Scheduling Service Workflows for Cost Optimization in Hybrid Clouds

Luiz F. Bittencourt, Carlos R. Senna and Edmundo R. M. Madeira
Institute of Computing - University of Campinas (UNICAMP)
P.O. Box 6196, Campinas, São Paulo, Brazil
{bit, crsenna, edmundo}@ic.unicamp.br

***Abstract*—Cloud computing has recently emerged as a convergence of concepts such as cluster computing, grid computing, utility computing, and virtualization. In hybrid clouds, the user has its private cloud available for use, but she can also request new resources to public clouds in a pay-per-use basis when there is an increase in demand. In this scenario it is important to decide when and how to request these new resources to satisfy deadlines and/or to get a reasonable execution time, while minimizing the monetary costs involved. In this paper we propose a strategy to schedule service workflows in a hybrid cloud. The strategy aims at determining which services should use paid resources and what kind of resource should be requested to the cloud in order to minimize costs and meet deadlines. Experiments suggest that the strategy can decrease the execution costs while maintaining reasonable execution times.**

I. Introduction

Cloud computing is nowadays being used for on demand storage and processing power. It allows the leasing of resources to improve the locally available computational capacity when necessary. When using a cloud, the user accesses computing resources as general utilities that can be leased and released [1]. 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.

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. In PaaS the consumers use a hosting environment for their applications. In IaaS the consumer uses computing resources such as processing power and storage. In terms of resources availability, we can classify clouds in three different types:

- **Public clouds**: 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**: Resources that can be accessed and used by individuals inside an organization, similar to 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 users to keep using their private systems (computers, clusters, and grids), aggregating the cloud resources as they 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 [1]. To supply these requirements it is important to the infrastructure to offer reconfiguration with the possibility of the deployment of new resources or update of the existing ones without stopping processes in execution. In our work we combine 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. The similarities between private clouds and private grids allow us to use a grid manager called Grid Process Orchestration [2] inside our private cloud. To execute service workflows in hybrid systems we use our hybrid infrastructure [3], which provides dynamic instantiation of services.

In this paper we propose a strategy to schedule workflows in hybrid cloud systems. The objective of the algorithm is to meet a deadline reducing the execution costs by requesting as fewer resources as possible to the public cloud. We deployed a hybrid system to evaluate the execution of an image processing application using the proposed algorithm, and we compared the resulting execution times and costs with executions in the private cloud only and in the public cloud only.

II. The Scheduling Algorithm

The workflow scheduling in hybrid cloud systems is a novel research challenge that comes together with the merging of private and public clouds. Only a few works considers a hybrid environment. In [4], the authors propose a scheduling strategy to use resources from a utility grid when locally available resources are not sufficient to execute the application. However, they do not consider workflows in the scheduling. In [5] the authors propose a hybrid system formed by the DIET Grid and Eucalyptus, showing some possible ways of connecting these two architectures but without support to services or service workflows. In [6] the authors show issues that limit the use of clouds for highly distributed applications in a hybrid system. However, it has lack of interoperability between different cloud platforms, and it does not offer support to service workflows.

This is a pre-print version.
The final version is available at the publisher's website.

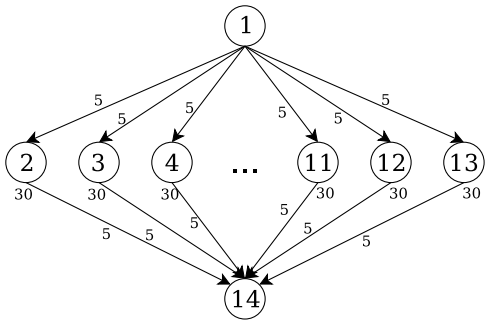


Fig. 1. Example of fork-join DAG with 14 nodes

The scheduling algorithm proposed here has the objective of reducing the makespan maintaining a reasonable cost. While executing every service of the workflow locally may delay the execution, on the other hand, executing all services in the cloud may result in prohibitive costs. Thus, the algorithm tries to balance the use of private resources with the ones available from the public cloud in a pay-per-use basis.

