Workflow Specification and Scheduling with Security Constraints in Hybrid Clouds

Daniel S. Marcon*, Luiz F. Bittencourt†, Ramide Dantas‡, Miguel C. Neves*, Edmundo R. M. Madeira†,
Stênio Fernandes§, Carlos A. Kamienski¶, Marinho P. Barcelos*, Luciano P. Gaspary*, Nelson L. S. da Fonseca†
*Institute of Informatics, Federal University of Rio Grande do Sul – Av. Bento Gonçalves, 9500, Porto Alegre - RS - Brazil
{daniel.stefani, mcneves, marinho, paschoal}@inf.ufrgs.br
†Institute of Computing, University of Campinas – Av. Albert Einstein, 1251, Campinas - SP - Brazil
{bit, edmundo, nfonseca}@ic.unicamp.br
‡Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco – Av. Prof Luiz Freire, 500, CDU - Recife - PE - Brazil
ramide@ieee.org
§Federal University of Pernambuco – Avenida Jornalista Aníbal Fernandes, Recife - PE - Brazil
sflf@cin.ufpe.br
¶Federal University of the ABC – Rua Catequese, 242, Santo André - SP - Brazil
cak@ufabc.edu.br

***Abstract*—Hybrid cloud management must deal with resources from both public and private clouds, as well as their interaction. When workflows are executed in a hybrid cloud, dependencies among their components bring new factors to be considered during specification, scheduling, and virtual machine provisioning. In this paper, we describe three components, namely workflow code, scheduler, and resource allocator, which enable the specification and execution of workflows in hybrid clouds in the context of the AltoStratus middleware. We present a case study that shows the interaction among these components, and their applicability in practice.**

I. Introduction

The use of Cloud Computing brings challenges to large enterprises, especially those that operate their own datacenters. These organizations would like to take advantage of cloud features (such as dynamic allocation of computing resources [1]) using their previous investment on infrastructure. In this context, security and privacy concerns [2], [3] tend to exist around fewer aspects since data is kept in the organizations' premises and not in the cloud provider machines. Such cloud infrastructures are called private clouds. They can be seen as an alternative to public cloud providers, which consist of public and shared on-demand pay-per-use virtualized infrastructures.

Hybrid cloud solutions attempt to get the best of both worlds, public and private clouds [4]. Using a hybrid cloud, enterprises can employ their local resources to have a cloud-like experience and still be able to use computing power from public providers to accommodate transient demand surges, for example. However, the hybrid solution imposes its own challenges [5]. In this context, the AltoStratus project[1] aims to propose middleware techniques and mechanisms for composing, executing, and managing services in hybrid and heterogeneous computational cloud environments.

Deploying a new service or business process (workflow) on a hybrid cloud raises the question of how to make the best use of local resources while meeting performance and security requirements of services and minimizing costs [6]. In this paper, we address such problem by proposing an integrated solution. In the proposed approach, the scheduler decides, for each workflow, the set of tasks that will run in which platform (private or public), with the goal of minimizing tenant cost while meeting the workflow deadline. After determining the amount of computing and network resources needed by the workflow, the scheduler sends this information to a resource allocation component, which will map the request onto the cloud infrastructure taking security and performance criteria into account. Therefore, the main contribution of this work is to reconcile efficient scheduling and allocation of workflows in hybrid clouds and the fulfillment of security requirements.

The remainder of the paper is organized as follows. Section II briefly describes AltoStratus. Section III introduces the proposed approach to schedule and allocate workflows in a hybrid cloud infrastructure according to performance and security criteria, while its evaluation is presented in Section IV. Section V discusses related work, and Section VI concludes the paper.

II. AltoStratus Middleware

In this section we describe the AltoStratus middleware, presenting its components and the interactions that the resource allocation components, which are the focus of this paper, need to perform.

