Scientific Programming 15 (2007) 213–234
IOS Press

GRIDCC: A Real-time Grid workflow system with QoS

A. Stephen McGough[a], Asif Akram[a], Li Guo[a], Marko Krznaric[a], Luke Dickens[a], David Colling[b], Janusz Martyniak[b], Roger Powell[c], Paul Kyberd[c], Chenxi Huang[c], Constantinos Kotsokalis[d] and Panayiotis Tsanakas[d]
[a]*London e-Science Centre, Imperial College London, London, UK*
E-mail: {asm, aakram1, liguo, marko, lwd03}@doc.ic.ac.uk
[b]*High Energy Physics Group, Imperial College London, London, UK*
E-mail: {d.colling, janusz.martyniak}@imperial.ac.uk
[c]*Brunel University, School of Engineering & Design, Uxbridge, UK*
E-mail: {Roger.Powell, Paul.Kyberd, Chenxi.Huang}@brunel.ac.uk
[d]*Computing Systems Laboratory, National Technical University of Athens, Athens, Greece*
E-mail: ckotso@cslab.ece.grnet.gr, panag@cs.ntua.gr

Abstract. The over-arching aim of Grid computing is to move computational resources from individual institutions where they can only be used for in-house work, to a more open vision of vast online ubiquitous 'virtual computational' resources which support individuals and collaborative projects. A major step towards realizing this vision is the provision of instrumentation – such as telescopes, accelerators or electrical power stations – as Grid resources, and the tools to manage these resources online. The GRIDCC project attempts to satisfy these requirements by providing the following four co-dependent components; a flexible wrapper for publishing instruments as Grid resources; workflow support for the orchestration of multiple Grid resources in a timely manner; the machinery to make reservation agreements on Grid resources; and the facility to satisfy quality of service (QoS) requirements on elements within workflows. In this paper we detail the set of services developed as part of the GRIDCC project to provide the last three of these components. We provide a detailed architecture for these services along with experimental results from load testing experiments. These services are currently deployed as a test-bed at a number of institutions across Europe, and are poised to provide a 'virtual lab' to production level applications.

1. Introduction

The Grid [21] has emerged in recent years as paradigm for allowing users, potentially as members of distributed communities – often referred to as Virtual Organisations, to obtain access to large quantities of computing or storage resources while providing a forum in which resource owners make their services available to a wider audience. The Grid has evolved into a Service Orientated Architecture (SOA) [29] with Web Services [36] emerging as the de-facto communication mechanism. With the increase of users and communities within the Grid there is a drive to support a broader range of resource types. Thus far most Grids have focused on making computational and, more recently, data resources available. However, for the Grid to gain mass adoption within the wider scientific and commercial communities there is a need to integrate instruments into the Grid. In this paper we define an instrument to be any piece of equipment that can be controlled through a computer interface such as telescopes, particle accelerators or electricity power stations. In the rest of this paper we shall discuss scientific instruments, however, the principals may be mapped easily to other instrument types.

In most cases when a scientist performs an experiment this will require the completion of many tasks. These may include such things as configuring an instrument, collecting and storing relevant data from the instrument, processing of this information and potentially further iterations of these tasks. These tasks, along with a description of how they interact in order to achieve the final result, define a workflow. Many scientists now seek to automate their workflow processes through the use of the Grid and Grid enabled resources. This desire dramatically increases the need for workflows and real-time-support within the Grid. Here we define real-time-support to mean the capability to make resources available to the user in a timely manner.

We present here the Grid-Enabled Remote Instrumentation with Distributed Control and Computation (GRIDCC) project [16]. More specifically we present here the real-time-control of existing Grid resources and instruments elements from the project. The GRIDCC project is a 3 year project funded by the European Union which started in September 2004. There are 10 project partners from Greece, Italy, Israel and the United Kingdom. GRIDCC is extending the Grid to include access to, and control of, distributed instruments. Instruments work in real-time and their successful operation often requires rapid interaction with conventional computing/storage resources and/or with other instruments. The real-time and interactive nature of instrument control provides a critical requirement for the definition of acceptable Quality of Service (QoS) constraints for interactions between the different Grid components.

In this paper we present those parts of the GRIDCC architecture responsible for providing workflow and QoS support. In Section 2 we present related work. In Section 3 we present the architecture for a real-time QoS aware workflow management system along with the workflow pipeline. This we follow by a more detailed breakdown of the architecture: the workflow editor with QoS support – Section 4, QoS aware workflow optimisation – Section 5, reservation services – Section 6 and performance repository – Section 6. In Section 8 we evaluate the performance of our architecture before concluding in Section 9.

