Scheduler for Data-Intensive Workflows in Public Clouds

Walisson F. Pereira, Luiz F. Bittencourt, and Nelson L. S. da Fonseca
Institute of Computing – University of Campinas (UNICAMP)
Campinas, São Paulo, Brazil 13083-852
Email: {walisson, bit, nfonseca}@ic.unicamp.br

Abstract—**Data-intensive workflows can require the use of intermediary data storage in the order of terabytes. Thus, planning the execution of such workflows in the cloud considering only processing demand, regardless its data storage needs, leads to performance decrease and potential increase in costs. In this paper, we present an integer linear program scheduler that considers disk storage scheduling besides the task scheduling based on processor time. The proposed scheduler aims to achieve the lowest economic cost while serving a deadline set by the user. The results show that the scheduler can find good schedules in situations where the disk size of rented virtual machines is a limiting factor.**

I. Introduction

Science and business have adopted computational tools to assist data analysis and simulation of complex models that usually deal with large amounts of data. Various complex e-Business and e-Science applications can be modeled as workflows, which are composed of sets of tasks to be processed in a well-defined order [1], since each task can depend on the computed data of other precedent tasks. Workflows are usually represented by a Directed Acyclic Graph (DAG), which represents the tasks and their dependencies.

Scientific applications typically demand large amounts of computing resources such as processing, storage, and data transmission. Such applications can be found in a variety of fields, such as Physics (astronomy, thermodynamics), Bioinformatics (DNA sequencing, proteomics), Chemistry (protein dynamics), and Computer Science (computer vision, image processing) [2]. There exist experiments that need to analyze large amounts of data, possibly in the order of terabytes, and that take a lot of time to execute. Such computations often involve thousands of tasks with data dependencies between them, which can be executed in parallel in a distributed environment. Scientific workflows have been employed to represent such applications.

Computing requirements of large workflows with large datasets can overwhelm the locally available resources, specially when those resources are being utilized by other applications. The pay-per-use paradigm is the option offered by public cloud providers to obtain on-demand resources to comply with quality of service requirements, such as maximum desired execution time for a workflow (deadlines). This characteristic of providing computing power and storage when needed avoids upfront investment and low utilization, which can lead to savings in the long-term. On the other hand, such approach also brings many challenges: it is hard to decide which resources to lease and how long such resources will be needed; and it is difficult to evaluate cost-effective options when running in specific environment [3].

The scheduler is the element responsible for determining which task of a workflow runs in which computational resources by distributing the tasks in order to run them in parallel. Thus, the problem of workflow scheduling consists in, given a set of dependent tasks and their dependencies, choosing which resource will run each task. However, the distribution of tasks must be performed according to certain objectives, such as the minimization of the workflow execution time (makespan) and minimization monetary costs.

There are several ways to set up, acquire resources and execute tasks, which makes cloud computing more challenging than in the traditional clusters and grid environments [4]. The task scheduler has the critical role of mapping tasks in virtual instances considering their dependencies, and the cost of use of resources. Schedulers commonly consider CPU power when scheduling applications into the cloud. However, the cost of running an application in the cloud also depends on storage and communication that will be consumed [5].

Cloud providers offer storage in the CPU instances as well as separate storage blocks independently of CPU instances, which, to the best of our knowledge, has not been considered so far in the literature. In this paper, we present a workflow scheduler based on an Integer Linear Programming (ILP) formulation for public clouds that considers storage constraints as well as the possibility of renting CPU-independent storage resources (extra storage).

This work is organized into the following sections: Section II discusses schedulers for public cloud and some solutions to deal with storage constraints; Section III introduces the cloud and application models considered in the paper; Section IV describes the proposed scheduler; Section V provides an analysis of the results; Section VI discusses the conclusion of this work.

II. Related Work

Ramakrishnan *et al.* [6] describe an approach that seeks to minimize the amount of space required by the workflow during its execution and ensures that the amount of data required and generated by the workflow fits on individual resources. Their approach adds cleaning tasks of data files when they are no longer required by other tasks in the workflow, or when they have already been transferred to permanent storage. Furthermore, the scheduler needs to ensure that the resource used has storage space available for the task.