AltoStratus is a middleware platform composed of two layers (see Figure 1). The application developer interacts with the bottommost layer, called workflow composition. This layer provides a GUI for the creation and manipulation of service workflows, allowing the specification of functional and non-functional requirements that need to be met. Via the GUI, workflows can be created directly by choosing the set of concrete services that compose them. The developer also has the option of creating semantic workflows based on abstract tasks instead of concrete services. The set of concrete services that implement these tasks are selected automatically at a later

[1]http://www.dimap.ufrn.br/altostratus/english/

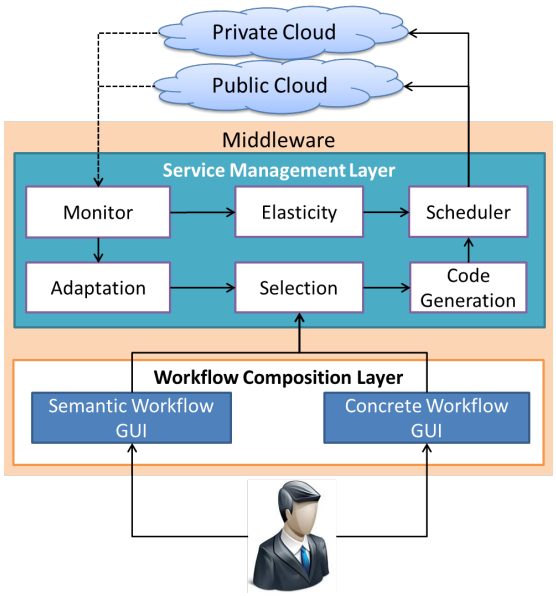

Fig. 1. AltoStratus architecture.

stage, according to non-functional requirements provided by the developer.

The second layer (service management) consists of six main components: service monitor, adaptation, elasticity, selection, scheduling (including security), and code generation. In addition to QoS monitoring of deployed services, these components are responsible for the composition, selection of services and adaptation of workflows generated by the modules of the previous layer.

The service management layer must also handle security and resource management among different cloud platforms. Security is taken into account when allocating resources in the cloud infrastructure (more details are presented in Section III). Resource management considers the creation and manipulation of VMs located in different platforms, as well as elasticity and load balancing among these VMs.

In general, AltoStratus has modules to enable several kinds of interactions, such as high-level ones (for instance, directly with tenants) as well as low-level ones (resource allocation in the infrastructure). The main components of the architecture are discussed next.

a) AltoStratus GUI: AltoStratus graphical interface enables the developer to define new service compositions (or workflows) either explicitly, by informing each concrete service, or via abstract tasks. In the latter, concrete services that match the desired tasks are selected based on their semantic descriptions in the next stage. The composition can also be partially defined, letting an automatic composition algorithm complete the specification. The output of this stage is a semantic specification, which serves as input to the selection module.

b) Selection: The semantic workflow needs to be interpreted and the services that satisfy each workflow component must be identified to generate the execution plan. This task is accomplished by the selection module, which is thus responsible for choosing the specific set of services that will form the execution plan (semantic language specification) in accordance with the semantic workflow. Several alternative plans can exist for the same input specification; the selection module chooses the one that matches the workflow's requirements, which is then transferred to the code generation module.

c) Code Generation: This module performs the translation of the execution plan created by the selection module into an executable service code. This code is packaged together with other information about the new service (and any subservices in which it depends on) to form the Workflow Code (WC).

d) Scheduler: Before the execution of the Workflow Code, the scheduler obtains information about running services from the resource monitor and chooses where the services specified in the Workflow Code should be deployed based on the workflow quality of service requirements.

e) Monitor: This module checks if the established QoS for the services deployed are being served. In case a problem with a virtual machine is detected, or if the QoS level is below the specified threshold, it determines which modules (elasticity and/or adaptation) should be called to solve this issue (according to the policy associated with the service).

f) Elasticity: The elasticity module scales in or scales out the amount of resources allocated for a given service in accordance with the service QoS parameters. Toward this end, it verifies if it is feasible to instantiate new VMs according to elasticity policies contained in the WC and, if so, it calls the scheduling module. This module also turns off virtual machines when they are not used (i.e., when cloud demand is low), to reduce costs. Note that the decision to shut down virtual machines is controlled by service policies.

g) Adaptation: When elasticity fails (if elasticity is not possible), or if the policy specifies that the elasticity module should not be called, the adaptation module seeks alternative services in the hybrid cloud to be used by the workflow to meet the specified QoS requirements. In fact, it triggers the selection module to get new execution plans for the workflow. While the elasticity module attempts to solve the QoS problem at the infrastructure level, the adaptation module selects an alternative set of services to increase QoS.

III. Workflow Deployment and Scheduling

For a workflow to be submitted to run in the hybrid cloud, several management tasks must be performed. In this paper, we are concerned with the following three steps:

1) Workflow specification: defines the necessary services and their dependencies;
2) Workflow scheduling: defines where (i.e., in which virtual machine instance) each service from the composition will run;
3) Virtual machine allocation: defines where (i.e., in which physical machine) each virtual machine instance specified by the scheduler will run.

