Service Workflow Monitoring in Private Clouds: The User Point of View

Carlos R. Senna, Luiz F. Bittencourt, and Edmundo R. M. Madeira
Institute of Computing - University of Campinas (UNICAMP)
Av. Albert Einstein, 1251, 13083-852, Campinas, São Paulo, Brazil
{crsenna, bit, edmundo}@ic.unicamp.br

Abstract—**Cloud computing environments offer, in general, tools and mechanisms for the user to monitor computational, storage, and networking resources. However, only a few middlewares for cloud computing provide tools that help the user in the application monitoring. When we consider workflows and their dependencies, monitoring from the user point of view is not found in the literature of cloud middlewares. In this paper, we present an infrastructure for execution of service workflows in private clouds with tools that allow the user to keep track of his/her workflows, measuring each activity performed in the cloud resources. Through this monitoring, our workflow manager is able to react during the workflow execution, requesting new or better resources in order to the execution to comply with the application quality of service requirements. Besides that, the metrics that our monitoring system offers allow users to keep a detailed track of the organization internal costs involved in the processing, and then use this information to evaluate other alternatives, such as leasing resources from public cloud providers.**

I. Introduction

The cloud provider must have the ability of feeding the user with computational resources on-demand, presenting characteristics that include the client auto-servicing, ubiquitous access, availability, elasticity, and pay-per-use. Three basic service levels exist in this paradigm: sotware as a service (SaaS), platform as a service (SaaS), and infrastructure as a service (IaaS). These three cloud types illustrate the usefulness of the service oriented computing (SOC) [1], allowing users to establish links among services, and thus organizing them as workflows instead of single standalone applications. A workflow is a standard to model business and scientific application data and control flows, as well as services that can be deployed to serve as part of large e-Science environments [2].

In the IaaS cloud service model, the client usually accesses cloud resources through an interface to perform user identification, to check the available resources, and to create/start/stop one or more virtual machine instances (VMIs). The user is charged in a pay-per-use basis, i.e., while the VMI is active. During this period, it is possible to start applications by verifying if all the necessary dependencies (other applications, services, libraries, etc.) are available and ready to use. If this is not the case, it is the user's duty to deploy them, installing and instantiating applications and publishing the services, so that the application, or the whole workflow, can be started and run smoothly. Despite the advances in resource abstraction and management in the cloud paradigm, the user yet has an expressive labour to be performed on the management of his/her application's execution.

One important point in the application management is monitoring. Current monitoring systems in cloud computing infrastructures have the objective to follow resources consumption, being oriented to provide information on resource usage metrics. While these metrics can be useful to the provider, they have limited usefulness to the cloud user. For example, in the Amazon EC2 [3], the available metrics are directed to the monitoring the instances: CPUUtilization, DiskRead/WriteOps, DiskRead/WriteBytes, and NetworkIn/Out are metrics with more meaning in managing resources than the applications running on them. With such metrics, the user may decide to add more resources to its cloud if a VM overload is detected, but the semantics of the obtained data from these metrics cannot be directly translated into impact in the applications running on the monitored resources. As a consequence, applications could be migrated or more resources could be included in the cloud with no need, or, on the contrary, the application may be yet unresponsive and with its quality of service compromised, and this may not be identifiable through resource monitoring metrics. This can happen if the VM itself is not overloaded, but if a hardware component of the underlying physical machine is overloaded by other VMs.

In this paper we present an infrastructure for the execution and monitoring of service workflows in private clouds. The tools in our infrastructure management system allow cloud users to follow their workflows execution, measuring the performance of computation and data transfer activities performed by each workflow component in the cloud resources. Through these measurements, the workflow manager can detect two basic situations where action can be taken: (i) if resources are not achieving the expected performance; and (ii) the application does not need as much resources as allocated to it. In the first case, the management system can react during the execution of the application to avoid quality of service degradation; in the second case the application can be run on a less powerful resource, leaving the most powerful ones for other resource-demanding applications. Besides that, the metrics in our monitoring system allow the user to keep track of the organization internal costs[1] involved in the workflow

[1] Such as in credit systems, where organization departments receive credits when letting other departments to use their resources.