Pandey *et al.* [7] proposed a workflow scheduler in clouds based on Particle Swarm Optimization (PSO). The scheduler takes into account both the cost of computing and the cost of transmission. The scheduler aims to minimize the cost.

Barrett *et al.* [8] proposed an approach for cloud workflow scheduling which employs the Markov Decision Process (MDP) to optimally guide the workflow execution depending on the state of the environment. The scheduler uses genetic algorithm to generate their schedules and MDP is used to choose the best scheduling based on the environment's observed state. Attached to it, a learning algorithm based on Bayesian model was used becoming an agent able to choose the resources according to the feedback of current environment. The goal of MDP is to ensure the complete execution of the workflow within deadline and budget constraints.

Zhu *et al.* [9] proposed a two-steps scheduler which seeks to maximize the use of resources and minimize the overhead of the cloud. A two-steps heuristic workflow mapping called "Algorithm allocation of virtual machines with limited cost-effective runtime" is used. The approach also considers the workflow execution delay if the calculated makespan is less than the deadline seeking to reduce the overhead of startup and shutdown of a virtual machine.

Genez *et al.* [10] address the importance of the scheduler to also consider the I/O performance of virtual machines in a cloud. The authors developed a task scheduler based on ILP which goal is to minimize the cost of execution provided it meets a given deadline. The approach neither tackles the different storage options, nor the cost and space limitations.

To the best of our knowledge, there is no scheduler that considers the possibilities of renting extra storage resources. A scheduler that can rent more storage resources can enable compliance with strict deadlines for accepting higher concentration of jobs in the same virtual machine. The proposed scheduler is an option to scheduling data-intensive workflows in public clouds.

III. Definition of the problem

In this paper, we consider a public cloud model where virtual machine instances can be leased on-demand. These on-demand virtual machines have an associated CPU capacity, a certain amount of RAM available, and storage space specified in the service level agreement established between the provider and the client. The cloud client is able to rent a set of different virtual machines of different capacities over time, leasing and releasing them in a pay-per-use basis.

Besides the storage available in the leased VMs, the client has the option to lease CPU-independent disk storage (e.g. as EBS[1]). These storage blocks are also rented by the client in a pay-per-use basis, where the payment is proportional to the amount of storage utilized and the number of I/O operations performed.

Our focus is on data-intensive applications, in which storage is an important component. Based on the storage options the user has, we consider the following possible combinations on leasing CPU-storage resources:

1) Client rents VMs and utilizes exclusively the in-VM storage to support his/her computation.
2) Client rents VMs with in-VM storage, but is also able to lease CPU-independent storage to support his/her computation.

We consider workflow applications composed of dependent tasks and modeled as directed acyclic graphs (DAGs), where each task can only start its execution after all its predecessors have finished and the data have been transferred to the machine where it is scheduled to execute. Each workflow task (i.e., each node of the DAG) has a computational demand associated, which is translated into how long it takes to run according to the CPU capacity of each VM. Also, the data transmission between two tasks occurs in the network that connects the VMs where these two tasks are scheduled. If they are scheduled to the same VM, the data transmission time between them is zero. For example, Montage [11] (Figure 1) is an application that makes mosaics from images of the sky for astronomy research. The DAG in Figure 2 is a Fork-Join DAG, a generic representation of applications that can be split in independent tasks, sometimes into an arbitrary number of homogeneous or heterogeneous tasks that can be run in parallel. Examples of fork-join applications include image processing filters and a MapReduce [12] application with single map and reduce functions[2].

During the workflow execution, the necessary data to each task can be kept into the storage available in the VM where the task is running, or it can be stored in extra storage from rented storage facilities. While the VM-storage cost is included in the VM price, the extra storage is charged independently. Moreover, this extra storage is deployed over the network, and therefore there exist a maximum transfer speed from this storage to the VM where the task is being run.