The set of AltoStratus components that are responsible for these tasks in the hybrid cloud platform are the workflow code, code generator, scheduler and resource allocator, which is part of the scheduler module.

The semantic workflow is submitted to AltoStratus, which is responsible for allocating, executing, and monitoring the workflow. Prior to its deployment, however, the semantic workflow must be converted into executable code. The result

of this process is the Workflow Code (WC), based on the Service Code (described in Section II). The WC contains an executable workflow (BPEL script), which is the translation of the execution plan generated by the selection module.

The initial step required for running a concrete workflow is the specification of resources needed and where each service is (or will be) located in the cloud platform. This task should be performed by the scheduling and resource allocation module, according to the workflow QoS requirements. In this sense, for each component of the service workflow, we envision three possible scenarios:

1) The scheduler chooses a service, which is already up and running, to be invoked for a workflow component;
2) The scheduler decides that a new service must be created in a free virtual machine instance that is already running;
3) The scheduler decides that the best virtual machine configuration for the service is one that does not have any free instance running.

In the first case, the workflow component is just invoked and, from the allocation viewpoint, no further action is required, since the service is already up in an instantiated virtual machine. In the second case, there is no need to allocate a virtual machine, but it is necessary to deploy the service to be invoked, which is performed by AltoStratus. In the third case, the virtual machine and the service must both be instantiated. In particular, the VM is allocated in a physical machine considering performance and security aspects. We discuss the scheduling and VM allocation, as well as their interaction, in more details next.

A. Scheduler

This component is responsible for choosing where each workflow service must be deployed to execute. The role of the scheduler is twofold. First, it decides which cloud infrastructure will be used for each service: public or private platform. Since the private cloud potentially lower cost, this is the primary option if there are enough available resources. Second, it defines the amount of resources necessary to run each service, i.e., the type of VM instance in which the service will run. The decision of the scheduler is based on user requirements of QoS and on service-level agreements (SLAs) signed with the cloud provider.

In particular, the scheduler can be called in three different situations: (i) when a new workflow must be scheduled and allocated in the cloud platform; (ii) when some (or all) components should be rescheduled; or (iii) when additional VMs must be allocated to guarantee the specified QoS parameters. In any of these cases, a request to the scheduler is composed of the maximum execution time (deadline) of the workflow, and additional information about the services that compose this workflow (such as previously monitored execution times of these services). With such information, and the knowledge of the cloud infrastructure, the scheduler can estimate service execution time on different types of VMs as well as tenant cost.

Each workflow application is represented by a directed acyclic graph (DAG) G = (V, E), where V represents the set of tasks and E the set of dependencies among tasks. The scheduling results in a mapping between the set of workflow services (nodes in the DAG) and the set of computing resources (associated with a VM type, instantiated or not). That is, the scheduler determines which virtual machine configuration is adequate to run each workflow service in order to minimize costs and meet the deadline. Therefore, the next step is to allocate resources on the physical substrate. Toward this end, the scheduler calls the resource allocation with the following parameters for each workflow application: (i) a set of VMs; (ii) the bandwidth needed by these VMs to communicate with each other; (iii) and mutual trust relationships between tenants. Next, we describe how the resource allocator utilizes these trust relationships to allocate VMs to physical resources.

B. Resource Allocator

This module is part of the scheduler. It uses trust relationships between tenants in order to increase security in the hybrid cloud platform while fulfilling network QoS requirements. Such relationships denote whether application a_i from one tenant trusts application a_j from another tenant. These relationships are direct, binary, and symmetric (i.e., a tenant may either trust or not another tenant, with whom he interacts; if there is trust, then it is reciprocal). They can be established based on the web of trust concept [7] or by matching properties contained within SLAs signed by different customers and providers (which would be assisted by the front-end responsible for receiving the requests and transferring them to the allocation module).

The allocation process is divided into two steps: functions $\mathcal{F}$ and $\mathcal{G}$. The former distributes and maps applications into virtual infrastructures[2] (VIs) according to trust relationships, while the latter allocates virtual infrastructures onto the physical substrate. In fact, each function has different objectives. While function $\mathcal{F}$ maximizes the number of mutually trusted relationships between tenants in the same VI, function $\mathcal{G}$ minimizes the amount of network resources required for the allocation of each VI in the physical substrate. These two objectives, combined, can increase security for tenants while maintaining high and efficient resource utilization for providers. We provide a detailed description of the allocation process in Marcon et al. [3].