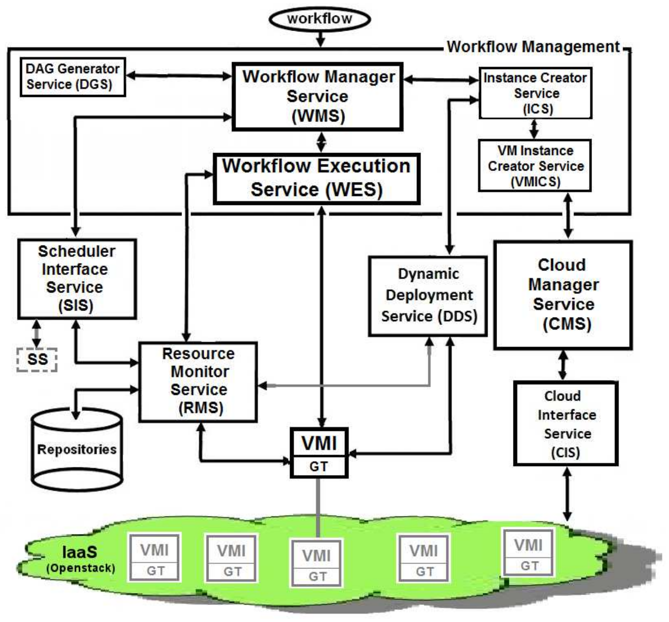
workflow
Workflow Management
DAG Generator Service (DGS)
Workflow Manager Service (WMS)
Instance Creator Service (ICS)
VM Instance Creator Service (VMICS)
Workflow Execution Service (WES)
Scheduler Interface Service (SIS)
Dynamic Deployment Service (DDS)
Cloud Manager Service (CMS)
SS
Resource Monitor Service (RMS)
Repositories
Cloud Interface Service (CIS)
VMI
GT
IaaS (Openstack)

Fig. 1. Management system infrastructure.

execution, which could be useful in deciding if it does worth adding resources from public providers to the cloud.

This paper describes in Section II our software infrastructure to support the workflow execution in private clouds. In Section III we detail the application monitoring system in the cloud, discussing its application as well as some application scenarios. Section IV presents a use case of the monitoring system. In Section V we describe related works, while Section VI presents the conclusion and future work.

II. Infrastructure for the Management of Workflows in Clouds

Our infrastructure for the execution of service workflows is composed of the following components: an IaaS middleware, a computational grid infrastructure, the workflow manager, and a set of support services.

The management of service composition in our infrastructure is performed by a set of services (Figure 1), whose coordination is made by the Workflow Manager Service (WMS). The WMS allows the creation and management of application flows, tasks, and services from computational grids. The WMS architecture is based on concepts presented by the OGSA (Open Grid Services Architecture) [4] and extended with characteristics of WS-BPEL (Web Service Business Process Execution Language) [5]. By using a computational grid we create a distributed environment to support the parallelism of activities, such as the concurrent execution of workflow services, the distributed monitoring, and the dynamic service publishing in the virtual machine instances. To accomplish this, we chosen the Globus Toolkit (GT) [6], which is an implementation of the OGSA. In the OGSA, all physical or logical resources are modeled as services in the grid, thus being in tune with an IaaS cloud platform.

The workflows are constructed by the user through the GPOL language [7], which uses concepts of service orchestration presented in WS-BPEL, but improved with specific directives for grid and cloud computing, such as state maintenance, potentially transient services, notification, data-oriented services and group-oriented services. GPOL includes variables, life-cycles, fabric/instance control, flow control, and fault treatment. Additionally, it allows the user to start the execution of tasks, services, and workflows sequentially or in parallel. Also, GPOL allows the user to create abstract workflows, leaving to the management system, along with the scheduler, the duty of deciding the best set of resources to run the application.

