QoS Support for Time-Critical Grid Workflow Applications

Ivona Brandic, Siegfried Benkner, Gerhard Engelbrecht, Rainer Schmidt
Institute of Scientific Computing, University of Vienna,
Nordbergstrasse 15, A-1090 Vienna, Austria
e-mail: {brandic, sigi, gerry, rainer}@par.univie.ac.at

Abstract

Time critical grid applications as for example simulations for medical surgery or disaster recovery have special Quality of Service requirements. The Vienna Grid Environment, developed and evaluated in the context of the EU Project GEMSS, facilitates the provision of HPC applications as QoS-aware Grid services by providing support for dynamic negotiation of various QoS guarantees like required execution time and price. In this paper, we extend the QoS mechanisms offered by the Vienna Grid Environment to workflow applications. We describe QoS extensions of the Business Process Execution Language and present a first prototype of a corresponding QoS-aware workflow engine which implements different strategies in order to bind the tasks of a workflow to adequate Grid services subject to user-specified QoS constraints. We present different Grid workflow planning approaches as well as first experimental results.

1 Introduction

Service oriented grid computing is a promising technology for providing on-demand access to HPC applications [12]. Time critical applications as for example medical simulation applications for near real-time surgical support have special requirements with respect to Quality of Service (QoS). For such applications it is crucial to know at which point in time the results of remote simulation tasks executed on some Grid hosts will be available. In order to address this issue, we have developed a QoS-enabled service provision framework, the Vienna Grid Environment (VGE) [3]. VGE enables a service provider to expose an HPC application, usually a parallel MPI code installed on a cluster, as a service which is capable of dynamically negotiating QoS guarantees with respect to response time, price and others with clients. VGE is based on standard Web Services technology and has been utilized and evaluated in the context of the EU Project GEMSS [5, 13], which developed Grid middleware and a testbed for medical simulation applications.

In this paper we address various issues of extending the basic QoS support developed within VGE and GEMSS to Grid workflow applications. In order to enable QoS-aware service composition and workflow execution, we extend our VGE environment with a QoS-aware Grid Workflow Language (QoWL) and a QoS-aware Grid Workflow Engine (QWE) [7]. QoWL is defined as a subset of Business Process Execution Language (BPEL) with QoS extensions. Using QoWL, users may specify different QoS constraints addressing the overall workflow or individual workflow tasks (i.e. service invocations). The QWE comprises a workflow planning component, which performs QoS negotiation and service selection, and a workflow execution component which executes the workflow by invoking the selected services. Based on the specified QoS constraints, the QWE negotiates with multiple candidate Grid services to select appropriate services which satisfy the specified QoS constraints or attempts to derive constraints for those parts of the workflow where the user initially did not specify constraints. QoS negotiation is either performed statically prior to executing the workflow, or, if this is not feasible, during workflow execution.

Specifying QoS constraints of a workflow makes only sense if the workflow engine can choose between different candidate Grid services providing different QoS characteristics. Therefore, a QoWL workflow is usually specified in an abstract form containing workflow tasks, which are independent of the actual Grid resources. Executable QoWL workflows contain only tasks with assigned Grid resources. In this paper we describe the transformation process of abstract QoWL workflows into executable workflows, which we call a workflow planning process and investigate different workflow planing approaches considering different QoS annotation strategies.

The remainder of this paper is structured as follows: Section 2 gives an overview about related work. Section 3 describes QoS-aware grid services as provided by the Vienna Grid Environment. Section 4 presents QoS extensions for BPEL and describes different workflow annotation strategies as well as abstract and concrete QoWL workflows. Section 5 provides an overview of the QoS-aware Grid Workflow Engine (QWE) and presents different grid workflow planning strategies. Section 6 presents first experimental results. Conclusion and outlook to future work are given in Section 7.