2. Related work

Many Grid workflow systems and tools exist, such as: Askalon [20], DAGMan [33], GridFlow [12], Gridbus Workflow [39], ICENI [25,28], Pegasus [18], and Triana [34]. Yu and Buyya [40] present a detailed survey of existing systems and provide a taxonomy which can be used for classifying and characterising workflow systems. We use elements of the taxonomy related to Quality of Service (QoS) and workflow modelling for comparing our approach to other systems.

A prominent feature of adding instruments to the Grid is the need for real-time remote control and monitoring of instrumentation. Users and applications will be allowed to make an *Advance Reservation* (AR) of computational resources, storage elements, network bandwidth and instruments. AR guarantees availability of resources and instruments at times specified [5]. In ICENI, the scheduling framework supports AR by using the meta-data of application components together with corresponding historical data stored for the purpose of performance analysis and enhancement [27, 37,38]. This approach is also used in Pegasus, Gridbus Workflow and Askalon. Here we adopt the idea of a performance repository of historical information and extend this with other factors about a service such as its availability and reliability. Some systems such as Askalon, DAGMan, ICENI, GridFlow and Gridbus Workflow allow users to define their own QoS parameters and to specify optimization metrics such as application execution time, desired resources and economical cost. We are using a similar mechanism for performance enhancement and estimation taking into account the real-time QoS constraints. We further extend this by bringing in the ideas of hard (compulsory) and soft (non-compulsory) QoS parameters. This allows for both feature critical QoS satisfaction (such as time-critical) and the support of preferential QoS satisfaction. For those cases where meeting the QoS requirements is preferable though not mission critical.

Workflow languages such as Business Process Execution Language for Web Services (BPEL4WS) [6], YAWL [35] and WS-Choreography [10] are powerful languages for developing workflows based on Web Services. However, they do not provide a mechanism for describing QoS requirements. Our approach to overcome this has been to develop a partner language to use with BPEL4WS. Instead of defining a new language for workflows with QoS requirements, or embedding QoS requirements within a language such as BPEL4WS, we use a standard BPEL4WS document along with a second document which points to elements within the BPEL4WS document. This allows us to annotate these elements of the BPEL4WS document with QoS requirements. This allows us to take advantage of standard BPEL4WS tooling for execution and manipulation as well as support QoS requirements. As we use XPath [14] notation to reference elements within the BPEL4WS document our approach can be easily adapted to other (workflow) languages based on XML.

The development and execution of workflows within the Grid is a complex process due to the mechanisms used to describe them and the Web Services they are based upon. The selection of the best resources to use within the Grid is complicated due to its dynamic nature with resources appearing and disappearing without notice and the load on these resources changing dramatically over time. These are issues that the scientist will, in general, not wish to be concerned about. The use of advanced reservations can help to reduce the amount of uncertainty within the Grid. However, in general, the selection of the best service to reserve is a complex process. We adopt the approach of a workflow pipeline presented by McGough et al. [26] to decompose this problem into more manageable steps.

Huang et al. [23] have proposed a just-in-time workflow optimization service in which a workflow calls a proxy service rather than the actual service, which then determines the best resource to use. Our approach improves on this by allowing the end points in a BPEL4WS document to be updated on the fly thus “hiding” the cost of computing the best service in line with execution.

Thus far we have used simple workflow optimization heuristics however we see that our approach can be easily adapted to use more optimal approaches such as stochastic optimization [31] and overall workload optimisation [3].

3. Real-time QoS aware workflow management system

The GRIDCC Workflow Management Service (WfMS) is defined as an end-to-end workflow pipeline. The workflow pipeline is illustrated in Fig. 1. The scientist develops a workflow within the workflow editor, which presents an abstracted view, thus hiding much of the complexity. Rather than presenting a generic workflow editor the scientist is presented with an editor tailored more to their specific needs. Instruments are presented as instrument entities rather than generic Web Services. It is then the task of the workflow editor (software) to generate appropriate workflow language documents based on the scientists design. The user may also have a number of QoS requirements, these should be specified alongside the workflow description and submitted to the next stage. Further discussion of the workflow editor can be found in Section 4.

Once the workflow has been specified the process of selecting the most appropriate set of resources is performed. This is the task of the workflow planner. Within the QoS document the scientist may have specified constraints. These may be of three types:

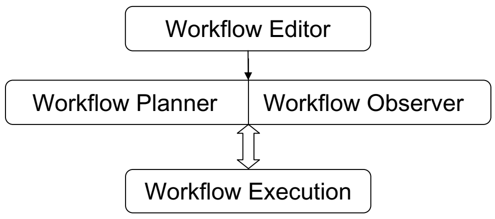


Fig. 1. An end-to-end workflow pipeline.

