Execution of service workflows in grid environments

C. R. Senna, L. F. Bittencourt and

E. R. M. Madeira*

Institute of Computing,

University of Campinas,

P.O. Box 6196, Campinas, São Paulo, Brazil

E-mail: {crsenna, bit, edmundo}@ic.unicamp.br

*Corresponding author

Abstract: Grid computing has emerged as a powerful environment for parallel processing and sharing computational resources to achieve high computational power. Grids based on the OGSA use the SOA concepts to provide mechanisms to share resources through interfaces which can be remotely accessed to invoke services running on grid resources. This new grid service paradigm brings new challenges for users and grid developers. In this paper we present the Grid Service Testbed (GST), an infrastructure for performance evaluation of grid service compositions in grid environments. The GST receives job submissions through a workflow manager using a workflow composition language, which allows services to be invoked sequentially or in parallel. Experimental results using a typical application, an image processing filter, show that the overhead when using a service composition scheme does not prevent the grid from giving fast workflow execution. Additionally, we demonstrate how GST manages the dynamicity of grid environments using its particular functionalities, namely the dynamic service instantiation and the automatic reference coordination, to choose the best computational resources without user interference. These functionalities offer suitable treatment for the dynamicity of the grid, specially for opportunistic grids. These GST features help users to choose the best resources and to build the compositions in a way to improve the performance, besides showing how the developers may use the GST information to support scheduling decisions.

Keywords: Grid computing, scheduling, service composition, service workflows.

Reference: to this paper should be made as follows: Senna, C.R., Bittencourt, L.F. and Madeira, E.R.M. (2009) "Execution of service workflows in grid environments", Int. J. Communication Networks and Distributed Systems, Vol. x, No. x, pp. xx-xx.

Biographical notes: C. R. Senna received his BS degree in Computer Science from the Federal University of São Carlos, Brazil, in 1979 and his MS degree in Computer Science from the University of Campinas, in 2007. He is currently a Ph.D. candidate at the University of Campinas, and his major interests include Distributed Systems and Grid Computing.

L. F. Bittencourt received his B.S. degree in Computer Science from the Federal University of Paraná, Brazil, in 2004 and his M.S. degree in Computer

Author

Science from the University of Campinas, Brazil, in 2006. He is currently a Ph.D. candidate at the University of Campinas, and his major interests are in the field of scheduling in heterogeneous and grid computing environments.

E. R. M. Madeira is an Associate Professor at the Institute of Computing of State University of Campinas—UNICAMP, Brazil. He received his Ph.D. in Electrical Engineering from State University of Campinas—UNICAMP, Brazil, in 1991. He has published over 120 papers in national and international conferences and journals. He has also supervised more than 40 master and Ph.D. students. He was the General Chair of the MATA'04 (IEEE/IFIP International Workshop on Mobility Aware Technologies and Applications), Technical Program Committee Chair of LANOMS'03 (IEEE Latin American Network Operation and Management Symposium), Technical Program Committee Co-chair of IPOM'05 (IEEE International Workshop on IP Operations and Management) and LANOMS'07, and Workshop Co-chair of NOMS'08. He is a member of the Editorial Board of Journal of Network and Systems Management. His research interests include Grid Computing, Network Management and Distributed Systems.

1 Introduction

The Internet open standards allow organizations to share their computational resources since the initial product development phase until the final steps of the business process. The union of these technologies resulted in the emergence of dynamic and multi-institutional Virtual Organizations, being the computational grid (Schulze et al., 2007), (Foster et al., 2002) an adequate tool to manage the resource sharing among these organizations. In a virtual organization, the service concept becomes not only the basis to applications, but also a fundamental part for the collaboration among the participants. This naturally collaborative environment allows the users to establish bindings between services, organizing them as workflows instead of isolated standalone applications. Such service compositions bring different difficulties as a result from the interaction among the virtual organization participants, concerning the coordination and dynamic composition of processes and services.

In this paper we present the Grid Service Testbed (GST), an infrastructure for performance evaluation of grid service compositions in grid environments, and show how the migration to the grid service paradigm impacts on the execution. Additionally, we discuss how the migration can be done, which are the performance benefits to the user, how the GST helps users to choose the best resources and parallelism for their compositions, and how the GST's information supports scheduling decisions. For this purpose, we use a typical application, an image processing filter, represented by a workflow and executed as a composition of grid services. We compare the traditional standalone execution scheme with the parallel workflow execution using grid services.

The paper is organized as follows. In Section II we present the testbed, giving details about the infrastructure, implementation languages and development issues. Section III describes the image processing application. The experimental results are shown in Section IV, while the Section V shows how the GST can be used by a user to improve her service composition. Section VI describes a selection of related works. Section VII concludes the paper and gives outlines for future works.

2 Infrastructure

The GST deployed to run the experiments is a computational grid composed of a grid toolkit, a workflow manager and a scheduler. In this section we describe these components.

2.1 The computational grid

Computational grids are shared, heterogeneous and dynamic environments. They allow resource sharing between organizations and individuals who want to make use of a distributed infrastructure to achieve high computational power. Grids are used for many purposes, including sharing storage, processors, software, data, and specialized instruments, with providers and consumers defining rules for such sharing. The set of people, institutions, and distributed resources interconnected by a network through multiple domains, sharing resources and objectives, constitutes the Virtual Organizations (Foster et al., 2002).

In our testbed we used the Globus Toolkit (GT) (Foster, 2005) as the grid infrastructure. It is an implementation of the OGSA (Open Grid Services Architecture) (Foster et al., 2002) proposed by the OGF (Open Grid Forum) (OGF, 2009). In the OGSA all logical and physical resources are modelled as services in the computational grid, bringing to the environment concepts like SOA (Service Oriented Architecture) (Curbera et al., 2003). We used the GT version 4 installed on a Debian GNU/Linux operating system. All resources are connected by a gigabit ethernet network.

2.2 The workflow manager

In a virtual organization, services become a primordial concept to provide collaboration between participants. In such an ambient, the traditional workflow standards limit the interaction among users of the virtual organizations, mainly in what concerns to the coordination and dynamic composition of services needed by these organizations. To provide a flexible and easy way to the development of services coordination and composition, a workflow mechanism controlled by the user is important. To manage the dynamicity of grid environments the software infrastructure provided by GST joins dynamic instantiation of services to on-demand provisioning in workflow executions, reaching better scalability, flexibility, and availability of resources as well as it improves the environment robustness. The GST's software offers functionalities for:

Dynamic service instantiation: the user doesn't identify the computational resource for service execution. The infrastructure chooses the best machine available at execution time;

Automatic reference coordination: the GST resolves the references between services after finding the best resource to process services without user interference.