We consider that the received workflows are translated into Directed Acyclic Graphs (DAGs). The user builds abstract workflows, thus not identifying the physical resources where each service in the workflow must be run. After receiving the workflow, the WMS notifies the DAG Generator Service (DGS), so it can generate the resulting DAG with the description of the workflow activities along with the involved computational costs (processing costs, transmission costs, etc). The generated DAG will be sent to the scheduler, which decides the best resources to run each workflow component. The interface between the workflow management and the scheduler is done by the Scheduler Interface Service (SIS).The SIS makes the middleware scheduler-independent, and the user can choose the best scheduler for the current application being scheduled or cloud objectives.

The scheduler, through the SIS, returns the resource allocation for the workflow components to the WMS, which starts the transformation of the abstract workflow into a concrete workflow. In order to accomplish this, the Instance Creator Service (ICS) is invoked to create the necessary instances of the services needed in the workflow execution. However, before an instance can be created, another verification must be done: check if all the VMs indicated by the scheduler are already instantiated.

When all VMs in the concrete workflow already exist, the cloud infrastructure (OpenStack) needs to take no action, but the ICS still needs to verify if the necessary services are ready to be used in the virtual machine instance (VMI). If they are not, the ICS starts the procedure to publish the necessary services dynamically, which is done through the Dynamic Deployment Service (DDS). The DDS requests the service from the repository, transfers it to the VMI, publishes it, and updates the VMI's service container.

When one or more VM instances need to be started, the ICS calls the Virtual Machine Instance Creator (VMICS), which is responsible for the creation of the new virtual machine instance. The VMICS must obtain the most adequate VM image, start its creation, and return to the ICS the localization of the freshly deployed VMI so the services related to the workflow can be instantiated on it. As in the first case, if the image does not have all the necessary services, the ICS deploys them through the DDS.

To provide our private cloud, we opted for using the Open-

Stack open source platform [8]. OpenStack fulfills the necessary requirements to support the cloud through its computational infrastructure (*Nova*), its storage infrastructure (Swift), and its VM image management service (Glance). The relation between our software and the cloud (private, public or hybrid) is performed through the Cloud Manager Service (CMS), supported by the Cloud Interface Service (CIS). This design guarantees the independence of our infrastructure in relation to the particularities of the cloud infrastructure. For each cloud provider, we can have a corresponding CIS, restricting to the CMS all the logic behind our infrastructure. The CIS utilizes definitions of the Amazon EC2 API [3], what facilitates the integration of platforms such as OpenNebula, Eucalyptus, and other cloud solutions that make use of this API's specification.

With all the necessary services for the workflow execution already instantiated, the VMs change abstract services for real URLs, transforming the abstract workflow into the concrete workflow. The concrete workflow execution is performed by the Workflow Execution Service (WES).

III. Monitoring the workflow execution

The monitoring provided by our infrastructure acts in various ways. It monitors the VMIs resources (memory, CPU, etc), monitors the network, and closely monitors the execution of workflow activities. Besides the workflow execution coordination, the WES also monitors the execution with an activity by activity granularity, working along with the Resource Monitor Service (RMS). Together, WES and RMS register the time taken to run each activity (a service, a data transfer, a database access, etc) and verify if the performance is complying with the expected QoS.

All information about the cloud resources is obtained through the RMS. It operates in a distributed manner, maintaining one instance on each VMI, and supplying, on demand, information about the instance to the other infrastructure services. The RMS gives information about the amount of memory available, number of cores, CPU utilization, as well as total and available storage. It also offers information about the network where the resource is placed on, such as maximum and available bandwidth. Besides that, the RMS maintains detailed information about the services available on each VMI. Moreover, it provides information about each container URL, its protocol, access ports, and maintains in the repository average execution times of each operation previously executed by the VMI. This information is normalized to reflect the performance of that resource in relation to the others. The information on the RMS can also help in the composition of a usage map from the cloud, which could be useful to the scheduler. Then, the scheduler can, for instance, place tightly coupled services closer and connected by faster links.