C. Interaction between workflow scheduling and resource allocation

The scheduler is responsible for receiving application requests, with computing and networking demands, and to define in which type of VM each workflow component will be executed; the resource allocator, then, determines physical servers that can hold these VMs, as well as the set of links to be used for communication among such VMs.

We need to define in which occasions the scheduler must call the resource allocator. Figure 2 shows an overview of the interactions (and inputs/outputs) of the modules discussed in this section. Upon receiving the workflow, the scheduler must then decide where each workflow component will be executed. In order to do so, it requires two sets of information: the concrete workflow specification; and the cloud infrastructure specification.

[2]The term virtual infrastructure is used to represent a logically isolated domain (i.e., a set of virtual machines as well as the virtual network interconnecting them).

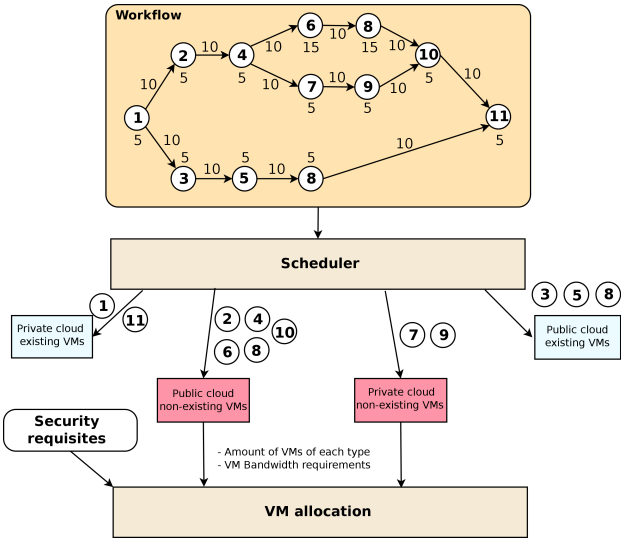

Fig. 2. Workflow scheduling and resource allocation.

As previously defined, a workflow is represented by a DAG, with information about tasks and data dependencies. The hybrid cloud infrastructure, in turn, includes information from the private cloud (i.e., the set of available heterogeneous resources and the network that interconnects such resources), and information from the public cloud platforms (computing and network resources and the price of such resources).

This fine-grained information allows the scheduler to optimize the amount of resources allocated to each workflow. In fact, the scheduler aims at minimizing monetary costs of tenants while maintaining the deadline specified for the workflow.

The scheduling process results in an association of workflow tasks (or nodes in the DAG) with computing resources, where each computing resource is associated to a type of virtual machine to be used. That is, the scheduler determines which virtual machine configuration is sufficient to perform each task in the workflow, in order to minimize costs and meet the deadline specified by the user.

The scheduling output can direct workflow components for VMs in four different states (Figure 2): i) VMs that are already allocated and running in the private cloud; ii) unallocated VMs at the private cloud; iii) already allocated VMs in the public cloud; and iv) unallocated VMs in the public cloud.

Workflow tasks scheduled to run in already deployed VMs (both in private or public clouds) do not require interaction with the resource allocator, as their VMs are already running on a given server. In contrast, if the scheduler decides that new, unallocated VMs are necessary, the next step is to determine in which physical machines such VMs can run in order to provide QoS guarantees. Thus, each workflow submission is scheduled independently, and the scheduler sends to the resource allocator a list of VMs to be created, the amount of bandwidth needed by these VMs to communicate with each other and trust relationships between tenants. Then, the resource allocator maps such requests (computing and network resources) in the cloud infrastructure, in a process composed of functions $\mathcal{F}$ and $\mathcal{G}$, as previously discussed.

In general, the scheduler operates in the allocation decision of each workflow independently. It can be inserted in a personal/local context of the hybrid cloud tenant. Each user, in turn, relies on other users, which reflects the trust between them. Finally, the resource allocation strategy (based on trust relationships between tenants) uses the information in the workflow scheduler to allocate resources, with the purpose of mitigating selfish and malicious attacks in the intra-cloud network (e.g., the consumption of an unfair share of the network to complete tasks in a shorter amount of time).