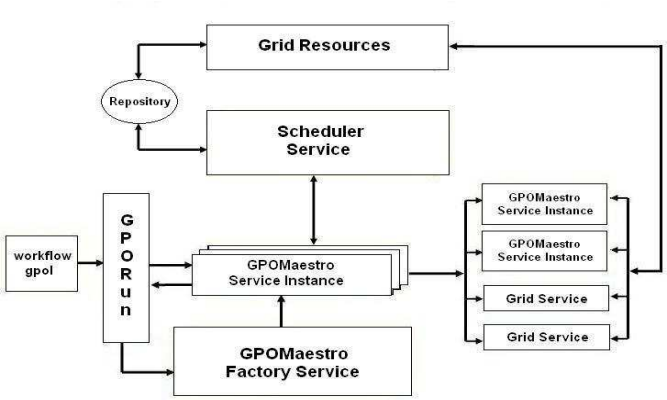
To manage the service composition in our testbed we used the Grid Process Orchestration (GPO), a scheduler, and a resource repository. The GPO (Senna and Madeira, 2007) is a middleware for interoperability of distributed applications that requires service composition on a computational grid. The GPO allows the creation and management of application flows, jobs and services on the grid. The GPO architecture is based on the concepts presented by the OGSA, and extends the GJS (WS-GRAM Job Description Language) (Foster, 2005) with WS-BPEL (Web Service Business Process

Execution Language) (OASIS WSBPEL, 2007) characteristics, specified by OASIS (Organization for the Advancement of Structured Information Standards).

2.2.1 The GPO Architecture

The Grid Process Orchestration architecture (Senna and Madeira, 2007) consists of three main components: the GPO Run, the GPO Maestro Factory Service and files specified in the GPO Language (GPOL). These components are shown in Figure 1. The GPO Run is the utility which receives the workflow submitted by the user and forwards the requisition to be processed. The GPO Maestro was built using the concepts of factory/instance provided by the grid infrastructures. The GPO Run uses the GPO Maestro Factory service to create new instances which are responsible for the execution of the submitted workflow. The scheduler service is responsible for scheduling the workflow based on information given by the resources and services repository, which has information collected by grid resource monitors.

Figure 1 The infrastructure architecture



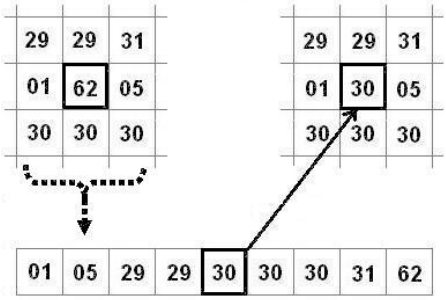
2.2.2 The Grid Process Orchestration Language (GPOL)

To build the workflows for our experiments we used the GPOL. The GPOL is an XML-based language that combines the easiness of GJD and the concepts of orchestration proposed by the WS-BPEL. Specific directives needed in the grid environment are present in GPOL, like state maintenance, potentially transient services, notification, and data and group oriented services. The GPOL includes workflow variables, lifecycle service instance control, factory/instance control, flow control, and fault control. Additionally, it permits to the user to start execution of services, jobs and workflows sequentially or in parallel. Grids are often used as a high performance platform to deliver high throughput and faster execution time for heavy load applications. Using the GPO/GPOL pair, we constructed workflows which use parallel processing to apply smoothness filters to images by segmenting and distributing pieces of the image to be independently processed. Using the GPO we also measured the time to instantiate services and invoke services as well as the time spent to apply the filter.

2.2.3 The scheduling service

In order to distribute the jobs composing the workflow, a scheduler is necessary. The scheduling service (Figure 1) is an architecture module which permits different scheduling algorithms depending on the application and use of the infrastructure. In our experiments we included executions using a simple algorithm which distributes services according to the number of cores on each resource, aiming at the minimization of the total execution time (makespan) for the median filter. Therefore, in this paper we focus on showing that the infrastructure using services is efficient even with a simple scheduling algorithm. The scheduler uses information from the repository, including resource characteristics, processor load, and the available services in the resources. More complex scheduling algorithms (Bittencourt and Madeira, 2008), which could explore more information from the resources monitor, or algorithms for a specific problem can be used without modifications in the infrastructure. In this sense, the development of a scheduler to be used in the context of service-oriented grids is an important problem. Such a scheduler would have to consider the service deployment times, for instance. Nevertheless, since this paper aims at showing the infrastructure and its usefulness, this is out of the scope here and is left as a future work.

Figure 2 The median filter operation



3 The application

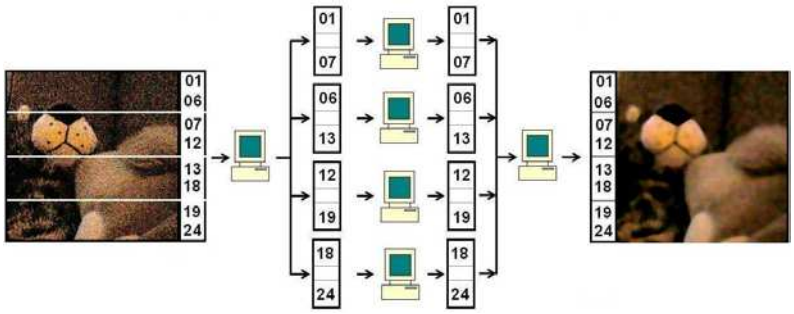
Image processing applications are commonly executed in computational grids. In general, an image is represented by a matrix of numbers, where each number has its correspondent colour to be shown on the screen, thus the higher the size of the image, the higher the matrix size. When dealing with medical images, the processing speed is critical, since the response time is a fundamental requirement. Additionally, this kind of application requires high resolution images to be as clear as possible. Furthermore, it is common to apply smoothness, attenuation or enhancement filters to obtain clearer images. Applying these filters to high resolution images can be expensive and may demand some computational power. Nevertheless, it can be done parallelizing the filter application by segmenting the image into smaller blocks. Then, the filter is applied in a distributed way, and the resulting blocks are merged into the final resulting image. This segmentation allows the parallelization of the filter using the grid resources. In our testbed we implemented the median filter (Lindley, 2008). This filter substitutes the value of a pixel by the median value from its neighbourhood. It sorts the pixels in the

neighbourhood and picks the median pixel value as the new value for the pixel of interest (Branco et al., 2000). Thus, for a neighbourhood of size 1 (3x3 filter), 9 elements of the matrix are sorted and the median value is selected, as shown in Figure 2. For a neighbourhood of size 2 (5x5 filter), 25 elements are sorted, and so on. In our experiments we show results for 3x3, 5x5, and 7x7 filters.