2 Related Work

Numerous projects like Business Process Execution Language (BPEL) [2], Triana [17], FreeFluo [19] and GridBus [20], to name a few, are contributing to the large body of work in the area of service-based Grid workflow technology. There are currently several projects dealing with QoS support for workflows. Abraham et al. examine nature's heuristics for scheduling jobs on computational grids comprising genetic algorithms, simulated annealing, tabu search, as well as different hybrid approaches [1]. Blyte et al. analyse scheduling strategies for workflow-based applications, mainly data-intensive applications distinguishing between task-based algorithms, that greedily allocate tasks to resources and workflow-based algorithms, that search for an efficient allocation for the entire workflow using min-min and weighted min-min algorithms [6]. The prediction of data transfer time is calculated using bandwidth and data size, whereas the prediction of the computational time is based on the queue length. Similarly, Cao et al. propose global grid workflow management and local grid subworkflow scheduling [10]. Cardoso et al. elaborate theoretical concepts of a QoS-aware workflow defining QoS workflow metrics [11]. Zeng et al. investigate QoS-aware composition of Web Services using integer programming method [21]. The services are scheduled using local planning, global planning and integer programming approaches. The execution time prediction of Web Services is calculated using an arithmetic mean of the historical invocations. Since Web Services require rather small data transfers, the prediction of data transfer time is neglected. Canfora et al. [9] compare the usage of genetic algorithms and integer programming approaches for the selection of the appropriate Web Services in the context of a Web Service workflow by specified global constraints. Similar to [21] neither application based performance models nor performance prediction of data transfer are considered. The Gria Project develops middleware which enables commercial use of the Grid in a secure, interoperable and flexible manner [14].

Buyya proposes a Grid Architecture for Computational Economy (GRACE) considering a generic way to map economic models into a distributed system architecture [8]. The Grid resource broker (Nimrod-G) supports deadline and budget based scheduling of Grid resources.

The main contribution of our paper is the novel QoS support for Grid workflows. We assume a Grid environment with a number of QoS-aware Grid services matching the same search criteria but offering different QoS characteristics (e.g. cheap and slow services, expensive and fast services). The selection of appropriate services is based on a negotiation process between the workflow engine and multiple service providers. As opposed to many other approaches, which are usually based on resource usage only, performance prediction for QoS-aware VGE services is based on application-specific performance models which depend on meta data about the input data (e.g. matrix size) supplied by the client during QoS negotiation. In the next section, the main features of QoS-aware VGE services are explained in more detail.

3 QoS support for Grid Services

An important prerequisite for the development of QoS support for Grid workflows are QoS-aware Grid services which are capable of guaranteeing certain QoS criteria. The Vienna Grid Environment (VGE) [3] is a service-oriented Grid infrastructure for the provision of HPC applications as QoS-aware Grid services. VGE relies on standard Web Services technologies such as WSDL, SOAP, WS-Security, Tomcat and Axis. VGE services enclose native HPC applications, usually parallel MPI codes running on a cluster, and expose their functionality via a set of common operations for job execution, job monitoring, data staging and error recovery.

In addition, VGE services may be configured in order to offer QoS guarantees with respect to response time, price and others. VGE services support a dynamic QoS negotiation model where clients may negotiate various QoS guarantees with multiple service providers [4]. VGE services rely on a generic QoS module which usually comprises an *application-specific performance model*, a *pricing model*, a *compute resource manager* that supports advance reservation, and a *QoS manager*. An application-specific performance model usually takes as input a *request descriptor*, containing meta data about the actual input data of a request, and a *machine descriptor* which specifies the amount of machine resources (e.g. number of processors, main memory, etc.) that may be provided for an application.

The basic QoS negotiation scenario as shown in Figure 1 is based on a request/offer model, where a client requests offers from services. Upon start of a QoS negotiation, the client specifies the desired QoS constraints (e.g. earliest begin time of a job, latest finish time) within a *QoS request*. Furthermore, the client provides a *request descriptor* containing input meta data for a specific service request. For example, in the case of a FEM simulation, input meta data in the request descriptor would typically include the size of the finite-element model, the number of iterations to be performed, etc.

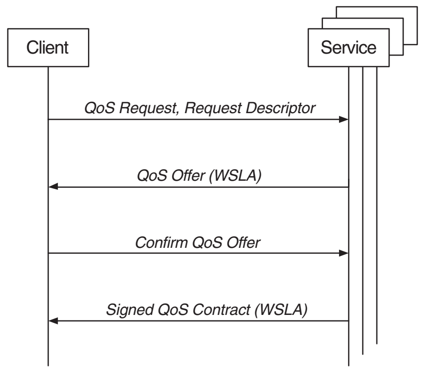

Figure 1. Basic QoS negotiation scenario.