IV. Evaluation

We have developed a simulator that models a hybrid cloud platform, in order to evaluate the integration between the scheduling and resource allocation module.

A. Methodology

A set of DAGs is first scheduled independently (according to the process described in Bittencourt and Madeira [6]) and, then, the output is transferred to the resource allocator.

The set of DAGs used in the simulations were: Montage [8], AIRSN [9], CSTEM [10] and MDP [11]. In fact, Montage was generalized, resulting in DAGs with size in the interval [5,100] nodes. Additionally, random DAGs with a size between 5 and 100 nodes were generated using the method described in Sakellariou and Zhao [12]. Each task has a computing cost in the interval (500,4000). For each type of DAG, we performed simulations using communication to computation ratio $CCR = 1.0$.

The environment was configured as two clouds: one private and one public, in which resources are allocated by the resource allocator. The private cloud platform $\mathcal{R}$ has a total amount of resources in the interval [2, 10]. Each resource has its processing capacity obtained from a uniform distribution in the interval (10,100). Each simulation was repeated 500 times.

The hybrid platform links are classified in three types: internal links of the private cloud, internal links of the public cloud and links that connect these platforms. Each link type has an associated capacity: (100, 100, 10), (100, 500, 10) and (100, 1000, 10), where the elements correspond to the bandwidth within the private cloud, the bandwidth within the public cloud and the bandwidth that connects the private and public clouds, respectively.

VM processing capacity was fixed, in each simulation, in the interval (50,150). Without loss of generality, we follow prior work [1] and assume that all VMs are equal, considering only virtual machines with one processing core. Additionally, each workflow has a maximum deadline $\mathcal{D}$, in $2.5 \times CP$, $3.5 \times CP$ and $4.0 \times CP$, where CP is the execution time of tasks in the critical path of the DAG at the best available resource.

Scheduling each workflow results in a set of VMs to be allocated in the public cloud in case the private cloud does not have enough resources (Figure 2). The scheduling process generates, for each workflow, the amount of resources that need to be allocated in the public cloud platform and calls the resource allocator. Each request is associated with an identifier, the number of VMs needed, the bandwidth needed by communication between VMs of the same workflow and the amount of time necessary for the execution. Moreover, the resource allocator also receives mutual trust relationships

between tenants. They were generated through direct relationships between users in a random graph with degree of each vertex (tenant) following a distribution $P(k) \propto \frac{1}{k}$.

Resource allocation evaluation in the public cloud focuses on the trade-off between the gain in security for tenants and the cost (internal resource fragmentation) it imposes on cloud providers. Following related work [1], [2], [13], [14], the cloud physical substrate was defined as a three-level multi-rooted tree topology. It consists of 1,020 servers at level 0, each with 4 VM slots (i.e., 4,080 VMs in the cloud infrastructure). Each virtual infrastructure was defined as a tree-like topology with similar size in comparison to the other VIs.

B. Simulation Results

Improved security for tenants. Security is quantified by measuring the number of mutually untrusted tenants assigned to the same VI. Therefore, the objective is to minimize this value, as it represents how exposed applications are to attacks. We verify trust relationships between tenants in two scenarios: when allocating application batches (i.e., when all application requests are known beforehand), and when applications arrive without prior knowledge (in an online setting). The results are compared with the baseline scenario (current cloud allocation scheme), in which all tenants share the same network.

Figure 3 shows the number of mutually untrusted relationships according to the number of VIs offered by the provider. The results show that the number of workflows allocated is not the main factor to increase security, but rather the number of VIs offered by the provider. In general, we find that the number of mutually untrusted relationships decreases, and therefore security increases, logarithmically according to the number of VIs.

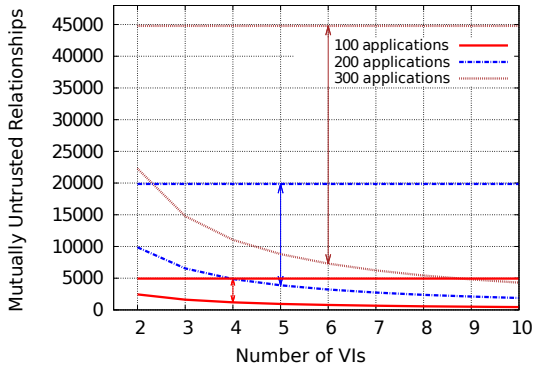

Fig. 3. Security when allocating workflow batches.