3.1 A. The parallelism model

To apply the filter in a distributed way, we first split the image into smaller blocks, then applied the median filter to each block separately, and merged the blocks to generate the final image. Because the filter uses a median value of a neighbourhood, there is data redundancy in the generated blocks. For instance, in the 3x3 filter the first and last lines on each block is duplicated, thus the first and last lines of the block can have the filter applied using their full neighbourhood. In the 5x5 filter the redundancy is of two rows in the beginning and two rows in the end of each block, and in the 7x7 filter the redundancy is of three lines. The Figure 3 illustrates the segmentation and merge operations.

Figure 3 The segmentation and merge operations



3.2 The filter execution

To analyze the migration process from a traditional application program to a service composition, we developed two programs and some workflows. The main idea is to do the migration process in three steps. In the first step, we built a program to apply the median filter to image files. In the second step, we created a second program dividing the first one into three independent modules in order to split the matrix file, apply the median filter, and merge the resulting files. We also developed a service which is used to execute system commands. Finally, we created a workflow that used modules of the second program encapsulated as a grid service with three operations. This hybrid option allows the evaluation of the functionalities of the service composition. In the third step we built a set of workflows to analyze several ways of parallelism for this application.

To execute the workflows which apply the filter to the matrix file, we developed a service for the grid. This service provides operations to create matrices to simulate image files (mCreateMatrix operation), to split the file into blocks according to the desired number of blocks (sliceMatrixToFiles operation), to apply the median filter to a file, generating another resulting file (MFilterMatrix operation), to merge the block files, generating the resulting image (mergeSliceFiles operation), and to run a system command (runCommand operation).

A set of workflows was generated in GPOL to evaluate the several ways of applying the median filter using a set of computational resources. For each image size, we built: a workflow to apply the filter to one single file with the entire matrix using only one resource; a workflow to split the file into smaller blocks, apply the filter to each block in parallel using only one resource, and merge the blocks generating the final image; and a workflow to split the file into smaller blocks, apply the filter to each block in parallel, using more than one resource, and merge the blocks generating the final image.

For instance, the example of workflow shown in Figure 4 was used to apply the 3x3 median filter to the file */tmp/A-2000x2000*, a file which represents an image with resolution of 2,000x2,000. In this workflow, the file is split into five blocks, which are processed by five instances of the service executed in parallel, and then they are merged into the resulting file. Since different schedulers can be used in GPO, the GPOL offers the option to left open the definitions of hosts where the instances will be created and where the execution will take place. In our example, the scheduler would receive the workflow and it would output the hosts, then the GPO could replace the *hosts* in the *uri* field according to the schedule. If there is no scheduler available, the user can just fill out the *uri* field with the host in which she wants to create each instance.

Figure 4 Example of a workflow file in GPOL

```
<?xml version="1.0" encoding="UTF-8"?>
<gpo:job name="ip-A2000x2000-5p-1rcz">
  <gpo:definitions name="job_Definitions">
(1) <gpo:variables>
      <gpo:variable name="retCode" type="int" value="0"/>
      <gpo:variable name="sliceIn" type="string" value="5 3 /tmp/A-2000x2000"/>
      <gpo:variable name="sliceOut" type="string"/>
      <gpo:variable name="mFIn-1" type="string"
                                    value="3 /tmp/A-2000x2000_s1 /tmp/A-2000x2000_smf1"/>
      ...
      <gpo:variable name="mFIn-5" type="string"
                                   value="3 /tmp/A-2000x2000_s5 /tmp/A-2000x2000_smf5"/>
     <gpo:variable name="mergeIn" type="string"
                                    value="5 3 /tmp/A-2000x2000 /tmp/A-2000x2000-MF3-5p"/>
     <gpo:variable name="mergeOut" type="string"/>
   </gpo:variables>
   <gpo:gservices>
(2)    <!-- instance 1 -->
       <gpo:gsh name="IA-mf1" uri="host:8080/factip/FactIPService" type="Factory"/>
        ...
       <!-- instance 5 -->
       <gpo:gsh name="IA-mf5" uri="host:8080/factip/FactIPService" type="Factory"/>
   </gpo:gservices>
 </gpo:definitions>

<gpo:process name="job_Process">
  <!-- Divide file in 5 parts -->
(3) <gpo:invoke name="IA-mf1" method="sliceMatrixToFiles">
        <gpo:argument variable="sliceIn" type="string"/> <gpo:return variable="sliceOut" type="string"/>
   </gpo:invoke>

(4) <gpo:flow>
      <!-- Aply median filter in parallel way -->
      <gpo:invoke name="IA-mf1" method="MFilterMatrix">
```

Author

```
            <gpo:argument variable="mFIn-1" type="string"/> <gpo:return variable="retCode" type="int"/>
        </gpo:invoke>
        <gpo:invoke name="IA-mf2" method="<MFilterMatrix">
            <gpo:argument variable="mFIn-2" type="string"/> <gpo:return variable="retCode" type="int"/>
        </gpo:invoke>

        <gpo:invoke name="IA-mf3" method="MFilterMatrix">
            <gpo:argument variable="mFIn-3" type="string"/> <gpo:return variable="retCode" type="int"/>
        </gpo:invoke>
        <gpo:invoke name="IA-mf4" method="MFilterMatrix">
            <gpo:argument variable="mFIn-4" type="string"/> <gpo:return variable="retCode" type="int"/>
        </gpo:invoke>
        <gpo:invoke name="IA-mf5" method="MFilterMatrix">
            <gpo:argument variable="mFIn-5" type="string"/> <gpo:return variable="retCode" type="int"/>
        </gpo:invoke>
    </gpo:flow>

(5)  <!-- Merge block files into a new file -->
     <gpo:invoke name="IA-mf1" method="mergeSliceFiles">
          <gpo:argument variable="mergeIn" type="string"/> <gpo:return variable="mergeOut" type="int"/>
      </gpo:invoke>

      <!-- Return value to GPO_Client -->
      <gpo:return variable="retCode" type="int"/>
     </gpo:process>
  </gpo:job>
```

In Figure 4, in (1) the workflow variables are created. These variables contain information that are passed to the services during the execution. In (2) the instances of the processing service are created. There are five instances, one for each matrix block. In (3) the operation to split the file into five blocks is invoked, and in (4) the five instances previously created are used to apply the filter to each block. Finally, in (5) the already finished blocks are merged and the final file is generated.