A. Background

A workflow is commonly represented by a Directed Acyclic Graph (DAG) $G = (V, E)$, where each node $n_i \in V$ represents a service and each edge $e_{i,j} \in E$ represents a data dependency among services i and j. We developed a service to apply a median filter in an image file, which is split, processed in parallel, and merged. The median filter application can be represented by a fork-join DAG, as shown in Figure 1. This example shows a 14 node DAG, where node 1 is the slice operation, nodes 2 to 13 are the median filter application, and node 14 is the merge operation. 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).

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 [7]. This initial schedule considers only the private resources 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 services to be scheduled in the public cloud. The algorithm uses some attributes computed for each node of the DAG:

• **Computation cost:**

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

$w_{i,r}$ represents the computation cost (time to execute the node) 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 $c_{i,j} = 0$.

• $\mathbf{suc(n_i)}$ and $\mathbf{pred(n_i)}$ are the sets of immediate successors and predecessors of node n_i in the DAG.

• **Priority:**

$$\mathcal{P}_i = \begin{cases} w_{i,r}, & \text{if } i \text{ has no successors} \\ w_{i,r} + \max\limits_{\forall\ n_j \in\ suc(n_i)} (c_{i,j} + \mathcal{P}_j), & \text{otherwise} \end{cases}$$

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

• **Earliest Start Time:**

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

where $ST_i = \max\limits_{\forall\ n_h \in\ pred(n_i)} (EST(n_h, r_k) + w_{h,k} + c_{h,i})$. $EST(n_i, r_k)$ 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 that are in the same path in the DAG. Such groups, called clusters of nodes, are scheduled in the same resource. Details about the PCH are in [7].

B. Scheduling in Hybrid Clouds

After the initial scheduling made by PCH, the algorithm checks if resources from the public cloud are needed based on the deadline $\mathcal{D}$. If the makespan of the schedule 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 $\mathcal{P}_i + EST_i$ 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 first line of Algorithm 1 makes the initial schedule using the PCH and considering only resources in the private cloud. If the deadline is not met (line 2), 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 $\mathcal{P}_i + EST_i$ is maximum and n_i is currently scheduled in the private cloud (line 5). In line 6, node n_i is added to the set $\mathcal{T}$, which is composed

Algorithm 1 Scheduling in Hybrid Clouds

1: Schedule G in the private cloud using PCH
2: $\mathcal{R}$ = all resources in the private cloud
3: **if** $makespan(G) > deadline(G)$ **then**
4: **while** $makespan(G) > \mathcal{D}$ AND $iteration < size(G)$ **do**
5: $iteration = iteration + 1$
6: select node n_i such that $\mathcal{P}_i + EST_i$ is maximum and n_i is currently scheduled in the private cloud
7: $\mathcal{T} = \mathcal{T} \cup t$
8: $num\_clusters$ = number of clusters of nodes in $\mathcal{T}$
9: **while** $num\_clusters > 0$ **do**
10: Select resource r_i from the public clouds such that $\frac{price_i}{num\_cores_i \times p_i}$ is minimum and $num\_cores_i <= num\_clusters$
11: $\mathcal{R} = \mathcal{R} \cup r_i$
12: $num\_clusters = num\_clusters - num\_cores_i$
13: **end while**
14: **for all** $n_i \in \mathcal{T}$ **do**
15: Schedule n_i in $r_n \in \mathcal{R}$ such that EFT_i is minimum
16: Recalculate $ESTs$ and $EFTs$
17: **end for**
18: **end while**
19: **end if**

of all nodes being rescheduled from the private cloud to the public cloud. In line 7, 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 private 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 resources pool $\mathcal{R}$. With the resources selected, the algorithm schedules each node n_i in $\mathcal{T}$ in the resource $r_n \in \mathcal{R}$ which results in the smallest EFT_i.

