991) Practical image processing in: C: acquisition, manipulation and storage: hardware, software, images and text. Wiley, New York
23. Mendell G (2008) Use of Condor by the LIGO scientific collaboration. In: Padaryn/Condor Week. Wisconsin, USA
24. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D (2009) The Eucalyptus open-source cloud-computing system. In: 2009 9th IEEE/ACM international symposium on cluster computing and the grid (CCGRID). IEEE Press, Washington, pp 124–131
25. Phillips C, Stein C, Wein J (1997) Task scheduling in networks. SIAM J Discrete Math 10(4):573–598
26. Pinedo ML (2008) Scheduling: theory, algorithms, and systems. Springer, Berlin
27. Rahman M, Venugopal S, Buyya R (2007) A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. In: Third IEEE international conference on e-Science and grid computing. IEEE Computer Society, Washington, pp 35–42
28. Ramakrishnan A, Singh G, Zhao H, Deelman E, Sakellariou R, Vahi K, Blackburn K, Meyers D, Samidi M (2007) Scheduling data-intensive workflows onto storage-constrained distributed resources. In: CCGRID '07: Proceedings of the seventh IEEE international symposium on cluster computing and the grid. IEEE Computer Society, Washington, pp 401–409
29. Sakellariou R, Zhao H (2004) A hybrid heuristic for DAG scheduling on heterogeneous systems. In: 13th heterogeneous computing workshop. IEEE Computer Society, Washington, pp 111–123
30. Senna CR, Bittencourt LF, Madeira ERM (2009) Execution of service workflows in grid environments. In: International conference on testbeds and research infrastructures for the development of networks & communities (Tridentcom). IEEE Computer Society, Washington
31. Smith JE, Nair R (2005) The architecture of virtual machines. Computer 38(5):32–38
32. Topcuoglu H, Hariri S, Wu MY (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
33. Yang T, Gerasoulis A (1994) DSC: scheduling parallel tasks on an unbounded number of processors. IEEE Trans Parallel Distrib Syst 5(9):951–967
34. Yu J, Buyya R (2006) Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci Program 14(3–4), 217–230
35. Yu J, Buyya R, Tham CK (2005) Cost-based scheduling of scientific workflow application on utility grids. In: Proceedings of the first international conference on e-science and grid computing. IEEE Computer Society, Washington, pp 140–147
36. Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. J Internet Serv Appl 1(1):7–18
37. Zhang S, Zong Y, Ding Z, Liu J (2005) Workflow-oriented grid service composition and scheduling. In: International symposium on information technology: coding and computing, v 2, April 2005, Las Vegas, Nevada, USA. IEEE Computer Society, Washington, pp 214–219
38. Zhao Y, Dobson J, Foster I, Moreau L, Wilde M (2005) A notation and system for expressing and executing cleanly typed workflows on messy scientific data. SIGMOD Rec 34(3):37–43