3.3 Implementation aspects

We used the Java Sun version 1.5 (Java SE, 2009). To implement the services, we used the Java Sun and development facilities provided by the Globus Toolkit 4, without requiring any additional API or library. The created services use the factory/instance standard on the services and on the resources, in line with the WS-Resource factory specification (OASIS WSRF, 2006), being able to manage multiple resources.

4 Performance Results

In this section we evaluate two important aspects of the service-oriented grid environment. The first one is the experimental results aiming at the evaluation of the overhead when using grid services on a testbed running the GPO over a GT4 installation. We evaluated the performance of a computational grid using the GPO as the service orchestrator, the GPOL to describe the workflows, and with the GT4 providing the underlying grid infrastructure. The evaluation was done by comparing the local execution of the median filter over a matrix with the distributed parallel execution of the same filter.

Whereas the local execution used a local service to apply the filter, the distributed execution used local and remote services to send, receive, split and merge the matrix, and apply the filter. The second aspect is a brief case study about how to migrate an application from the traditional batch execution to work in a service-oriented environment.

4.1 Performance evaluation

The filter was applied to matrices with sizes from 50x50 to 5.000x5.000, which would represent high resolution images with up to 24 Mpixels. We used the median filter with 3x3, 5x5, and 7x7 masks. For each matrix size and for each filter size we made the following comparisons:

- Service overhead: comparison between the execution time of the application with and without the use of services.
- Parallel execution in single resource: comparison between the execution using a single file and using multiple slices in parallel using only one resource.
- Parallel execution on multiple resources: comparison between the execution of the image processing workflow on one resource and on multiple resources.
- Migration to services: the expected gains when moving from a traditional application to a service-oriented workflow execution.

4.2 Testbed setup

The grid environment was composed of three shared resources, namely *zeus, cronos, and dionisio*, interconnected by an 1Gbps ethernet network. The network is part of the Computer Network Laboratory at the Institute of Computing, thus it is also shared and subject to load variation. The characteristics of these resources are shown in Table 1.

Table 1 Resources used in the testbed

Name	*Processor*	*Clock*	*Memory*
apolo	Intel Pentium IV 3.2E HT	3.2 GHz	2.5 GB
zeus	Intel Pentium IV 3.2E HT	3.2 GHz	2.5 GB
dionísio	Intel Core 2 Quad	2.4 GHz	4.0 GB
cronos	Intel Core 2 Quad	2.4 GHz	4.0 GB

To collect the information about the executions two logs were produced. One in the client resource console which shows times for instance creation, workflow execution time, and total execution time. The second log in the host where the GPO was being executed shows times for each instance creation, times for each executed service operation, times for GPOL operations, such as the <*flow*> operation, and the total time for the workflow execution. Note that the services could be on any resource, while the GPO Maestro was deployed on a single resource and it was responsible for coordinating the execution of each service call.

We experimented some variations while executing the workflow, combining executions with one single file, multiple slice files, single resources, and multiple resources. The keys in the graphics follow the pattern *SizeOfFilterMask-#OfSlices-*

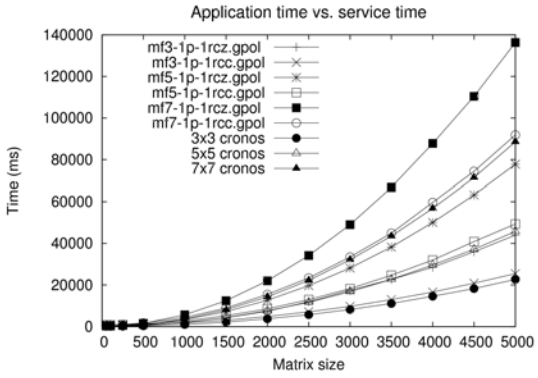
#OfResoures. For example, the key *mf3-5p-3rcc* means that the executed filter used a 3x3 mask, the matrix was split in 5 slices (or blocks) and these blocks were executed using 3 resources. The GPO was deployed in *zeus*, thus all executions were started from *zeus* and all matrices initially were in this resource, except when declared otherwise. Additionally, the *rcc* in the end of the key means that *cronos* split and merged the matrix files, thus the whole matrix file was sent to *cronos*, which distributed the slices to the other resources and sent the result to *zeus* in the end of the execution. On the other hand, *rcz* means that the split and merge were executed in *zeus*.

In the following sections we gradually compare some ways of executing our example application. First we evaluate the overhead associated with the use of services. After this, we evaluate how the GPO performs when executing the filter in parallel using a single resource. Next we do this same evaluation but with multiple resources, and we show a reasoning of which are the overheads related to the remote execution. Finally, we present some alternatives to improve the remote parallel execution of the filter, showing how the grid heterogeneity can be used to improve the execution of the workflow. All results show averages over 10 executions.

4.3 Service overhead

In this experiment we aim at evaluating the overhead of using services instead of the traditional local batch applications. In Figure 5 we compare the execution of a non-service client executed locally with a service client, both running the filter over 1 single file and executing it in 1 resource. The aim of this comparison is to evaluate the impact of using services in the execution time. In this figure, the key *3x3 cronos* represents the client executing locally the median filter with 3x3 mask. The results show a small impact in the evaluated scenario, since we can note a difference of about 2,000ms between the curves *mf3-1p-1rcc* and *3x3 cronos* for a 5,000x5,000 matrix, for instance. This small difference is also noted when the filter used the 5x5 or the 7x7 masks. As expected, the executions with *rcz* in the key were worse because the filter was executed in *zeus*, a slower resource.

Figure 5 Local application versus service time



4.4 Parallel execution in a single resource

Once shown that the overhead of using services to execute the filter over a single file is almost null, now we aim at showing that a parallel execution using the GPO workflow manager can overcome the limitations of a sequential execution, consequently reducing the execution time. To do this, we now compare the execution of the median filter service over 1 file with the execution of the service over multiple slice files on a single resource. Such comparison is given by Figure 6. While in *zeus* we can note a worsening as the number of slices increases, in *cronos* the behaviour is the opposite: the higher the number of slices, the smaller the execution time. However, despite this pattern in a general view, there are some observations which deserve attention. First, we can note that *cronos* performed better when there were 8 slices than when there were 10 slices. Since *cronos* is a quad core resource, an 8 slice parallel execution can take advantage of executing smaller pieces with less context changes in memory, cache and processor registers. Second, the 5 slices execution performed as good as the 10 slices execution on *cronos*, despite its bigger file sizes. We also assign this behaviour to the constant context changes when using 10 parallel pieces.