On the service-side, the input meta data is fed into an application-specific performance model in order to obtain an estimate for the required execution time. In case of parallel MPI applications, the QoS manager, which controls the QoS negotiation on the service-side, uses heuristics to determine the number of processors required to execute a service request within the user-specified time constraints. The QoS manager then interacts with the compute resource manager to check whether a reservation of the required machine resources can be made at the required time. If QoS support for the price of a service request is required, the QoS manager subsequently invokes the pricing model to check whether the client's price constraints can be met. If all QoS constraints can be fulfilled, a temporary resource reservation is made and a corresponding QoS offer in the form of a Web Service Level Agreement (WSLA) document is returned to the client. Only when the client, which usually negotiates with multiple service providers to get the best deal, confirms an offer, a signed QoS contract in the form of a WSLA is established. Temporary reservations for offers that are not confirmed by the client expire within a short time frame.

When a client has successfully negotiated the required QoS with a service provider and a QoS contract is in place, the usual job execution phase can be entered, which comprises the invocation of service operations for `upload`ing the input data, for `start`ing the job execution and for `download`ing the result. In order to support direct data transfer between services, corresponding `push` and `pull` operations are supported as well.

The generic QoS module of VGE enables the provision of parallel applications as dynamically configurable Grid services. Depending on the requirements of a client, an application may be executed on many processors in short time but for a higher price, or it may be executed on a few processors with a lower price.

The VGE service provision model and QoS support has been successfully applied within the European GEMSS project for the development of six medical Grid applications [16] which utilize computationally demanding methods such as parallel Finite Element Modeling, parallel Computational Fluid Dynamics and parallel Monte Carlo simulation, realized as remote Grid services running on clusters or other parallel computing platforms.

In the next section we extend the QoS support of VGE services to Grid workflows.

4 QoS-aware Grid Workflows

The basic Grid workflow scenario assumes a coupling of several Grid applications, where the output data of the first grid application (A_1) is the input data of the second one (A_2) (see Figure 2 (a)). This scenario may be specified as a Grid workflow exemplified in Figure 2 (b).

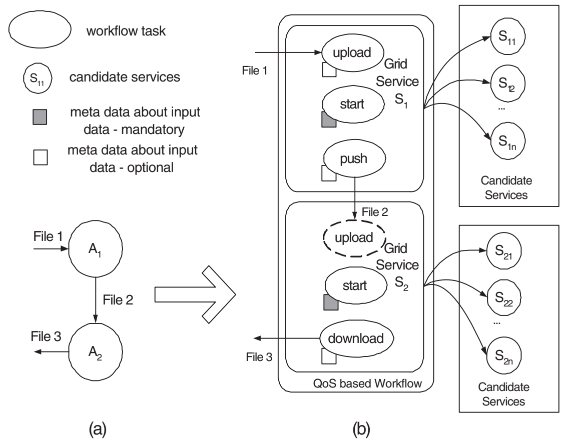

Figure 2. QoS-aware Grid Workflow Example

Due to the nature of VGE services, we consider a set of coherent tasks (e.g. upload, start, download), which form one complete invocation of a Grid application service (e.g. S_1). Each set of tasks may match several candidate services (S_{11}, S_{12}, ..., S_{1n}) which are discovered via a service registry. As shown in Figure 2 (b), an invocation of a Grid service implies at least one *start* task, one *upload* (or *pull*) task and one *download* (or *push*) task. The first task of S_1 uploads an input file, the second task starts the execution of the native application. The third task transfers the output data of the first application to the second one. The fourth task starts the execution of the second application, and finally, the fifth task downloads the results. The last three tasks represent the Grid service S_2. The *upload* task of S_2 is implicitly invoked by the *push* task of S_1. The workflow shown in Figure 2 (b) currently has to be written manually using a subset of BPEL. An automatic tool which transforms the graphical representation (Fig.1 a) into a workflow is currently under development.

In the following we describe the QoS-aware Grid Workflow Language (QoWL) which provides support for annotating workflows with QoS constraints.

4.1 QoS Annotation Strategies

The required QoS constraints of a workflow may be specified in different ways. As shown in Figure 3, we distinguish between *local* and *global* constraints of a Grid workflow. Those constraints marked with question mark have to be calculated by the QoS-aware engine.