Figure 2 shows an example considering the DAG of Figure 1 and the resources in Tables I and II with $\mathcal{D} = 45$. The initial schedule considering only the private cloud resources is on the left hand side of Figure 1. It allocates services 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 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 services $14, 12, 13, 10, 11, 8, 9, 7$, and 6, in this order. The scenario where only resources from the public clouds are considered results in a schedule with $makespan = 35$ and $cost = (35 \times 0.9) + (20 * 0.52) = 41.9$, using 8 cores during 35 time units plus 4 cores during 20 time units.

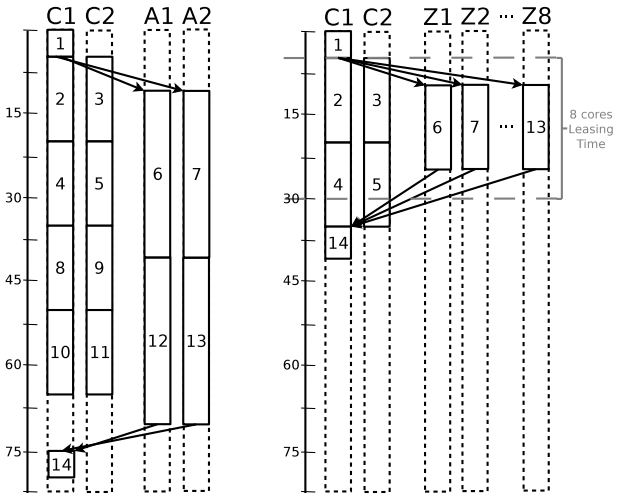
Note that the algorithm deals with a deadline (or desired execution time), but it can be easily adapted to deal with budget instead of deadlines.

TABLE I
RESOURCES IN THE PRIVATE CLOUD.

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

TABLE II
RESOURCES IN THE PUBLIC CLOUD.

Type	Cores	RAM	p_r per core	Cost per time unit
Y	1	1Gb	1.5	\$0.14
Y	2	1Gb	1.5	\$0.25
Z	1	2Gb	2	\$0.17
Z	2	4Gb	2	\$0.3
Z	3	6Gb	2	\$0.4
Z	4	8Gb	2	\$0.52
Z	8	16Gb	2	\$0.9



(a) Private cloud only (b) Hybrid cloud

Fig. 2. Example of scheduling in the private and hybrid clouds.

III. EXPERIMENTAL RESULTS

We evaluated the proposed algorithm in the resources shown in Tables I and II with the median filter workflow using different matrix and filter sizes, different number of slices and different $\mathcal{D}$ values. We measured the execution time (makespan) and cost for executing the workflows, computing the average over 3 executions. All results show 95% confidence intervals.

The first set of executions was run with $5,000 \times 5,000$ matrices and $\mathcal{D} = 20$ for 3×3 filter and $\mathcal{D} = 30$ for 5×5 filter. Figure III 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 execution considering only resources from the public cloud can meet the deadline with a very low makespan, however the cost for the execution is high. The execution in