Figure 6 Single file and multiple slices service execution time

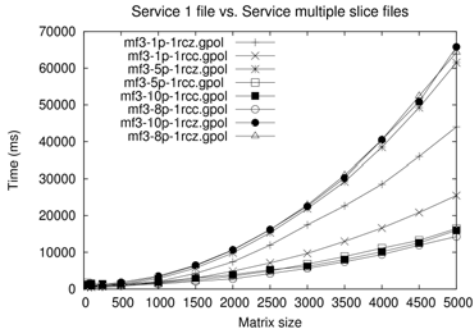
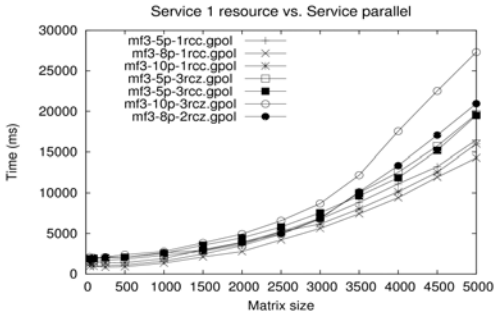


Figure 7 Single resource and parallel service execution time

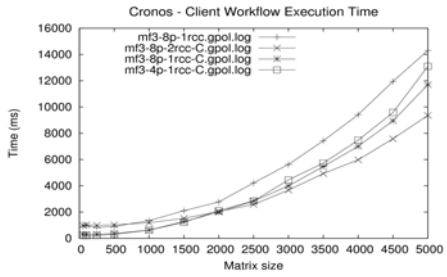


4.5 Parallel execution on multiple resources

In these experiments we compare the execution of the filter with multiple slices in a single resource with multi-slice executions in multiple resources. While executing everything in parallel on a single resource has the advantage of having no data transfer between resources, the use of multiple resources has the advantage of offering more computational power. We evaluate how this traditional scheduling issue impacts in our application scenario, and then we analyze what can be done to avoid potential delays in the execution. The first noticeable remark in Figure 7 is that the best execution times were given by parallel executions in *cronos*, while executions with 5 slices in 3 resources given intermediary results and executions with 8 and 10 slices given higher times.

To investigate why executions in more than one resource gave worse results than that on one resource, we evaluated the steps of these executions. Besides the data transfer times, we found that the merge and slice times were responsible for a significant part of the overhead in the distributed execution, since *zeus* is the least powerful resource and the slice/merge steps were performed there. Please refer to (Senna et al., 2009) for further details on the overheads involved in the distributed execution. The first direction to take in order to overcome the remote execution overheads is to use a more complex scheduling algorithm, which would take into account the data transfer times and merge/slice times before deciding where to execute each part of the workflow. Also, one could improve the merge/slice times by using optimized routines and using compressed matrix files to minimize the data transfer time. In the next experiment we evaluated another solution: deploy the workflow manager in a resource with more computational power. This way some parts of the matrix can be processed without being transferred through the network and the merge/slice times can be minimized. Figure 8 shows the GPO workflow execution time and the filter time when the GPO was deployed in the *cronos* resource (rcc-C keys). With the GPO Maestro in *cronos* we achieved better results when executing the filter in a distributed way. The mf3-8p-2rcc-C workflow improved by 30% the execution time of the best execution scheme achieved when using the workflow manager in *zeus* (mf3-8p-1rcc). It also improved the execution of all blocks in a single resource, showing that the distribution of many blocks over the more powerful resources can compensate the overhead of transmitting data and splitting the matrix file. We also executed all experiments for 5x5 and 7x7 masks, which showed similar results with variations in the absolute values, but maintaining the same patterns.

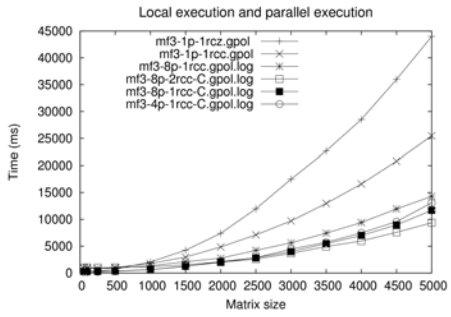
Figure 8 Execution times when GPO was deployed in *cronos* (rcc-C keys).



4.6 Migration to services: a brief case study

In our comparative analysis between traditional applications and composed services for grids we see another important aspect beyond the performance of the whole system: the migration process to an operational service grid environment. To evaluate the transition steps, we build an intermediary step between the application and the service workflows. As seen in Section 3.2, we briefly modified the application to allow the median filter application in steps. We created an option to split the original file in blocks, an option to apply the filter to a file, and an option to merge the resulting blocks. Using these new functionalities from the application, we built workflows which used the operation that executes commands from the system (*runCommand*), already implemented in our service, to execute the application as it was a service from the grid. This procedure allowed us to test if the strategy defined to the change of paradigm is viable. Encapsuling the application as a service allows the verification of the service platform without the necessity of developing the whole service. Notwithstanding, this makes the process more scalable allowing the definition of a migration schedule without stopping the working system. Once the service is ready to use, it can be deployed and it can substitute the old non-service application by invoking the new service inside the workflow files, substituting the *runCommand* Argument. Using this migration steps, the user would be able to execute her application in a remote host or in multiple remote hosts. Figure 9 presents the gain achieved when migrating from a local sequential execution scheme to parallel execution schemes in one or multiple resources. Considering *zeus* as a common desktop resource and 5,000x5,000 matrix files, we can initially note a significant performance gain, in the order of 45%, from the local sequential execution in *zeus* (*mf3-1p-1rcz*) to a remote sequential execution in *cronos* (*mf3-1p-1rcc*), a resource which would be part of a computational grid as a processing resource. There is another bold gain, in the order of 40%, when we compare the remote sequential execution in *cronos* with the remote parallel execution in *cronos* (*m3f-8p-1rcc*), which can be achieved using the GPO Maestro and the GPOL straightforwardly. The next possible execution scheme, the remote execution in multiple resources (*mf8p-1rcc-C*), leads to a improvement in the order of 30% when compared to the remote parallel execution in *cronos*. Thus, in our testbed with our sample application, the overall gain for a user at an ordinary desktop machine *zeus* who migrates to a parallel execution environment using the services facilities provided by the GPO can be as high as 78% for a 5,000x5,000 matrix size, from an execution time of around 45,000 ms to an execution time of around 10,000 ms.