IV. The scheduling algorithm

We developed a cloud workflow scheduler based on integer linear programming formulation. The scheduler receives as input the description of the workflow, the description of both the virtual machines and storage available for rent as well as their costs, and the deadline required by the user. With this information, the scheduler decides where tasks should run and where results should be kept so that the execution time is within the desired deadline and the execution cost is minimized.

A. Notation and Problem Modeling

To model the system characterized by the description given in Section III, we used the following notation:

[1]http://aws.amazon.com/pt/ebs/

[2]The MapReduce paradigm includes fault-tolerant mechanisms in addition to the fork-join tasks.

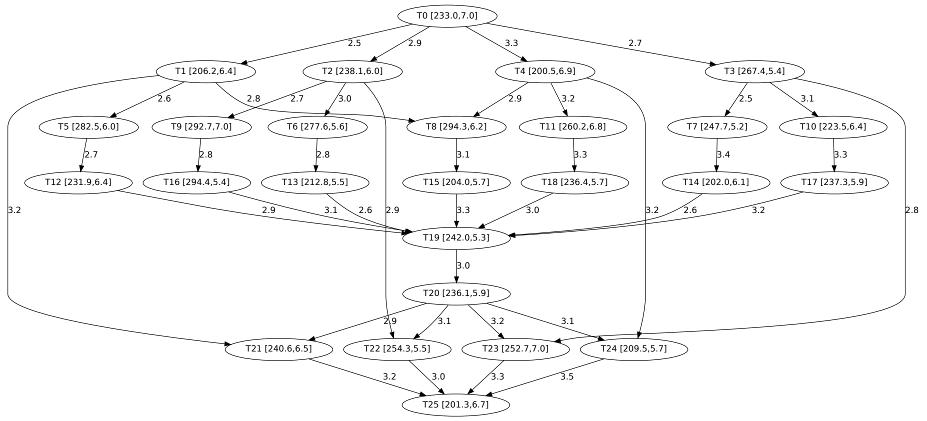

Fig. 1: DAG Montage

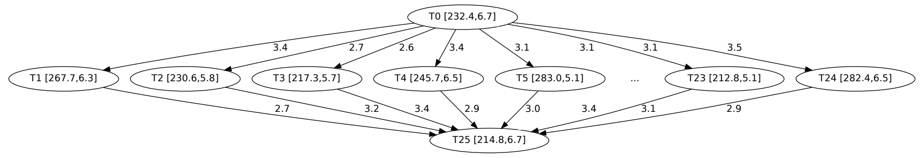

Fig. 2: DAG Fork-Join

- DAG $\mathcal{G} = (\mathcal{U}, \mathcal{E})$ represents a scientific workflow, where $\mathcal{U}$ is the set of tasks (nodes) and $\mathcal{E}$ is the set of dependencies (arcs);
- n indicates the number of tasks of the DAG ($n = |\mathcal{U}|$).
- $w = \{w_1, w_2, \dots, w_n\}$ is the set of computational demands, where w_i is the demand units of task $i \in \mathcal{U}$ ($w \in \mathbb{R}^+$);
- $o = \{o_1, o_2, \dots, o_n\}$ is the set of I/O demands, where o_i is the demand units of task $i \in \mathcal{U}$ ($o \in \mathbb{R}^+$);
- $b_{i,j}$ indicates the communication demand between the tasks i and j, such that $i, j \in \mathcal{U}$ and $\exists e_{i,j} \in \mathcal{E}$ ($b \in \mathbb{R}^+$);
- A graph $\mathcal{C} = (\mathcal{V}, \mathcal{L})$ represents the available infrastructure to rent from a public cloud $\mathcal{C}$, where $\mathcal{V}$ is the set of VM and $\mathcal{L}$ is the set of links. We assumed that all VMs are fully meshed;
- $\mathcal{R}$ is a set of types of available virtual disks to rent in the cloud $\mathcal{C}$. We defined that $r = 0$ means in-VM disk and $r = 1$ means CPU-independent storage. ($\mathcal{R} \in \{0, 1\}$);
- TW_v indicates the time needed to fulfill one computational demand unit by one virtual machine $v \in \mathcal{V}$ ($TW \in \mathbb{R}^+$);
- $TO_{v,r}$ indicates the time needed to fulfill one I/O demand unit by one disk of the type $r \in \mathcal{R}$ on the virtual machine $v \in \mathcal{V}$ ($TO \in \mathbb{R}^+$);
- $TB_{v,k}$ indicates the time needed to transfer one data block from the virtual machine v to a virtual machine k, being $v, k \in \mathcal{V}$. If $v = k$, then $TB = 0$ ($TB \in \mathbb{R}^+$);
- $\mathcal{D}_{\mathcal{G}}$ is the expected deadline ($\mathcal{D}_{\mathcal{G}} \in \mathbb{N}^+$);
- $\mathcal{P}_v$ indicates the number of processing cores of a virtual machine v ($\mathcal{P}_v \in \mathbb{N}^+$);
- $\mathcal{T}_{max}$ is the time needed to execute the whole application serially on the fastest virtual machine with the standard disk (type $r = 0$) ($\mathcal{T}_{max} \in \mathbb{N}^+$);