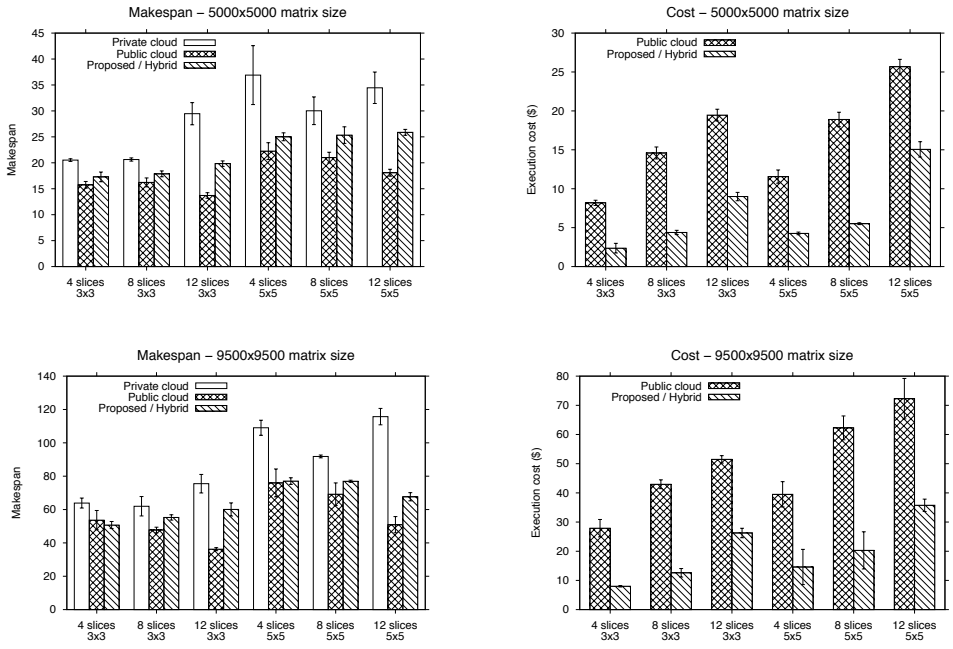


Fig. 3. Results for matrices of size $5,000 \times 5,000$ and $9,500 \times 9,500$

the hybrid system with the proposed algorithm can meet the deadline with costs 2 to 4 times lower than costs using only the public cloud resources. With $9,500 \times 9,500$ matrices and $\mathcal{D} = 62$ for 3×3 filter and $\mathcal{D} = 80$ for 5×5 filter, the hybrid execution performed better with 4 slices, maintaining a lower cost. With 8 and 12 slices, the hybrid execution meets the deadline with costs 2 to 3 times lower.

From the executions in our cloud testbed we observe that the proposed algorithm can reduce the makespan when compared to the local execution, as well as the cost when compared to the execution in the public cloud. 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 all 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.

IV. Conclusion

The cloud computing paradigm is being widely used for the execution of many types of applications, including ones with data dependencies, which can be represented by workflows. To execute such workflow applications in a hybrid cloud, the scheduling algorithm must take into consideration costs and the execution time. In this paper we propose an approach to schedule service workflows that consider a deadline and execution costs. We deployed a hybrid cloud that offers support for automatic service installation in the resources dynamically provided by the grid or by the cloud to execute the proposed algorithm. We also compare it with executions in the private cloud only and in the public cloud only. The experimental results show that the algorithm is capable of scheduling fork-join workflows in a real cloud testbed reducing its execution costs and meeting the desired execution times.

Future works include the introduction of budget with deadlines, developing a bi-criteria scheduling for hybrid clouds.

Acknowledgment

We would like to thank CAPES, FINEP, FAPESP (09/15008-1), and CNPq (472810/2006-5 and 142574/2007-4) for the financial support.

References

[1] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *J. Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[2] C. R. Senna and E. R. M. Madeira, "A middleware for instrument and service orchestration in computational grids," in *7th IEEE International Symposium on Cluster Computing and the Grid CCGrid 2007*, Rio de Janeiro, Brazil, 2007.

[3] L. F. Bittencourt, C. R. Senna, and E. R. M. Madeira, "Enabling execution of service workflows in grid/cloud hybrid systems," in *IEEE International Workshop on Cloud Management (Cloudman 2010)*, Osaka, Japan, 2010.

[4] P. D. M. Jr., F. de Figueiredo, D. Maia, F. V. Brasileiro, and A. Coelho, "On the planning of a hybrid it infrastructure," in *Network Operations and Management Symposium*, Salvador, Brazil, 2008, pp. 496–503.

[5] E. Caron, F. Desprez, D. Loureiro, and A. Muresan, "Cloud computing resource management through a grid middleware: A case study with diet and eucalyptus," Bangalore, India, 2009, pp. 151–154.

[6] S. Jha, A. Merzky, and G. Fox, "Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes," *Concurr. Comput. : Pract. Exper.*, vol. 21, no. 8, pp. 1087–1108, 2009.

[7] L. F. Bittencourt and E. R. M. Madeira, "A performance-oriented adaptive scheduler for dependent tasks on grids," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 9, pp. 1029–1049, 2008.