Figure 9 Comparison between local execution and parallel execution



Author

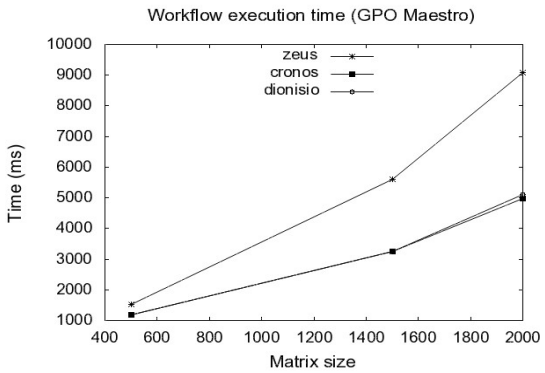
5 The testbed as a support tool for the user

In this section we show how the GST can be utilized by the user to refine the workflows. First we discuss the use of GST to assist the choice of the best computational resources for each workflow. Next we show how the GST can be used to evaluate the execution of a service composition in sequence and in different parallel ways, showing which one has better performance. Finally we show how to use the GST to evaluate use policies through service scheduling options in the grid.

5.1 Dynamic service instantiation as a tool to define the computational resources

In the workflows shown in the previous section we used precise references about which computational resources should be used in the execution of each service. Those concrete workflows could restrict the use of the grid, where dynamicity is an important characteristic, since they cannot use new resources that could be available when they are executed. To deal with the grid dynamicity, the GST allows the execution of flexible workflows where the user does not indicate the computational resource localization. At the moment of the execution, the GST dynamically instantiates the necessary services, choosing the best available computational resources, and automatically coordinating the references among the workflow services. To discuss the use of concrete and flexible workflows, we analyze the execution of a median filter workflow in only one computational resource. Figure 10 shows the execution times for *zeus*, *cronos*, and *dionisio* to apply the filter to images with sizes between 0.75MB (500x500) and 12MB(2000x2000).

Figure 10 Execution time using one resource according to the image size



In the concrete workflow, if the user chooses *zeus* she has the worst execution option, while if the user chooses *cronos*, she has the best execution option. However, this depends on the situation of the grid at the moment of the execution. For example, if *cronos* is not available, the workflow will wait until it becomes available to be executed. Furthermore, if a new computational resource with a better performance joins the grid, the user cannot use it to execute this concrete workflow.

To create a flexible workflow, the only necessary step is to remove the localization of the computational resource in the service workflow definition. Figure 11 shows the

flexible workflow definition for IA-mf1 used in Figure 4 (excerpt 2). With the flexible workflow, the GST chooses the best resource available. In our example, if *cronos* is available, the GST will choose it, making the execution equal to the best choice that could be made by the user. On the other hand, if *cronos* is not available at the moment of the execution, the GST can use *dionisio*, whose performance is similar to *cronos*. In the concrete workflow there is no such possibility. Also, if new resources with best performance join the grid, they will be always used by the GST when they are available, without the need of user interference. Even in grids with low dynamicity the GST with its flexible workflows can help the user in the construction of the best concrete workflow. The user executes the flexible workflow during the time she thinks it is appropriate, and using the performance measurements made by the GST, she chooses the most suitable resources to generate the concrete workflow.

Figure 11 Flexible workflow

```
<gpo:gsh name="IA_mf1"
        uri="/factip/FactIPService/IPService"
        type="Factory"/>
```

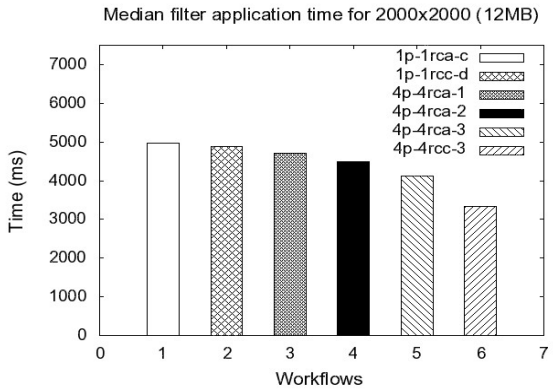
5.2 Evaluation of different sequential/parallel executions of a composition using the GST

Similarly to what happens when constructing traditional programs, some options in the construction of a service workflow may exist. When these constructions involve parallelism, these options are, in general, even wider. Here we show the GST as a support tool to refine the sequential/parallel execution performance in the workflows. To illustrate this functionality, we constructed workflows which apply the median filter to a 12MB image (2000x2000) using several combinations of parallelism and computational resources: **1p-1rca-c**: the workflow is executed from *apolo*, which sends the original file to *cronos*. *Cronos* applies the median filter and returns the resulting file to *apolo*. This does not use parallelism; **4p-4rca-1**: *apolo* splits the original file into four slices and sends them sequentially to *zeus*, *cronos* and *dionisio*. The filter is applied using *apolo*, *zeus*, *cronos* and *dionisio*. The resulting slices are copied back to *apolo*, which merges them generating the final file; **4p-4rca-2**: *apolo* splits the original file in four slices and sends them in parallel to *zeus*, *cronos* and *dionisio*. The filter is applied using *apolo*, *zeus*, *cronos* and *dionisio*. The resulting slices are copied back sequentially to *apolo*, which merges them generating the final file; **4p-4rca-3**: *apolo* splits the original file in four slices and sends them in parallel to *zeus*, *cronos* and *dionisio*. The filter is applied using *apolo*, *zeus*, *cronos* and *dionisio*. The resulting slices are copied back in parallel to *apolo*, which merges them generating the final file; **1p-1rcc-d**: the workflow is executed in *cronos*, which sends the file to *dionisio*. *Dionisio* applies the median filter and the resulting file is copied back to *cronos*; and **4p-4rcc-3**: *cronos* splits the original file in four slices and sends them in parallel to *zeus*, *cronos* and *dionisio*. The filter is applied using *apolo*, *zeus*, *cronos* and *dionisio*. The resulting slices are copied back in parallel to *apolo*, which merges them generating the final file.

Figure 12 shows the average times for 40 executions of each workflow. The results show that the filter applied to a single file in the same resource where the workflow is executed is slower. As we add parallel executions, the performance gets better. The best resource is obtained when we use the workflow manager in the resource with best performance applying the filter using parallelism in all operations. The measurements

made by the GST allow the analysis of the times and the observation of other interesting characteristics of the used algorithms. We can see that, for this workflow, there is no significant difference between the performance of *apolo* (1p-1rca-c) and *cronos* (1p-1rcc-d), when they are used by the workflow manager.