The formulation uses the following variables and constants:

- $x_{u,t,v,r}$ is a binary variable. It assumes the value 1 if the task u at virtual machine v using the disk type r finishes at time t. Otherwise, it assumes the value 0;
- $y_{t,v}$ is a binary variable. It assumes the value 1 if the virtual machine v is activated at the time t. Otherwise, it assumes the value 0;
- $s_{v,r}$ is a integer variable. It indicates the amount of rented blocks for each type of disk r used by the virtual machine v ($s_{v,r} \in \mathbb{N}^+$).
- $\mathcal{C}_v$ is the cost to rent a virtual machine v by one time unit ($\mathcal{C}_v \in \mathbb{R}^+$);
- $\mathcal{M}_r$ is the cost to rent a gigabyte of a virtual disk of the type r ($\mathcal{M}_r \in \mathbb{R}^+$);
- $\mathcal{X}_r$ is the cost of the ordering a gigabyte of I/O operations of the type r ($\mathcal{X}_r \in \mathbb{R}^+$).

The timeline granularity is a key issue when we use integer linear programming. The time can be represented by both continuous and discrete values. Many public cloud that offers Infrastructure as a Service (IaaS), like Amazon EC2[3], Windows Azure[4], and Rackspace[5], charge a full time unit

[3]http://aws.amazon.com/pt/ec2/
[4]https://www.windowsazure.com
[5]http://www.rackspace.com

(hour or minute) for partial usage. Consequently, the proposed scheduler uses only discrete time to represent the workflow execution. The level of timeline granularity and the workflow length can lead to high execution times. For this reason, we defined λ as a multiplicative factor that determines the granularity of the discrete-time scheduling problem. So, the timeline is described as $\mathcal{T} = \{\lambda, 2\lambda, 3\lambda, ..., \Lambda\}$. Λ is limited by $\mathcal{D}_{\mathcal{G}}$ and $\Lambda \in \mathbb{N}^{+}$. In the simulations carried out in this paper we utilized $\lambda = 2$, i.e., the timeline considered by the scheduler presents twice the granularity offered by the provider with the aim to reduce the search space and reduce the scheduler running time. A detailed study on the impact of timeline granularity is presented in [13].

B. Integer Linear Program Formulation

The developed ILP formulation is presented in Figure 3.

The objective function minimizes the total economic cost. The function is composed of the three parts: the first is the sum of the cost of renting virtual machines; the second is the sum of the cost to rent storage; and the third is the sum of the cost of all the I/O operations.