The WES monitors the execution times of each section defined in the GPOL workflow, including times taken to execute other operations invoked in the process, registering them in a cumulative form into a separate log file for each workflow. To illustrate this monitoring, we show below a fragment of a GPOL workflow that applies a filter to an image file [7].

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="ip-A5000x5000" met="31500">
  <definitions name="job_Definitions">
(1) <variables>
      <variable name="retCode"  type="int" value="0"/>
    </variables>
(2) <gservices>
      <gsh name="vm1" uri="FactIPS" type="Factory"/> <!-- cm="01" -->
      <gsh name="vm2" uri="FactIPS" type="Factory"/> <!-- cm="02" -->
      <gsh name="vm3" uri="FactIPS" type="Factory"/> <!-- cm="02" -->
      <gsh name="vm4" uri="FactIPS" type="Factory"/> <!-- cm="03" -->
    </gservices>
  </definitions>

<process name="p-ip-A5000x5000">
(3) <invoke name="vm1" method="sliceMatrix" met="2500">
        <argument variable="sliceIn" type="string"/>
        <return variable="sliceOut" type="string"/> </invoke>

(4)<flow name="F-SCP-send" met="7500">
     <invoke name="vm1" method="runCommand" tagname="sendS1Tovm3">
        <argument variable="S1Tovm3" type="string"/>
        <return variable="retCode" type="int"/> </invoke>
     <invoke name="vm1" method="runCommand" tagname="sendS1Tovm4">
        <argument variable="S2Tovm4" type="string"/>
        <return variable="retCode" type="int"/> </invoke> </flow>

(5) <flow name="F_medianFilterMatrix" met="13000">
        <invoke name="vm3" method="medianFilterMatrix" tagname="vm3MF">
           <argument variable="mFIn-1" type="string"/>
           <return variable="retCode" type="int"/> </invoke>
        <invoke name="vm4" method="medianFilterMatrix" tagname="vm4MF">
           <argument variable="mFIn-2" type="string"/>
           <return variable="retCode" type="int"/> </invoke> </flow>

(6)<flow name="F-SCP-receive" met="7500">
     <invoke name="vm1" method="runCommand" tagname="rsf_vm3">
        <argument variable="R1Fromvm3" type="string"/>
        <return variable="retCode" type="int"/> </invoke>
      <invoke name="vm2" method="runCommand" tagname="rsf_vm4">
        <argument variable="R1Fromvm4" type="string"/>
        <return variable="retCode" type="int"/> </invoke> </flow>

(7) <invoke name="vm1" method="mergeSliceFiles" met="700">
        <argument variable="mergeIn" type="string"/>
        <return variable="mergeOut" type="int"/> </invoke>
<return variable="retCode" type="int"/>
</process></job>
```

In step (1), the workflow variables are created with information of the services to be used. In step (2), the service instances (IS) utilized in the workflow are created. It is important to note that there is no indication about the resources localization or about the virtual machines localization where the ISs must be created. Therefore, this specification is related to an abstract workflow. The computational resources will be indicated by the scheduler and properly utilized by the WMS, transforming the abstract workflow into a concrete one during the execution. The association between the service instance and the VM where it will be created is given by the tag "gsh-name". In our GPOL example, we chose to create four instances of the *FactIP*, which associations are with *vm1*, *vm2*, *vm3*, and *vm4*. In the rest of the workflow, the instance of the service to be used is always referenced through this name, no matter in what VMI it was created. In step (3), we used the IS *vm1* to split the image file in two parts. The resulting parts are then sent in parallel to the resource where they will be executed, i.e., the resources where *vm3* and *vm4* are running, so the filter can be applied independently (4). In step (5) the median filter is applied in parallel by the ISs *vm3* and *vm4*. In (6) the resulting filtered file parts are sent to the VMI where the IS *vm1* is running, which in turn merges the files into the resulting image. At each execution of this workflow, the WES adds to the log file the following information:

```
... GPOMaestro Vr. GPO Maestro Service 1.5.1 Jul 2012
      Processing ip-A5000x5000.gpol at Sat Sep 15 10:37:20 BRT 2012
.   Definitions Execution Time: 5
.   ExecProcess: invoke vm1 sliceMatrix  end - Time: 2378
.      flow.Invoke: vm1 runCommand  sendS1Tovm3  end - Time: 6246
.       flow.Invoke: vm2 runCommand  sendS2Tovm4  end - Time: 7050
.   ExecProcess: flow F-SCP-send end - Time: 7051
.      flow.Invoke: vm3 medianFilterMatrix  vm3MF  end - Time: 10202
.      flow.Invoke: vm4 medianFilterMatrix  vm4MF  end - Time: 11958
.   ExecProcess: flow F_medianFilterMatrix  end - Time: 11958
.      flow.Invoke: vm2 runCommand  rsf_vm4  end - Time: 6701
.      flow.Invoke: vm1 runCommand  rsf_vm3  end - Time: 6797
.   ExecProcess: flow F-SCP-receive   end - Time: 6799
.   ExecProcess: invoke vm1 mergeSliceFiles  end - Time: 617
.   Process Execution Time: 28806
--- GPO end Processing ip-A5000x5000.gpol Time: 32162
```

In this execution, 2378ms were taken to split the file into two parts (sliceMatrix), 6246ms to send the first slice to *vm3* (sendS1ToVM3), and 7050ms to send the second slice do *vm4* (sendS2ToVM4). The filter application took 10202ms in *vm3* (vm3MF), while it took 11958ms in *vm4* (vm4MF). To bring the filtered slices back to *vm1*, 6797ms and 6701ms were taken for rsf_vm3 and rsf_vm4, respectively. The merge of the two filtered pieces took 617ms in *vm1* (mergeSliceFiles). The total Process Execution Time was 28806ms , considering that many operations were performed in parallel. This small example shows the relevance of the performance information. It is possible to note in the data transmission of the slices that the communication channel between *vm1* and *vm3* is faster than between *vm1* and *vm4*. Even though *vm3* and *vm4* are two VMs of the same class (cm="02") and run the same service FactIP, *vm3* was faster in the execution of the filter.

Besides generating this log for each workflow, the WES maintains this information in repositories associated with the RMS in each cloud resource. Since we are using computational resources made available in the form of virtual machines, it is important to normalize the collected information. To accomplish this, our workflow management system gathers information about the machine class where the VMI was instantiated. By combining the execution times and the class of the host machines, the infrastructure is able to provide relevant information to the scheduler in its choice for the best resources. However, the cloud administration should decide how the machines are classified, and should populate the middleware repositories with the available hardware classes. One option to make such classification is to use the model provided by the public clouds, such as Amazon [9], which adopts a combination of characteristics (hardware, usage type, resource options, etc) to charge for the use of its resources. In our example, we illustrate this by putting this information as a comment where services of the workflow are defined (vm1/vm2 cm="01" and vm3/vm4 cm="02"). This is only an example, since this classification is obtained during the execution of the workflow directly from the cloud infrastructure. When the execution is finished, all information is sent to the global repository, where they are maintained.

Our infrastructure provides to the schedulers not only information about the resources, such as processing capacity, storage and communication, but also information about the resources performance on each operation performed in the VMIs. The association of the VMI with the hardware where it is running allows to infer the performance of the cloud resources without identifying its physical resources.

The RMS also monitors the time involved in other workflow activities. The time to create each VMI, the time to dynamically deploy each service not available at the time of the VMI deployment, times taken to initiate service containers, times taken to create each service instance, and times taken to distribute service instances and VM images are examples of information collected by the infrastructure. This information is made available to the scheduler, supporting it in the choice for the best resource to be used in the workflow execution. Below we discuss how information collected through our monitoring, harvested from the perspective of the user, is used by the infrastructure to make decisions and act in an attempt to guarantee the desired QoS.

A. *Application scenarios*