We also measure how security increases when workflow requests arrive without prior knowledge. The arrival rate is given by a Poisson distribution with a mean of 12 requests every 100 time units. Figure 4 depicts how the number of mutually untrusted relationships varies during 14,000 time units. We see that the greater the isolation among tenant applications, the higher the security, since the number of mutually trusted relationships inside each VI is maximized. However, the level of security offered by the provider tends to stabilize after a certain number of VIs, as security increases logarithmically.

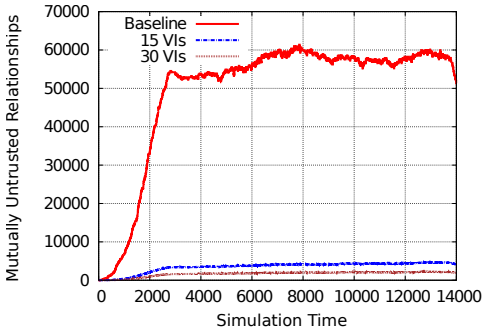

Fig. 4. Security in an online setting.

Low resource fragmentation. The creation of VIs and grouping of workflows may restrain the allocation process because of internal fragmentation of resources (i.e., the cloud might have enough available resources to allocate an incoming request, but no VI alone has the necessary amount of resources to accept the request). We adopt a simple admission control, similar to Amazon EC2 [15], which rejects an application if it cannot be allocated upon its arrival.

Figure 5 shows the acceptance ratio of requests according to the cloud load. Workflow requests begin to be rejected when the load is around 96% for 15 VIs and 95% for 30 VIs. The acceptance ratio is significantly reduced only when the load goes over 97%. Although this is a side effect of our approach, we consider that this cost is pragmatically negligible, due to the fact that public cloud platforms (such as Amazon EC2) operate their datacenters with 70-80% occupancy [16].

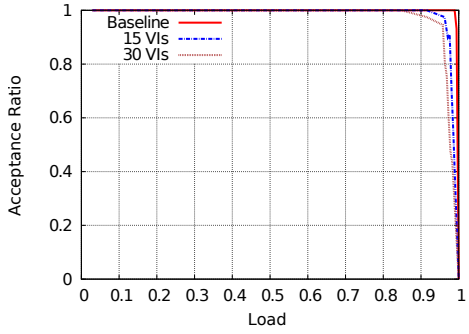

Fig. 5. Acceptance ratio according to the cloud load.

Figure 6, in turn, depicts the overall acceptance ratio of requests. Initially, all requests are accepted because of the high amount of available resources in the cloud. As time passes and the load increases, some requests are rejected, resulting in a convergence period. Since rejections occur at different loads for different settings (as shown in Figure 5), the best scheme is not clear. Finally, the acceptance ratio stabilizes around 52%, showing a negligible overhead (acceptance 0.5% lower) when compared to the baseline. Thus, we conclude that it is possible to increase security with minimal addition of rejected requests.

V. Related Work

In this section, we highlight some relevant work in the field of workflow management and scheduling, as well as in

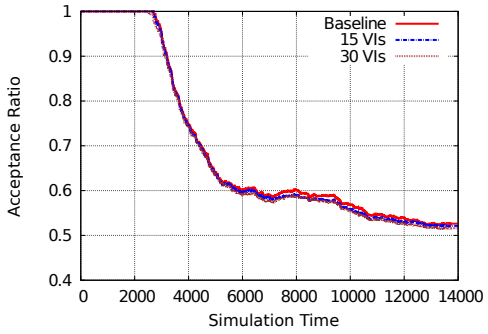

Fig. 6. Overall cloud acceptance ratio.

network sharing and security.

Workflows. Workflow management and scheduling have been widely studied [8]. In utility grids and hybrid clouds, the scheduling problem has been tackled considering processing capacity and monetary costs [4], [11], [17]. Yu et al. [11] proposed a deadline-driven cost-minimization algorithm. The Deadline-MDP (Markov Decision Process) algorithm breaks the DAG into partitions, assigning a maximum finish time for each partition according to the deadline given by the user. Abrishami et al. present the Partial Critical Paths (PCP) algorithm [18]. The PCP algorithm schedules the workflow backwards, searching for not scheduled partial critical paths. Constraints are added to the scheduling process when the scheduling of a partial critical path fails, starting the scheduling process again. This algorithm presents the same characteristics of MDP, however with higher time complexity, since a combinatorial number of re-schedulings may occur. A self-adaptive global search optimization technique called Particle Swarm Optimization (PSO) is utilized to schedule workflows in the algorithm proposed by Pandey et al. [17]. The algorithm was developed to work in cloud computing environments with one-level SLAs and on-demand resource leasing. It does not consider multi-core resources nor workflow deadlines, focusing solely on the monetary cost minimization. None of these algorithms consider security issues, which led us to the proposed approach for integrating the scheduling and the security into a module in the AltoStratus project.