Figure 12 Median filter application using different sequential/parallel execution



5.3 Using the GST to evaluate scheduling policies

The GST functionalities can be used to evaluate scheduling algorithm options. Using the GST it is possible to measure the performance of each operation in the services as well as the performance of each GPOL operation involved (flow, sequence, etc) and the total execution time. This information feeds the resource and service repositories, which are used by the scheduler to make decisions. To show how the GST can be used in the tuning of scheduling policies used in the grid, we constructed an experiment where the workflows apply the median filter to an image of 12MB (2000x2000) using two resources. The file is split into two parts, and they are sent to two resources, processed, and the filtered files are returned to the resource where the workflow manager is executing. The resulting files are merged to a final file at this resource. The objective is to verify the possible combinations involving two or three resources and to use the results to refine the service scheduling algorithms. We used two workflow sets, shown in Table 2. The workflows in the left-hand side are executed with the GPO in *apolo* and the workflows in the right-hand side are executed with the GPO in *cronos*.

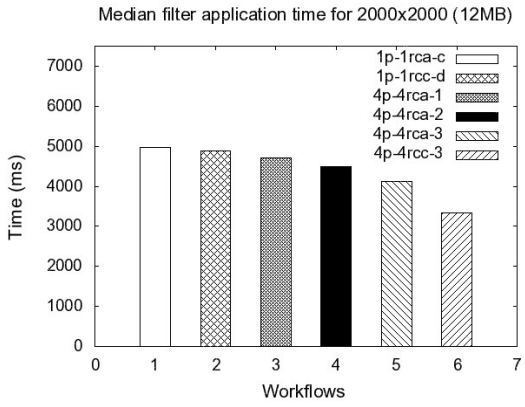
Table 2 Median filter execution with different resource configurations

Executed by apolo		*Executed by cronos*	
2p-2rca-dz	*dionisio* and *zeus*	2p-2rcc-dz	*dionisio* and *zeus*
2p-2rca-cz	*cronos* and *zeus*	2p-2rcc-cz	*cronos* and *zeus*
2p-2rca-cd	*cronos* and *dionisio*	2p-2rcc-cd	*cronos* and *dionisio*
2p-2rca-az	*apolo* and *zeus*	2p-2rcc-az	*apolo* and *zeus*
2p-2rca-aa	*apolo* and *apolo*	2p-2rcc-cc	*cronos* and *cronos*

Figure 13 shows the average times for 40 executions from *apolo* and *cronos*. The best results from *apolo*, as the workflow manager (2p-2rca-), are obtained when it is using *cronos* and *dionisio*, the faster machines. When everything is executed in *apolo*, we have the worst results. When we combine fast machines with slow ones we have similar performance as we have when used only the slower machines. This fact indicates a restriction to the scheduler. If at the moment of the scheduling decision there are 3 resources, 1 fast and 2 in the average, it is better to use the 2 average resources and leave the faster one to another task, since the performance would be the same.

In this figure we can also see that there is a significant improvement when we allocate the GPO workflow manager in *cronos* (2p-2rcc-). The executions are around 1 second faster, maintaining the same proportion between resources combination. In some sense, this fact indicates that the combination of resources with different performances are leveled by the processing power of the slowest machine, what should be avoided. Another interesting information is that *cronos* had the best performance when dealing with all the processing (2p-2rcc-cc). The contrary occurs when *apolo* made all the processing, which was the option that had the worst performance (2p-2rca-aa). This shows that if the resource has good performance, it can be considered to execute all tasks, avoiding file transfers.

Figure 13 Executions using *apolo* (2p-2rca) and *cronos* (2p-2rcc) as the workflow manager



Analyzing the average results we can observe other important characteristics to support the scheduler on the service allocation in the grid:

- Allocating services in the same resource depends on the resource performance. *Apolo* showed the worst performance, while *cronos* showed the best performance;
- Using computational resources with high performance difference may not be good. It may be better to use two average resources and keep the best resources to execute other applications; and
- The workflow manager should be allocated in resources with good performance. This is a characteristic to be added to the service repository, since the workflow manager is a service as well.

Author

5.4 Discussion

In the experiments presented here we show the use of dynamic instantiation of services as a tool to choose the best computational resources. We also show the GPOL usefulness in helping the refinement of compositions considering different sequential/parallel executions of the services, and how to evaluate scheduling policies using the GST. These simple experiments, in a some sense, show the flexibility of the GST as an environment to develop and test services and their compositions. The dynamic instantiation of services, the automatic coordination of references, and the GPOL resources allow the construction of other experiments to evaluate services and their compositions. Even in a large opportunistic grid, where resources can join and leave the grid, the GST functionalities allow the evaluation of the resources behaviour, while helping in the scheduling policies tuning according to the environment dynamism.

6 Related work

Testbeds have been explored in many different contexts. The PRAGMA (Pacific Rim Applications and Grid Middleware Assembly) provides an infrastructure and a collaborative environment for grid middleware and grid applications. (Abramson et al., 2006) shows how scientific applications are deployed in this testbed. The Panlab (Pan European Laboratory) (Gavras et al., 2006) enables the trial and evaluation of service concepts, technologies, system solutions and business models to the point where the risks associated with the launching of these as commercial products is minimized, however it does not use grids.

The problem of allocating tasks and workflows is discussed in (Blythe et al., 2004), which defines grid testbed requirements, describes several strategies for resource allocation in grids, and makes an empirical analysis of their performance using a grid simulation environment. In (Constantinescu et al., 2004) a synthetic testbed is proposed and used for simulating large deployments of services and service compositions, but here are no experimental results.

The execution of service compositions in grids is made by workflow engines and their performance depends on the scheduler. (MacLaren et al., 2004) describes an infrastructure for job scheduling on grids based on the notion of Service Level Agreements (Keller et al., 2002), but here are no practical experiments. Dynamic resource allocation mechanisms for grids are proposed in (Ismail et al., 2008). The mechanisms are based on a combination of the best fit algorithm with the process migration, but its focus is on tasks, thus not addressing service composition or workflows. (Lee et al., 2007) discusses efficient scheduling of workflow applications based on heuristics, and proposes a service model for predicting performance and an adaptive workflow scheduling strategy which uses maximum flow algorithms to relate services and users QoS requirements. The HAND middleware (Qi et al., 2007) proposes a dynamic service deployment mechanism. It is designed for the Globus Toolkit 4 and requires specific functions of the OGSA implementation. DynaGrid (Byun and Kim, 2007) is a framework to build grids for WSRF-compliant applications and provides mechanisms for dynamic service deployment, resource migration, and transparent request dispatching.