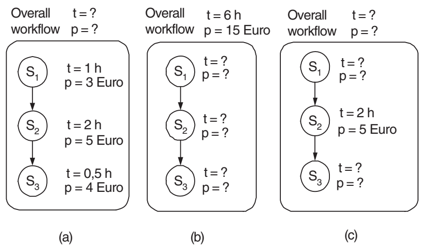

Figure 3. QoS workflow annotation strategies

Local constraints usually address QoS constraints of single tasks invoking external services. As shown in Figure 3 (a) the QoS constraints of the basic activites S_1, S_2 and S_3 are specified by the user. The QoS constraints of the overall workflow are calculated in the *bottom-up* way considering aggregation functions of the particular tasks. We use aggregation functions described in [9, 21] for the recursive computation of the QoS constraints of the overall workflow.

Global constraints address the QoS of the overall workflow or of composite activities. Figure 3 (b) depicts such approach where the QoS constraints of the overall workflow are specified by the user. The appropriate services for the underlying tasks (S_1, S_2, S_3) are selected using a heuristic or analytical solver where the global constraints of the workflow correspond to the constraints of the solver (e.g. maximum execution time or budget for the overall workflow). Global constraints may be used to annotate composite tasks, e.g. if the concerning composite task encompasses some critical simple tasks (Figure 3 (c)). If we assume, that the task S_2 shown in Figure 3 (c) is a composite task, we may apply a global constraints approach in order to bind underlying simple tasks. Constraints of the remaining workflow are computed in a *bottom-up* approach considering the aggregation function of the particular workflow elements. Thus, the user will be informed about the overall QoS constraints of a workflow.

4.2 QoS-Aware Grid Workflow Language

For the specification of QoS constraints for Grid workflow applications we propose QoWL, a QoS-aware Grid Workflow Language, which utilizes a subset of the Business Process Execution Language (BPEL) [2] with extensions for expressing QoS constraints. In the following we briefly describe the BPEL subset used in QoWL and the QoS extensions.

4.2.1 BPEL Subset

BPEL is an XML-based workflow language that uses WSDL documents in order to invoke Web Services. The basic language construct is an `activity` which is a corresponding element to a `task` element of a workflow. BPEL distinguishes between basic and complex activities. Basic activities perform primitive actions such as invoking other services with the `<invoke>` activity, receiving a request from a client with the `<receive>` activity or sending a reply to a client with the `<reply>` activity. Complex activities group basic or other complex activities defining a special behavior of that group of services. The `<sequence>` activity executes activities in the given order, the `<flow>` activity concurrently executes underlying activities, the `<while>` activity iterates until some condition is violated and the `<switch>` activity executes one of the specified branches based on the evaluated condition. Based on the BPEL subset we define the QoS extensions as described next.

4.2.2 QoS Extensions

QoS extensions are necessary to express both the *requested* QoS constraints of a workflow before the QoS negotiation and the *offered* QoS of a workflow after the negotiation with the services since offered QoS may differ from the requested one. Therefore, we distinguish between *abstract QoWL workflows* and *concrete QoWL workflows*. An *abstract QoWL workflow* addresses only registries, where the potential services can be found, and requested QoS. *Abstract workflows* are specified before the QoS negotiation. The example below specifies an *abstract* workflow.

```
...
<invoke name="start" portType="appex"
   operation="start" inputVar="startRequest">
   <qos-constraints reqDescVar="startReqDesc">
      <candidate-registry inputVar="queryRequest"
        ...
        wsdl="http://kim:9357/registry/reg?wsdl"/>
      <qos-constraint name="beginTime" weight="0.3"
        value="18-08-2005 12:00:00,0 MET" />
      <qos-constraint name="endTime" weight="0.2"
        value="18-08-2005 14:00:00,0 MET" />
      <qos-constraint name="price" weight="0.5"
        value="20.00" />
   </qos-constraints>
</invoke>
...
```

QoS constraints of a workflow and of the underlying tasks are expressed in terms of `qos-constraints` elements which may contain several `qos-constraint` elements and several `candidate-registry` elements [1].