All workflow activities are monitored during the execution. By detailing the execution steps, it is possible to identify some important points from the monitoring in the user's perspective. In the *definitions* section of the workflow, the system monitors the dynamic creation of service instances that will be used. The most simple case occurs when the private cloud already has the VM instances ready to use. In this case, only the time taken to create the service instance is computed.

The monitoring system allows the user to choose the best allocation strategies for the VMIs. For workflows that are often executed, it would be interesting to create VM images with all services already published. Moreover, the creation of all necessary instances beforehand may be unfeasible, since this could consume time and hardware resources, impairing the execution of other applications in the VMs already deployed in the same physical hardware. Also, by using our infrastructure monitoring system, the cloud administrator can observe the cloud usage not only in terms of hardware load, but also profiling the applications being executed and decide the best deployment strategy for each type of VM available, considering costs and energy/green aspects of each strategy.

Note that in the private cloud, the types of VMI available for deployment are restricted by the available hardware. For instance, if there is only one 16 core machine in the pool, and it already has a VMI with 4 cores deployed on it, no VMI with 16 cores could be deployed for an application with higher demand for CPU parallelism. However, if the 4 cores VMI was deployed on a 4 cores physical machine, this 16 cores could still be used in the cloud by deploying a 16 core VMI on it. Therefore, the dynamic deployment of VMs and service instances brings benefits to the cloud as a whole, potentially increasing resource utilization. Thus, for workflows executed for the first time or workflows with low execution frequency, it may be interesting to deploy VMIs and services only on demand, leaving more resource options to other workflows.

B. *Rescheduling: correcting QoS*

Our infrastructure allows the user to associate Service Level Agreements (SLAs) with the workflow execution. This can be done by declaring requirements of each step of the workflow in the repository or directly in the workflow specification. The GPOL has the tag *met* (Maximum Execution Time), which can be optionally specified in the time-consuming activities of the workflow. In our example, we indicate *met="31500"* for the workflow execution as a whole, *met="2500"* to the *sliceMatrix* operation, *met="13000"* for the flow *medianFilterMatrix*, *met="7500"* for the flow *F-SCP-receive*, and *met="700"* for the *mergeSliceFiles* operation.

If the *met* is defined, the WES utilizes the measured times given in a cumulative form to monitor the whole workflow execution. If this global measurement shows that the time indicated will not be obeyed, the WES interrupts the execution and requests to the WMS the generation of a new scheduling for the activities that were not yet run in the workflow. The new scheduling should be performed in a way to try to obey the QoS indicated by the user. When the WMS receives this rescheduling request, it uses the DGS to obtain the partial DAG, and then it invokes the scheduler through the SIS. Finally, the same steps are taken as in the original workflow execution in order to deploy VMs and service instances through the cloud services (ICS, VMCIS, etc), returning to the WES the new concrete workflow to be run.

In our example, an elapsed time above 2500ms in the *sliceMatrix* operation indicates that *vm1* is not corresponding to the requested QoS. When this deviation is perceived, the WES can generate a new DAG, but with the flow *F-SCP-send* as the first task to be accomplished. Then, the scheduler should indicate a faster resource in order to avoid the overall workflow *met* to be surpassed. All the necessary references to synchronize the workflow activities are automatically corrected by the workflow manager, making this rescheduling process totally transparent to the user.

C. *Monitoring to evaluate public clouds*

The presented monitoring service can be also used in public clouds. To do this, it suffices to create a VM image with the infrastructure (middleware, published services, workflows, and the necessary files), and then create the necessary VMs for the execution of the workflow. Then, the infrastructure can monitor the execution of each activity of the workflow, generating the execution log. After some executions, it would be possible to characterize the performance of the VMs in the public cloud using the local VMs as references. With that, decisions made by schedulers for hybrid (public + private) clouds could be better supported, evaluating with better precision the cost-benefits involved in outsourcing workflows (or part of them) to public clouds [10].