The constraint (C1) determines that a task u can begin only when all its precedent tasks complete and submit all data to the virtual machine where the task u is allocated. The constraint (C2) states that the maximum number of tasks running simultaneously in the same virtual machine must not exceed the number of processor cores. The constraint (C3) establishes that a virtual machine must remain active while there are tasks running on it. The constraint (C4) determines that a virtual machine must remain active while receiving data from all precedent tasks and while sending data to all successor tasks. The constraint (C5) specifies that every task must be executed only once and in a single virtual machine. The constraint (C6) states that a task cannot end its operations until all instructions are executed and all I/O demands have been performed. The constraint (C7) establishes that the sum of the storage demand of all tasks that use a type of disk in a virtual machine does not exceed its storage capacity. This constraint guarantees that the I/O operations during task execution are considered. The constraint (C8) determines that the workflow execution should not exceed the stipulated deadline. The constraints (C9) and (C10) specify that the variables x and y will only assume the binary values 0 or 1. The last constraint, (C11), determines that the variable namely s will be an positive integer.

V. Evaluation

In this section, we evaluate the benefit of renting storage space in addition to that configured with VM rented. The trade-off analyzed is whether it is more advantageous to rent a higher number of VMs to support the storage demand of applications or to rent a lower number of VMs and additional storage.

We implemented the scheduler as an integer linear program using java[6] and the IBM ILOG CPLEX Optimizer [7]. The evaluated metrics are the monetary cost of the schedule and the workflow makespan.

TABLE I: IaaS Pricing

Service	Cost	Unit
Virtual machine usage	\$0.090	per hour
Virtual machine standard storage	\$0.000	per gigabyte
I/O operations at standard storage	\$0.000	per gigabyte
Extra storage usage	\$0.125	per gigabyte
I/O operations at extra storage	\$0.025	per gigabyte

A. Simulation Configuration

We simulated a cloud that let us rent up to 10 virtual machines. We considered two types of storage: in-VM disk with fixed size, and CPU-independent storage that can be rented without a VM instance (extra storage).

The computational performance of each virtual machine was generated randomly from a uniform distribution between 796-1025 MIPS. The I/O performance for each disk type in each virtual machine are generated randomly from a uniform distribution between 259-316 IOPS whereas each block has 4 KBs. The performance of data transmission on the network is generated randomly from a uniform distribution between 70-702 Mbps. All virtual machines have four processing cores and a 10 GB standard disk. Simulations experiments were run 20 times, each with a different seed for the random number generator. The Table I shows the prices used for renting cloud resources.

We limited the CPLEX solver to execute up to 10 minutes, when the solver delivers the best result found so far. We configured our scheduler in two different manners:

a) Configuration 1: we allow the scheduler to use only VM storage. This was done by adding the following constraint to equation 1 of our ILP formulation (Figure 3).

$$\sum_{u \in \mathcal{U}} \sum_{t \in \mathcal{T}} \sum_{v \in \mathcal{V}} x_{u,t,v,r} = 0, \quad \text{for} \quad r = 1 \tag{1}$$

The constraint in Equation (1) determines that no CPU-independent storage available for rental can be used.

b) Configuration 2: the ILP formulation as presented in Figure 3. It means that the scheduler is free to chose the best way to use the available storage from both the VMs and from the CPU-independent storage.

We simulated the DAG of astronomic application MONTAGE [11] (Figure 1) and a Fork-Join DAG (Figure 2) with 26 tasks. Both applications are data-intensive. The weights of both tasks and arcs were generated randomly from a uniform distribution. The computational demands are between 200-300 millions of instructions. The I/O operation demands are between 5-7 gigabytes. The communication demands are between 2.5-3.5 gigabytes.

To represent possible deadlines, we ran simulations with the deadline $\mathcal{D}_{\mathcal{G}}$ with different ratio in relation to the makespan of the sequential execution of the workflow. We utilized deadlines