Each `qos-constraint` element defines a name/value pair and a weight of a QoS constraint. The `qos-constraint` element `beginTime` specifies the requested begin time of the workflow execution. The element `endTime` specifies the desired finish time of the workflow execution. The budget for the overall workflow execution is specified defining the `price` `qos-constraint` element. The `weight` constraint is a number between 0 and 1. The sum of all weights of a `qos-constraints` element should be equal to 1. The weight attribute balances the impact of different QoS constraints on the service selection process. The `candidate-registry` element specifies where the potential services may be found. Each `invoke` activity as well as other complex activities and the overall workflow activity may have several `qos-constraint` elements and several `candidate-registry` elements. In the example above there is one basic `start` activity among other elements containing one `qos-constraints` and one `candidate-registry` element. The meta data necessary for QoS negotiation of that activity is specified using the `startReqDesc` variable. The payload data is set using the `startRequest` variable. The piece of code below depicts a corresponding *concrete workflow*.

```
...
<invoke name="start" portType="appex"
   wsdl="http://bridge:9355/SPECT/appex?wsdl"
   operation="start" inputVar="startRequest">
  <qos-constraints reqDescVar="startReqDesc"
     wslaVar="wslaStart">
     <qos-constraint name="beginTime" weight="0.3"
       value="18-08-2005 12:13:06,0 MET" />
     <qos-constraint name="endTime" weight="0.2"
       value="18-08-2005 13:45:04,0 MET" />
     <qos-constraint name="price" weight="0.5"
       value="16.00" />
  </qos-constraints>
</invoke>
...
```

Now, in the concrete workflow, the `invoke` element `start` contains a `wsdl` attribute with the endpoint of the selected service instead of the `candidate-registry` element. The `qos-constraints` element now also contains the `wslaVar` variable specifying the Service Level Agreement between the engine and the particular service. All `qos-constraints` elements of the `invoke` activities consider offered QoS of a VGE service. The QoS constraints of the remaining elements are calculated in a recursive way starting with the QoS constraints of the invoke activities and considering the aggregation function of the particular workflow elements.

[1]We neglect the fact that workflow QoS can be expressed in context of a semantic workflow since semantic workflows are out of scope of this paper.

5 QoS-aware Grid Workflow Engine

Currently, we are developing a Java based QoS-aware Grid Workflow Engine (QWE). The engine transforms the abstract QoWL documents into an intermediary representation, performs QoS negotiation with potential service providers, generates concrete QoWL documents and, finally, executes QoWL workflows.

5.1 Architecture

The main architectural components of the QWE are depicted in Figure 4.

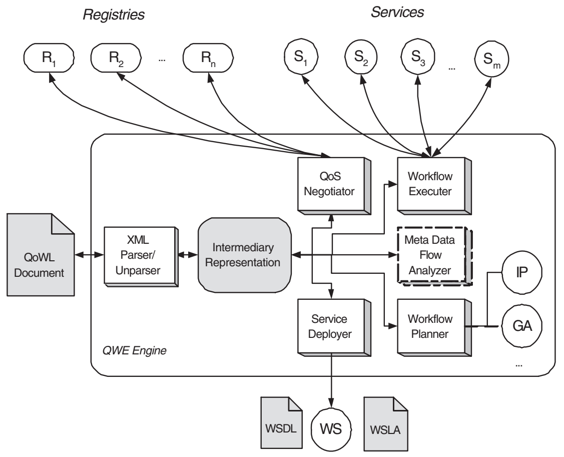

Figure 4. QoS-aware Grid Workflow Engine

The *XML parser and unparser* generates the intermediary representation of the QoWL workflow. The *QoS negotiator* queries the registries, generates necessary QoS requests and receives offers from services. The *Workflow planning* component calculates a workflow execution plan considering

- the QoS annotations (*local, global*)
- the chosen workflow planning strategy (*static, dynamic*) and
- the selected workflow planning technique (*Integer Programming, Genetic Algorithm, MCDM, etc.*).

The *Service Deployer* exposes a QoWL workflow as a Web service and the *Workflow Executer* starts the execution of the QoWL workflow. In case of dynamic planning strategy *WorkflowPlanner* and *WorkflowExecuter* are invoked in an alternating way.

5.2 Workflow Planner

An important part of the QWE is a flexible *Workflow Planning* component responsible for the transformation of abstract workflows into concrete workflows which satisfy the QoS constraints specified by the user.

We distinguish between static and dynamic workflow planning strategies. Static planning implies the generation of the concrete workflow before the execution of the first workflow activity.