IV. Use case analysis

In order to illustrate the application monitoring provided by our infrastructure, we set up a small experiment with a median filter workflow. The median filter is a fork-join DAG with one call to a service that slices an image in two, two parallel calls to a service that applies a median filter to each slice, and one call to a service that merges the results into the final image.

TABLE I
VMI Classes characteristics.

Class	Cores	RAM	Network	CPU (hardware)
cm01	1	2 GB	100 Mbps	Intel Pentium 4 3.00GHz
cm02	1	4 GB	1 Gbps	Intel Core2 Quad 2.40GHz
cm03	1	16 GB	1 Gbps	Intel Xeon E5430 2.66GHz

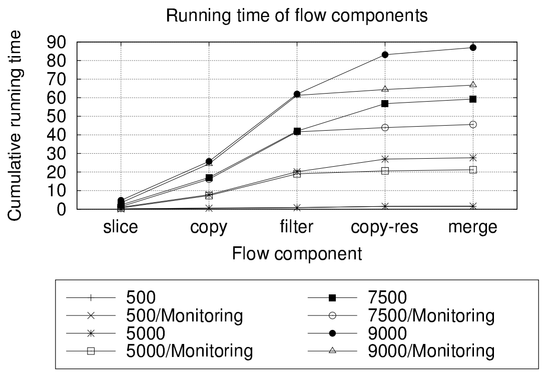

Fig. 2. Cumulative makespan for the median filter workflow.

When these calls are made to services on different resources, the needed image files are transferred through the network so the service can be run.We have run the experiments with four different image sizes: 500×500, 5000×5000, 7500×7500, and 9000×9000. We deployed a cloud with 3 VM classes, as shown in Table I.

We compared two executions of the workflow: (i) without the monitoring system, where the slice and merge operations were performed on a *cm01* resource, while the filters were performed in parallel on two *cm02* resources; and (ii) with the monitoring system, where the infrastructure is capable of, based on the *met* tag in the workflow, detecting that using a *cm01* resource would delay the execution, and then it decides to reschedule the merge operation to a *cm03* resource.

Figure 2 shows the cumulative makespan of the workflow execution after each operation performed. After the slice of the image file, the monitor system detects that it took too long to copy the slices from the *cm01* resource to *cm02* resources where the filter is applied. With this, when the monitoring was active, the rescheduling took place by moving the merge operation to a *cm03* resource, which has a faster network interface. We note that with the monitoring, the operation *copy-res* was faster than the first *copy* operation, accelerating the execution of the workflow. For the 500×500 image size, the slice size is too small and the network performance of the *copy* operation did not delay the workflow execution, and the rescheduling could be disposable.

V. Related Work

Cloud computing platforms such as Amazon [9], OpenStack [8], OpenNebula [11], Eucalyptus [12], and Nimbus [13], offer functionalities for monitoring. In general, they offer a portal through which the users can be aware of the cloud resources behavior in real time. Some portals can show performance graphs with statistics about the vital resources, allowing the creation of alarms for given conditions. Also, reports can be personalized using tools provided by some platforms. The Amazon Web Service (AWS) platform, through the Amazon CloudWatch [14] provides an interesting monitoring system that allows the user to follow the performance not only of the VM instances, but also of the database systems. The Amazon CloudWatch has VM monitoring metrics for the CPU usage, latency, memory usage, I/O frequency, among others. However, its monitoring system, as the monitoring from the other cloud platforms, is resource-oriented, not providing relevant information for the applications execution, whereas our proposal not only monitors the resources of the cloud but also the performance of services allowing the user to monitor each activity of the workflow.

In spite of the rapid evolution of the cloud computing, there are a few works about monitoring. In [15], the authors discuss the project and implementation of a monitoring system in private clouds named PCMONS. Their proposal allows the construction of a cloud with open source solutions, and it offers functionalities for the integration with traditional tools such as Nagios [16]. In [17], the authors propose a holistic monitoring system based on agents. The system functionalities are mainly focused on network node monitoring. The network monitoring is only part of what our infrastructure can provide.