[6]http://openjdk.java.net/

[7]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

$$\text{Minimize} \quad \sum_{t\in\mathcal{T}}\sum_{v\in\mathcal{V}}(y_{t,v}\times\mathcal{C}_v)+\sum_{v\in\mathcal{V}}\sum_{r\in\mathcal{R}}(s_{v,r}\times\mathcal{M}_r)+\sum_{u\in\mathcal{U}}\sum_{t\in\mathcal{T}}\sum_{v\in\mathcal{V}}\sum_{r\in\mathcal{R}}(x_{u,t,v,r}\times\lceil o_u\rceil\times\mathcal{X}_r)$$

$$\text{subject to} \quad \sum_{u\in\mathcal{H}(z)}\sum_{v\in\mathcal{V}}\sum_{r\in\mathcal{R}}\sum_{s=1}^{t-\lceil(w_z\times TW_k)+(o_z\times TO_{k,r})+(b_{u,z}\times TB_{v,k})\rceil} x_{u,s,v,r} \geq \sum_{s=1}^{t} x_{z,s,k,h}\times|\mathcal{H}(z)|, \quad \forall z\in\mathcal{U},\ \forall k\in\mathcal{V},\ \forall h\in\mathcal{R},\ \forall t\in\mathcal{T} \tag{C1}$$

$$\sum_{u\in\mathcal{U}}\sum_{r\in\mathcal{R}}\sum_{s=t}^{t+\lceil(w_u\times TW_v)+(o_u\times TO_{v,r})\rceil-1} x_{u,s,v,r} \leq \mathcal{P}_v, \quad \forall v\in\mathcal{V},\ \forall t\in\mathcal{T} \ \mid\ t\leq(\mathcal{D}_\mathcal{G}-\lceil(w_u\times TW_v)+(o_u\times TO_{v,r})\rceil) \tag{C2}$$

$$\sum_{s=t-\lceil(w_u\times TW_v)+(o_u\times TO_{v,r})\rceil+1}^{t} y_{s,v} \geq x_{u,t,v,r}\times(\lceil(w_u\times TW_v)+(o_u\times TO_{v,r})\rceil), \quad \forall u\in\mathcal{U},\ \forall v\in\mathcal{V},\ \forall r\in\mathcal{R},\ \forall t\in\{\lceil(w_u\times TW_v)+(o_u\times TO_{v,r})\rceil,\ldots,\mathcal{D}_\mathcal{G}\} \tag{C3}$$

$$\sum_{p=t+1}^{s-\lceil(w_z\times TW_k)+(o_z\times TO_{k,h})\rceil}(y_{p,v}+y_{p,k}) \geq 2\times(x_{u,t,v,r}+x_{z,s,k,h}-1)\times\pi, \quad \forall z\in\mathcal{U},\ \forall u\in\mathcal{H}(z),\ \forall v,k\in\mathcal{V},\forall r,h\in\mathcal{R},\forall s,t\in\mathcal{T}, \ \alpha=(s-t-\lceil(w_z\times TW_k)+(o_z\times TO_{k,h})\rceil),\ \beta=(\lceil b_{u,z}\times TB_{v,k}\rceil), \ \text{if}\ \alpha\geq\beta\ \text{then}\ \pi=\alpha\ \text{else}\ \pi=\beta \tag{C4}$$

$$\sum_{t\in\mathcal{T}}\sum_{v\in\mathcal{V}}\sum_{r\in\mathcal{R}} x_{u,t,v,r} = 1, \quad \forall u\in\mathcal{U} \tag{C5}$$

$$\sum_{u\in\mathcal{U}}\sum_{v\in\mathcal{V}}\sum_{r\in\mathcal{R}}\sum_{t=1}^{\lceil(w_u\times TW_v)+(o_u\times TO_{v,r})\rceil-1} x_{u,t,v,r} = 0, \tag{C6}$$

$$\sum_{u\in\mathcal{U}}\sum_{s=t}^{t+\lceil(w_u\times TW_v)+(o_u\times TO_{v,r})\rceil-1}(x_{u,t,v,r}\times o_u) \leq s_{v,r}, \quad \forall v\in\mathcal{V},\ \forall r\in\mathcal{R},\ \forall t\in\mathcal{T} \ \mid\ t\leq\mathcal{D}_\mathcal{G}-\lceil(w_u\times TW_v)+(o_u\times TO_{v,r})\rceil \tag{C7}$$