In a dynamic planning approach the concrete parts of the workflow are not created until needed. Therefore the concrete activities of the workflow are generated only for ready-to-start activities. The decision, which of those techniques should be used, depends on the *meta data* of the invoked services. If all *meta data* required for performance prediction is statically known (e.g. matrix size), the static planning strategy can be chosen. If the *meta data* is generated or changed during workflow execution (e.g. by the predecessor activities), the dynamic planning approach has to be chosen. A *Meta Data Flow Analyzer* is currently being developed in order to determine which kind of the workflow planning approach (static, dynamic) is feasible by analyzing the meta data flow. At the moment, the user has to specify whether the static approach is feasible.

5.3 Static Planning

In case of *global* QoS constraints analytical or heuristic solvers are needed in order to find services, which satisfy the QoS constraints. Currently our implementation is restricted to straight-line code (i.e. loops are currently not handled), which, however, may include nested activities. We apply the Integer Programming Method using the *lp_solve* package [15]. Parameters of the objective function are calculated by unrolling `sequence` activities[3]. In case of the `switch` activity the constraints have to be satisfied for all branches. The solution of *lp_solve* is a workflow execution plan, which satisfies the specified constraints (e.g. budget) and includes one appropriate service for each task. Obviously, since finding such a solution is a NP-hard problem, there is no guarantee that the best and the most efficient solution is found.

The following objective function is considered:

$$Max\Bigg(\sum_{j=1}^{T}\Bigg(\sum_{i=1}^{S_j}\Bigg(\sum_{l=1}^{n}\Bigg(\frac{Q_l^{max}-Q_{i,l}}{Q_l^{max}-Q_l^{min}}*W_l\Bigg)*y_{ij}\Bigg)\Bigg)\Bigg) \quad (1)$$

where $Q_{i,l}$ is the parameter of i^{th} service and l^{th} criterion (e.g. second service and begin time). W_l is the weight of the criterion (e.g. 0.3 for begin time, 0.4 for price) where $\sum_{l=1}^{n} W_l = 1$. We denote y_{ij} as an integer variable with acceptable values 0 or 1, 1 if the service is selected, 0 otherwise. j denotes the j^{th} task of the workflow, which consists of T tasks. i is the i^{th} candidate service of the j^{th} task. Task j has S_j candidate services. Since we assume that only one service per task can be selected, we have $\sum_{i=1}^{S_j} y_{i,j} = 1$. The entire theoretical elaboration of the IP method is out of scope of this paper, interesting readers are referred to [21]. If *lp_solve* cannot find a solution, all ready-to-start activities are selected using local optima. Then *lp_solve* is invoked for all remaining activities, where no solution is found, and the global constraints are adapted considering the QoS constraints of the already selected services. This procedure is iterated until all activities have assigned services.

[3]Because of the non-linear aggregation function the complex activity `flow` is currently not handled.

If the input meta data is not statically known, a dynamic planning strategy as described next has to be applied.

5.4 Dynamic Planning

The dynamic planning strategy starts the workflow planning for ready-to-start activities only. The dynamic approach is sketched in Figure 5.

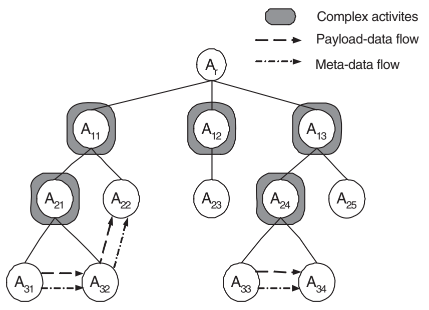

Figure 5. Dynamic workflow planning

First, the QoS negotiation is started for the leaf activities, which do not have meta data dependencies with any other activities, e.g. activity A_{31}. Dashed lines depict the data flow and the dashed dotted lines define the meta data flow, which mostly follows the data flow. The services are selected using the Multiple Criteria Decision Making (MCDM) mechanisms [21]. After the execution of A_{31} the QoS negotiation of A_{32} is started and the activity is finally executed. In the next step, the QoS of the A_{21} is calculated and then the A_{22} is executed and so on. Dynamic workflow planning leverages the late binding process of grid services. Currently, we do not perform payload data and meta data flow analysis. We assume that the meta data is available just before the task execution.

6 Experiments

In this section we present first experimental results for an artificial reference workflow as shown in Figure 2. The time for the QoS negotiation, workflow planning and workflow execution is measured and compared for both the static and dynamic approach.