7 Conclusion

Computational grids are becoming a common way of sharing heterogeneous resources. Grids based on the OGSA use the SOA concepts to provide mechanisms to share resources through interfaces which can be remotely accessed to invoke services running on grid resources. The grid service paradigm brings new challenges for users and grid developers. In this paper we present the Grid Service Testbed (GST), an infrastructure for performance evaluation of grid service compositions in grid environments. Using the GST we show how the migration to the grid service paradigm can be done, and its impact on the execution time. To show the GST in action we use a typical grid application, an image processing filter, whose experimental results show that the use of a service composition scheme does not have a significant overhead. Moreover, a case study suggests that the migration from a sequential traditional application to a parallel execution in grids has worth, since the migration process can be done easily without modifying the original application and it can improve the services execution in parallel. Additionally, we demonstrate that dynamic service instantiation and automatic reference coordination can choose the best computational resources without user interference, offering suitable treatment for the dynamicity of the grid, specially for opportunistic grids. These features from the GST help users to choose the best resources and improve their compositions with sequential/parallel execution, besides showing how the developers may use the GST information to support scheduling decisions.

As on going work we are developing experiments including grid resources connected by an optical network. In addition, the use of more advanced scheduling algorithms to deal with heavy loads on faster resources is left as future work.

Acknowledgements

The authors would like to thank CAPES, FAPESP (05/59706-3) and CNPq for the financial support.

References

Abramson, D., Lynch, A., Takemiya, H., Tanimura, Y., Date, S., Nakamura, H., Jeong, K., Hwang, S., Zhu, J., hua Lu, Z., Amoreira, C., Baldridge, K., Lee, H.-C. Wang, C.-W. Shih, H.-L. Molina, T., Li, W.W. and Arzberger, P.W. (2006) "Deploying scientific applications to the pragma grid testbed: Strategies and lessons", in *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid(CCGRID 06)*, pp. 241–248.

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

Blythe, J., Gil,Y. and Deelman, E. (2004) "Coordinating workflows in shared grid environments", in *Proceedings of The 14th International Conference on Automated Planning and Scheduling (ICAPS 04)*.

Branco, K.R.L.J.C., Santana, M.J., Santana, R.H.C. and Junior, N.C. (2000) "A parallel programming supporting tool", *Intl. Conf. on Parallel and Distributed Processing Techniques and Applications PDPTA 00)*.

Byun, E., Kim, J. (2007) "DynaGrid: A dynamic service deployment and resource migration framework for WSRF-compliant applications", *Parallel Computing*, Vol. 33 Issues 4-5, February 20, pp. 328-338.

Constantinescu, I., Faltings B. and Binder, W. (2004) "Large scale testbed for type compatible service composition", in *Proceedings of The 14th International Conference on Automated Planning and Scheduling (ICAPS 04)*.

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

Foster, I., Kesselman, C. and Tuecke, S. (2002) "The physiology of the grid: An open grid services architecture for distributed systems integration", Available at: http://www.globus.org/research/papers/ogsa.pdf.

Foster, I. (2005) "Globus toolkit version 4: Software for service-oriented systems", *IFIP Intl. Conf. on Network and Parallel Computing,* Springer-Verlag LNCS 3779, pp. 2–13.

Gavras, A., Brüggemann, H., Witaszek, D., Sunell, K. and Jimenez, J. (2006) "Pan european laboratory for next generation networks and services", in *2nd Intl. Conf. on Testbeds & Research Infrastructures for the Development of Networks & Communities (Tridentcom 06)*.

Ismail, L., Mills,B. and Hennebelle, A. (2008) "A formal model of dynamic resource allocation in grid computing environment", in *International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp. 685–693.

Java SE (2009) Sun Microsystems, "Java platform, standard edition (Java SE), Available at: http://java.sun.com/javase/

Keller, E., Kar, G., Ludwig, H., Dan, A. and J. L. Hellerstein, J.L. (2002) "Managing dynamic services: A contract based approach to a conceptual architecture", in *8th IEEE/IFIP Network Operations and Management Symposium (NOMS 02)*, pp. 513–528.

Lee, J.-H., Chin, S.-H., Lee, H.-M., Yoon, T., Chung, K.-S and Yu, H.-C (2007) "Adaptive workflow scheduling strategy in service-based grids", in GPC, ser. *Lecture Notes in Computer Science*, Cérin C. and Li, K.-C, Eds., vol. 4459. Springer, 2007, pp. 298–309.

Lindley, C.A. (1991) *Practical image processing in C: acquisition, manipulation and storage: hardware, software, images and text,* New York, NY, USA: John Wiley & Sons, Inc.

MacLaren, J., Sakellariou, R., Krishnakumar, K.T., Garibaldi, J. and Ouelhadj, D. (2004) "Towards service level agreement based scheduling on the grid", in *Proceedings of The 14th International Conference on Automated Planning and Scheduling (ICAPS 04)*.

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

OASIS WSRF (2006) "Web Services Resource (WS-Resource) v1.2, OASIS Organization for the Advancement of Structured Information Standards", Available at: http://docs.oasis-open.org/wsrf/wsrf-ws resource-1.2-spec-os.pdf.

OGF (2009) The Open Grid Forum, Available at: http://www.ogf.org/

Qi, L., Jin, H., Foster, I., Gawor, J. (2007) "HAND: Highly Avaiable Dynamic Deployment Infrastruture for Globus Toolkit 4", in *Proc. Of The 15th Euromicro Int. Conf. Parallel, Distributed and Network-Based Processing (PDP 07)*, IEEE CS Press, pp. 155-162.

Schulze, B., Coulson, G., Nandkumar, R. and Henderson, P. (2007) "Special issue: Middleware for grid computing: A 'possible future': Editorials", *Concurrency and Computation: Practice and Experience*, vol. 19, no. 14, pp. 1879–1884.

Senna C.R. and Madeira, E.R.M. (2007) "A middleware for instrument and service orchestration in computational grids", in *Proc. Of The 7th IEEE Intl. Symposium on Cluster Computing and the Grid (CCGrid 07).*

Senna, C.R., Bittencourt, L.F. and Madeira, E.R.M. (2009) "Execution of Service Workflows in Grid Environments", in *Proc. Of The 5th International Conference on Testbeds and Research Infrastructure for the Development of Networks & Communities and Workshops (Tridentcom 09)*.