Network sharing and security. Recent proposals try to increase network guarantees and security for tenants by designing network-aware allocation algorithms. These approaches can be divided in two main classes: i) proportional sharing [2], [19], which focus on providing fair network sharing at flow-level in accordance with weights assigned to tenants; and ii) network virtualization [1], [13], [20], which isolates one tenant (or application) per virtual network. The former, however, adds substantial management overhead to control how network resources are shared (since large-scale cloud platforms can have over 10 million flows per second [21]), while the latter results in low utilization of resources (because they statically reserve bandwidth according to the application peak demand).

VI. Conclusion

The use of hybrid cloud platforms requires an optimized and efficient scheme to utilize resources from both private and public clouds, in a way that minimizes tenant cost and meets workflow requirements (in particular, performance and security). To this end, we have proposed a strategy composed of two steps: the scheduler and resource allocator. The former receives workflow requests and decides the amount of resources needed by each workflow task and in which platform each task will run. The latter maps requests onto the cloud infrastructure according to performance and security criteria. In future work, we intend to consider other requirements when scheduling and allocating workflows, such as energy consumption and resiliency.

Acknowledgment

We would like to thank FINEP, CTIC/RNP and CNPq (through the SecFuNet project) for the financial support.

References

[1] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM*, 2011.
[2] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *USENIX NSDI*, 2011.
[3] D. S. Marcon, R. R. Oliveira, M. C. Neves, L. S. Buriol, L. P. Gaspary, and M. P. Barcellos, "Trust-based grouping for cloud datacenters: improving security in shared infrastructures," in *IFIP TC6 Networking*, 2013.
[4] L. F. Bittencourt, E. R. M. Madeira, and N. L. S. Da Fonseca, "Scheduling in hybrid clouds," *IEEE Communications Magazine*, vol. 50, no. 9, pp. 42 –47, September 2012.
[5] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. De Turck, "Network-aware impact determination algorithms for service workflow deployment in hybrid clouds," in *CNSM*, 2012.
[6] 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.*, 2011.
[7] P. R. Zimmermann, *The official PGP user's guide*. Cambridge, MA, USA: MIT Press, 1995.
[8] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.
[9] Y. Zhao, J. Dobson, I. Foster, L. Moreau, and M. Wilde, "A notation and system for expressing and executing cleanly typed workflows on messy scientific data," *SIGMOD Record*, vol. 34, no. 3, pp. 37–43, 2005.
[10] A. Dogan and F. Özgüner, "Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems," *Computer Journal*, vol. 48, no. 3, pp. 300–314, 2005.
[11] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *e-Science and Grid Computing*, 2005.
[12] R. Sakellariou and H. Zhao, "A hybrid heuristic for dag scheduling on heterogeneous systems." in *IEEE IPDPS Workshops*, 2004.
[13] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *ACM CoNEXT*, 2010.
[14] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in *ACM SIGCOMM*, 2012.
[15] "Amazon ec2 api ec2-run-instances," Amazon, 2013, available at : http://goo.gl/8S97La.
[16] R. Bias, "Amazon's EC2 Generating 220M," Cloudscaling, 2011. [Online]. Available: http://goo.gl/d1Vai.
[17] S. Pandey, L. Wu, S. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *IEEE AINA*, 2010.
[18] S. Abrishami, M. Naghibzadeh, and D. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," in *IEEE/ACM GRID*, 2010.
[19] V. T. Lam, S. Radhakrishnan, R. Pan, A. Vahdat, and G. Varghese, "Netshare and stochastic netshare: predictable bandwidth allocation for data centers," *ACM SIGCOMM CCR*, vol. 42, no. 3.
[20] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "CloudNaaS: a cloud networking platform for enterprise applications," in *ACM SoCC*, 2011.
[21] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM IMC*, 2010.