In our experimental setup, all files transferred between services and engine have the size of 15 MB. First an `upload` activity uploads an input file to the Grid service S_1. Next the *start* operation is invoked. We simulate the invocation of the native application by starting a Java program, which generates the output file with the same size as the input file (15 MB). Then the `push` activity is invoked, which uploads the output data to the second service S_2. In the next step, the *start* operation of S_2 is invoked and finally the output file of the second service is downloaded. In this experiments we realized QoS support for the *start* as well as *upload*, *download* and *push* activities. The QoS constraints of the workflow have been globally specified.

We installed services, registries, the client, and the QWE on a heterogeneous system. The 16 candidate service of S_1 and S_2 are installed on 3 different Sun Blade 1500 (UltraSPARC-IIIi 1062 MHz) machines with 1024 MB RAM, on 7 different Sun Blade 100 (UltraSPARC-IIe) with 640 MB RAM, on 5 different Sun Blade 150 (UltraSPARC-IIe 550 MHz) with 768 MB RAM and on 1 Ultra 10/333 (UltraSPARC-IIi 333 MHz) with 768 MB RAM. Therefore we have 16 services, which are registered in 2 different registries. The QWE is installed on a Sun Fire V880 server with 4 sparcv9 processors with 750 MHz and 8 GB RAM. All machines are running under Solaris 9. The workflow client is running on a Windows 2000 PC with a 2.4 Ghz Intel Pentium4 Processor and 1024 MB RAM. The machines are connected over Fast Ethernet (100 MBit) on a SMC 6912M Tiger Switch that supports 4.4 GBits aggregate bandwidth.

Table 1 shows the QoS results of the static workflow planning approach before and after the QoS negotiation.

	req. QoS	**pred. QoS**	**measured QoS**
WF time	5 min	4 min 51 sec	4 min 52 sec
WF price	20 Euro	17.2 Euro	17.2 Euro

Table 1. Results of the static planning approach

In this experiment, the user-requested time and price constraints for the workflow execution are 5 min and 20 Euro. During the workflow planning approach, the registries are queried, and the offers of all 16 services are evaluated.

The QoS offer generated by the workflow engine comprised 4 min and 51 sec for the execution and 17.2 Euro for the price. As shown in Table 2, the measured time of the workflow execution was slightly larger than the predicted one. In this experiment, the solution was found already in the first iteration (i.e. lp_solve was called only once). The time to transfer the files between S_1 and S_2 (push) is smaller than the time required for the upload and download, since S_1 and S_2 share a common file system. The overall workflow time comprises the time for the execution of all tasks listed in Table 2 plus the computational time of the workflow.

QoS negotiation S_1, S_2	4.145 sec
lp_solve	0.692 sec
upload S_1	13.891 sec
start S_1	2 min 10.037 sec
push S_1	0.580 sec
start S_2	2 min 10.044 sec
download S_2	13.630 sec
overall WF	**4 min 51.722 sec**

Table 2. Static workflow planning and execution

In case of the dynamic planning approach, planning and QoS negotiation is carried out during workflow execution. Therefore, as expected, the overall time for the workflow execution is larger than in the static case.

QoS negotiation S_1	3.572 sec
upload S_1	13.791 sec
start S_1	2 min 10.033 sec
QoS negotiation S_2	3.343 sec
push S_1	1.112 sec
start S_2	2 min 10.058 sec
download S_2	13.998 sec
overall WF	**4 min 59.198 sec**

Table 3. Dynamic workflow planning and execution

As shown in Table 3, with dynamic planning the overall workflow execution time is 4 min 59 sec.

Also note that in the static case, the time for QoS negotiation with the 16 candidate services is smaller than in the dynamic case. In the static case, the negotiation with all 16 services was performed in parallel, while in the dynamic case the QWE first negotiated with the 8 candidate services of S_1 and then with the eight candidate services of S_2.

7 Conclusion and Future Work

In this paper we presented our ongoing work on QoS support for time-critical Grid workflow applications. Using the proposed QoS-aware Grid Workflow Language, users may specify QoS constraints for a Grid workflow. Based on these constraints, a QoS-aware Grid Workflow Engine negotiates with multiple candidate services to find services that fulfill the specified QoS constraints. We implemented two preliminary workflow planning approaches and presented initial experimental results. In the future we will extend our workflow planning strategies by incorporating data flow analyses techniques originally developed for traditional compilers and experiment with additional planning strategies.