$$\sum_{u\in\mathcal{U}}\sum_{v\in\mathcal{V}}\sum_{r\in\mathcal{R}}\sum_{s=\mathcal{D}_\mathcal{G}+1}^{\mathcal{T}_{max}} x_{u,s,v,r} = 0, \tag{C8}$$

$$x_{u,t,v,r} \in \{0,1\}, \quad \forall u\in\mathcal{U},\ \forall t\in\mathcal{T},\ \forall v\in\mathcal{V} \tag{C9}$$

$$y_{t,v} \in \{0,1\}, \quad \forall t\in\mathcal{T},\ \forall v\in\mathcal{V} \tag{C10}$$

$$s_{v,r} \in \mathbb{N}^+, \quad \forall v\in\mathcal{V},\ \forall r\in\mathcal{R} \tag{C11}$$

Fig. 3: ILP formulation

varying from $T_{max} \times 1/7$ to $T_{max} \times 6/7$ in $1/7$ steps, where T_{max} is the makespan of the serial execution of the application in the fastest virtual machine of the cloud.

The scheduler ran on a computer with the following configuration: Intel® i7™CPU 2.4GHz and 32GB of RAM.

B. Results

Figure 4 presents the distribution of costs for the MONTAGE scheduling. Costs are detailed by the cost of rented VMs, the cost of the rented additional storage (Disk), and the cost of I/O performed by this additional storage. Configuration 1 was not able to fulfill deadlines $1/7$ and $2/7$. This happens because a larger number of VMs are needed to cope with the storage demand. However, given the limitation of VMs that can be rented, not all the tasks can be run in parallel, increasing the makespan which is larger than the required deadlines $1/7$ and $2/7$. In configuration 2, which employs both VM storage and additional storage, the utilization of VM storage is less than that of configuration 1, therefore, it is not necessary to break task to run in different VMs as in configuration 1 just to support the storage demand of the application. On the other hand, configuration 2 implies in higher costs due to the need of extra storage rental and associated I/O operations. In this example, the cost tripled to reduce the makespan to $1/7$ of that given by configuration 1, as shown in Table II.

TABLE II: Average makespan of MONTAGE

	Deadline ratio					
	1/7	**2/7**	**3/7**	**4/7**	**5/7**	**6/7**
Configuration 1	-	-	18.0	14.0	14.0	14.0
Configuration 2	6.0	10.0	2.8	3.4	2.0	2.0

The execution cost distribution is shown in Figure 5 for the Fork-Join DAG. Configuration 1 could not fulfill any deadlines for the $1/7$ ratio. However, for configuration 2, it was possible to meet this deadline by reducing the number of VMs and