Other alternative is the use of traditional resource monitoring tools such as IBM Tivoli [18], Nagios [16], and Ganglia [19], adapted to cloud environments. However, in our scenario, such tools would need to be adapted in order to cope with applications monitoring, virtualization and its relation with the underlying hardware (i.e., association with VMs and hardware classes, and also relation between VMs to detect underlying hardware overload), as well as to deal with the execution time of distributed tasks in parallel with their dependencies.

None of the above solutions offers detailed monitoring of applications that are a composition of tightly-coupled services in clouds.

VI. Conclusion

Cloud computing environments provide resource monitoring, but they usually do not offer tools that support the user in the application monitoring. When considering workflows composed of services, monitoring options from the user point of view are not available to the best of our knowledge. In this paper we present an infrastructure for the execution of service workflows in private clouds. The presented infrastructure offers tools for the user to keep track of the execution of the activities completed during the execution of services that compose the workflow. By having the performance of the applications monitored, the infrastructure can take actions in order to improve the workflow execution.

The workflow management in our infrastructure makes the access to the cloud resources transparent to the user, providing uniform access by giving automatic support for the creation of VMs, as well as managing their life-cycle. Through our monitoring system, the user can observe the performance of each activity in the workflow, also enabling the gathering of information about the normalized resources performance. Moreover, the infrastructure is able to perform rescheduling of parts of the workflow according to the execution performance being observed. This can avoid application delays and low QoS. Our monitoring system can also be utilized to assess resources leased from public clouds, allowing the user to make a more concrete evaluation of the cost-benefit offered by each public cloud provider.

As future work, we are expanding the infrastructure functionalities in order to provide an easy way of measuring and comparing resources from public clouds with the ones in the private cloud. The development of rescheduling algorithms for clouds are also of interest.

Acknowledgment

We would like to thank CNPq and RNP for the financial support.

References

[1] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The next step in web services," *Communications of ACM*, vol. 46, no. 10, pp. 29–34, 2003.

[2] I. Taylor, E. Deelman, D. Gannon, and M. E. Shields, *Workflows for e-Science: Scientific Workflows for Grids.* Springer, 2007.

[3] Amazon Co., "Amazon Elastic Compute Cloud (ec2) API," http://aws.amazon.com/ec2/, 2012.

[4] "The physiology of the grid: An open grid services architecture for distributed systems integration," 2002. [Online]. Available: http://www.globus.org/alliance/publications/papers/ogsa.pdf

[5] "Web services business process execution language version 2.0," OASIS Web Services Business Process Execution Language (WSBPEL) TC, Tech. Rep., April 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[6] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, Beijing, China, 2005, pp. 2–13.

[7] C. R. Senna, L. F. Bittencourt, and E. R. M. Madeira, "Execution of service workflows in grid environments," *International Journal of Communication Networks and Distributed Systems (IJCNDS)*, vol. 5, no. 1/2, pp. 88–108, 2010.

[8] Openstack Org., "Openstack cloud software," http://http://www.openstack.org, 2012.

[9] A. Co., "Amazon Web Services," http://aws.amazon.com/, 2012.

[10] 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.

[11] Openstack Project, "Opennebula," http://www.opennebula.org, 2012.

[12] Eucalyptus System Inc., "Eucalyptus," http://www.eucalyptus.com, 2012.

[13] Nimbus Project, "Nimbus platform," http://www.nimbusproject.org, 2012.

[14] Amazon Co., "Amazon Cloudwatch," http://aws.amazon.com/cloudwatch/, 2012.

[15] S. A. de Chaves, R. B. Uriarte, and C. B. Westphall, "Toward an architecture for monitoring private clouds." *IEEE Communications Magazine*, vol. 49, no. 12, pp. 130–137, 2011.

[16] Nagios Enterprises, "Nagios," http://www.nagios.org/, 2012.

[17] P. Hasselmeyer and N. d'Heureuse, "Towards holistic multi-tenant monitoring for virtual data centers," in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*. IEEE, Apr. 2010, pp. 350–356. [Online]. Available: http://dx.doi.org/10.1109/NOMSW.2010.5486528

[18] IBM, "IBM tivoli monitoring," http://www-01.ibm.com/software/tivoli/, 2012.

[19] "Ganglia Monitoring System," http://ganglia.sourceforge.net/, 2012.