Acknowledgments

The work was partially supported by the Austrian Science Fund as part of the AURORA project under contract SFB F011-02 and by the European Union's GEMSS (Grid-Enabled Medical Simulation Services) Project under contract IST 2001-37153.

References

[1] A. Abraham, R. Buyya, and B. Nath. *Nature's Heuristics for Scheduling Jobs on Computational Grids*. The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), December 14-16, 2000, Cochin, India.

[2] BPEL Specification, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/ 2003.

[3] S. Benkner, I. Brandic, G. Engelbrecht, R. Schmidt. *VGE - A Service-Oriented Grid Environment for On-Demand Supercomputing*. Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004), Pittsburgh, PA, USA, November 2004.

[4] S. Benkner, G. Engelbrecht. *Generic QoS Support for Application Web Services*. Proceedings of the International Symposium on Web Services, CSREA Press, Las Vegas, USA, June 2005.

[5] S. Benkner, G. Berti, G. Engelbrecht, J. Fingberg, G. Kohring, S. E. Middleton, R. Schmidt: *GEMSS: Grid-infrastructure for Medical Service Provision*, Methods of Information in Medicine 2005; 44: 177-181, Schattauer Publishers, 2005.

[6] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy. *Task Scheduling Strategies for Workflow-based Applications in Grids*, CCGrid 2005, Cardiff, UK.

[7] I. Brandic, S. Benkner, G. Engelbrecht, R. Schmidt. *Towards Quality of Service Support for Grid Workflows.* Proceedings European Grid Conference 2005 (EGC2005), Amsterdam, The Netherlands, February 2005.

[8] R. Buyya. *Economic-based Distributed Resource Management and Scheduling for Grid Computing*, Ph.D Thesis, Monash University, Melbourne, Australia, 2002.

[9] G. Canfora, M. Di Penta, R. Esposito, M. L. Villani. *An Approach for QoS-aware Service Composition based on Genetic Algorithms*. Proceedings of the Genetic and Computation Conference (GECCO 2005), Washington, DC, ACM Press, 2005.

[10] J. Cao, S. A. Jarvis, S. Saini, G. R. Nudd. *GridFlow: Workflow Management for Grid Computing*, 3rd International Symposium on Cluster Computing, Tokyo, Japan, 2003.

[11] J. Cardoso, A. Sheth, and J. Miller. *Workflow Quality of Service*, Enterprise Inter- and Intra-Organisational Integration - Building International Consensus, Kluwer Academic Publishers 2002.

[12] I. Foster, C. Kesselman, J. Nick, S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[13] The GEMSS Project: Grid-Enabled Medical Simulation Services, EU IST Project, IST-2001-37153, http://www.gemss.de/

[14] The GRIA Project http://www.gria.org/ 2005.

[15] The lp_solve Project http://www.geocities.com/lpsolve 2005.

[16] D. M. Jones, J. W. Fenner, G. Berti, F. Kruggel, R. A. Mehrem, W. Backfrieder, R. Moore, A. Geltmeier. *The GEMSS Grid: An evolving HPC Environment for Medical Applications*, HealthGrid 2004, Clermont-Ferrand, France, 2004.

[17] S. Majithia, M. S. Shields, I. J. Taylor, I. Wang. *Triana: A Graphical Web Service Composition and Execution Toolkit*. International Conference on Web Services, San Diego, USA, 2004.

[18] Web Service Level Agreement (WSLA) Language Specification. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, IBM 2001-2003.

[19] T. Oinn, M. Addis, J. Ferris et al. *Delivering web service coordination capability to users*. Proceedings of the 13th international World Wide Web conference, ACM Press, New York, NY, USA, 2004.

[20] J. Yu and R. Buyya. *A Novel Architecture for Realizing Grid Workflow using Tuple Spaces*. Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004, Nov. 8, 2004, Pittsburgh, USA), IEEE Computer Society Press, Los Alamitos, CA, USA, 2004.

[21] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, H. Chang. *QoS-Aware Middleware for Web Services Composition*. IEEE Transactions on Software Engineering, vol. 30, no. 5, pp. 311-327, May 2004.