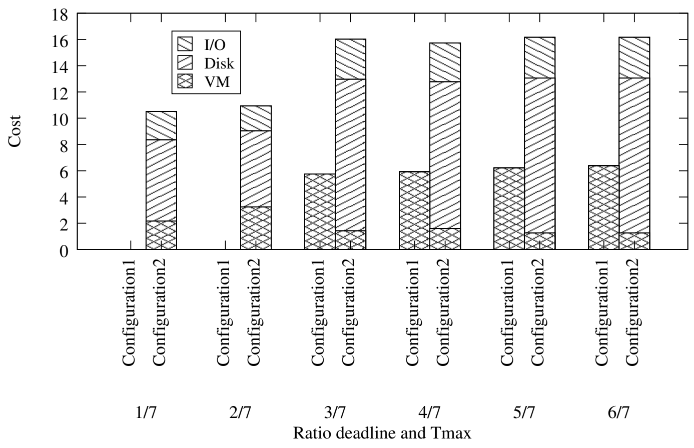

Fig. 4: MONTAGE Execution Cost Distribution

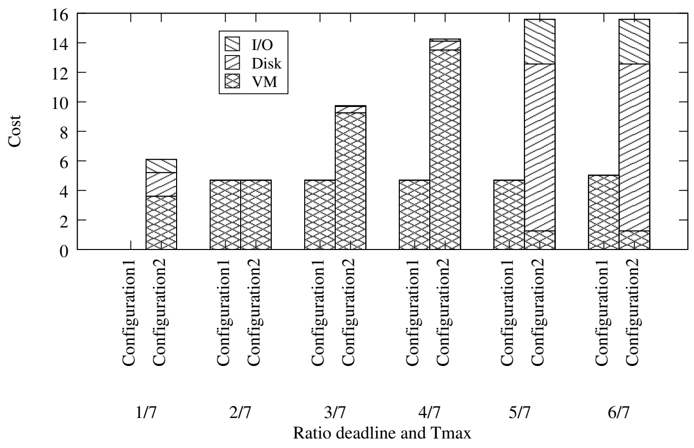

Fig. 5: Fork-Join Execution Cost Distribution

making use of additional storage. By doing that, less tasks are broken to run in different VMs and the existing parallelism can be better explored. However, the parallel execution paths in the Fork-Join DAG has less dependent tasks than in Montage, i.e., there are less dependent task serially in a path. Consequently, using additional storage does not pay off for deadlines up to $4/7$, as can be seen by the larger makespan produced by configuration 2 (Table III). Moreover, under less strict deadline requirements, the advantage of not breaking tasks to run in different VMs and using additional storage takes advantages of existing parallelism and reduces the makespan as in the Montage DAG.

TABLE III: Average makespan of Fork-Join

	Deadline ratio					
	1/7	**2/7**	**3/7**	**4/7**	**5/7**	**6/7**
Configuration 1	-	10.0	8.0	8.0	8.0	8.0
Configuration 2	6.0	12.0	17.2	22.0	2.0	2.0

VI. Conclusion

In this paper, we presented scheduler for the scheduling of data intensive workflows in public clouds. The uniqueness of the proposed scheduler is that it has the ability to choose between the default storage capacity available in virtual machines and extra storage that can be rented. Results shows the scheduler can fulfill more restrict deadlines when additional storage is rented. Indeed, in general, renting additional storage reduces the number of tasks that has to be broken, and in most of the cases it also reduces the makespan since the intrinsic parallelism can be better explored.

In order to optimize the storage usage, we are extending the ILP scheduler to deal with multiple workflows.

Acknowledgment

The authors would like to thank CNPq (156563/2012-6) for financial support, and IBM for providing the CPLEX tools.

References

[1] L. F. Bittencourt and E. R. M. Madeira, "HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207–227, 2011.

[2] L. F. Bittencourt, E. R. Madeira, and N. L. Da Fonseca, "Scheduling in hybrid clouds," *Communications Magazine, IEEE*, vol. 50, no. 9, pp. 42–47, 2012.

[3] G. Juve and E. Deelman, "Scientific workflows and clouds," *Crossroads*, vol. 16, no. 3, pp. 14–18, 2010.

[4] Y. Zhao, X. Fei, I. Raicu, and S. Lu, "Opportunities and challenges in running scientific workflows on the cloud," in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*. IEEE, 2011, pp. 455–462.

[5] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 50.

[6] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi, "Scheduling data-intensiveworkflows onto storage-constrained distributed resources," in *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*. IEEE, 2007, pp. 401–409.

[7] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. IEEE, 2010, pp. 400–407.

[8] E. Barrett, E. Howley, and J. Duggan, "A learning architecture for scheduling workflow applications in the cloud," in *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*. IEEE, 2011, pp. 83–90.

[9] Q. Wu, Y. Zhao *et al.*, "A cost-effective scheduling algorithm for scientific workflows in clouds," in *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International*. IEEE, 2012, pp. 256–265.

[10] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, "Escalonamento de workflows com uso intensivo de dados em nuvens," in *XI Workshop de Computação em Clouds e Aplicações*, 2013.

[11] NASA, "Applications of montage - scientific product generation," http://montage.ipac.caltech.edu/applications.html, 2013, [Online; accessed 05-October-2013].

[12] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[13] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, "iscretização do tempo na utilização de programação linear para o problema de escalonamento de workflows em múltiplos provedores de nuvem," in *XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC, 2012.