– *Specified resource*: The scientist has specified the resource that should be used. This may be due to the fact that the scientist has placed a sample onto a specific instrument at a given time. This informs the planner of fixed requirements on the workflow.
– *Specified requirement*: The scientist may not know the best resource to use for a given task even though they know that a reservation will be required. This could be for ensuring enough space is made available to store data. It is the role of the planner to convert this into a specified resource.
– *Unspecified*: The resource is not specified nor is there a specific requirement on this resource. The planner may choose to convert this into a specified resource in order to achieve some overall QoS requirement.

Each QoS requirement may be strict (hard), in which case requirements should be considered mandatory – a hard constraint of ‘this must take less than five minutes’ must be adhered to or the whole workflow rejected. Alternatively the requirement may be soft (loose) in which case a value is given indicating the confidence required – a soft constraint of ‘this should complete in less than five minutes in 80% of cases’ can be accepted as long as the planner believes that it can achieve this in 80% of cases. The workflow observer is designed to deal with cases where the workflow deviates from that planned. The observer monitors the progress of the workflow and, if the tasks deviate from the plan, will trigger the planner to modify the workflow in order to increase the chance of the workflow completing successfully. This is an active area of research. A full breakdown of the workflow planner/observer can be found in Sections 3.1 and 5.

3.1. The grid architecture

Figure 2 shows the overall architecture of the real-time-system used within the GRIDCC project. In the architecture the Virtual Control Room (VCR) is the user interface to the Grid. The workflow editor is integrated within the VCR. The VCR also allows the user to inspect or directly control instruments connected to the Grid. The Workflow Management Service (WfMS) contains the workflow planner and observer along with the workflow engine, in the case of GRIDCC this is the ActiveBPEL [2] engine. The workflow engine may communicate with various Grid services such as Compute Elements (CE), Storage Elements (SE) and, as defined within the GRIDCC project, the Instrument Elements (IE) – a web service abstraction of an instrument along with the Agreement Service (AS) for brokering reservations with other services. The Performance Repository (PR) contains information about previous executions of tasks on the Grid. As both the PR and the AS are exposed as Web Services they can be integrated into workflows as well as being used as part of the WfMS. This allows for both “compile time” and “run time” optimization of the workflow and its tasks.

On receiving a workflow and QoS document the WfMS must decide, based on information from the PR along with an understanding of the workflow, if the submitted request can be achieved within the provided QoS constraints. In order to achieve this the WfMS may choose to manipulate the workflow. This may include making reservations for tasks in the workflow and/or changing the structure of the workflow [26]. The workflow engine is invoked and is responsible for the ordered execution of the workflow tasks, with the observer monitoring progress.

Figure 3 illustrates in more detail the internal structure of the WfMS. First a user must delegate their proxy through the delegation service. This proxy will be stored within the WfMS for later use. A BPEL4WS document along with an associated QoS document is received. This can first be validated before being passed through various stages (outlined further in Section 5) before the BPEL4WS document is submitted to the BPEL4WS engine and the revised QoS (and BPEL4WS) is submitted to the Observer. In order to allow the BPEL4WS engine to communicate with the existing Grid resources, inline SOAP message interceptors (indicated as small circles within Fig. 3) are used to manipulate the Web Services calls, and facades are used to provide simplified composite operations and to sign x509 certificate requests from Grid services with delegated proxies.

The WfMS is split into three main parts those of the BPEL engine, planner and observer. The planner consists of a chain of components used to perform those stages required to take an abstract workflow through to a concrete workflow. The exact chain of components within the planner chain can be adapted and is inspired by the SEDA [24] architecture. Each stage of the chain will consume and produce the same documents of BPEL4WS and QoS. Though in general the contents of these documents will be modified at each stage. The chain will in general contain resolvers to map invocations within the workflow to resources where these are not specified and reservers for calling the AS to make reservations. The return path through the planner consists of a clean up stage which will clear up any reservations and tidy up the resources as necessary.

The WfMS may communicate with many external Grid services that can communicate through Web Services. These include (though are not limited to): AS, Storage Resource Manager (SRM), WMProxy (for communicating with CE through a Workload Manager Service (WMS), CREAM which provides an alternative method for communicating with a WMS, PR, IE and the LBServer which provides access to the logging and bookkeeping service – a store of information about resources on the Grid. It should be noted that to delegate proxies to either CREAM or WMProxy it is necessary to make the calls via a facade as direct communication would lack the appropriate credentials. Though the “deanonymization” of messages (see Section 4), with the assistance of the message interceptors, allows other secure calls to be direct.

4. Workflow editor with QoS support

In recent years, Web Services have established themselves as a popular ‘connection technology’ for implementing a SOA. The interface required for Web Services is described in a WSDL file (Web Service Description Language) [13]. Integration of two or more services into a more complex service can be achieved through ‘Service Orchestration’ or workflows. These are managed by a workflow engine, which orchestrates the interactions between services.

The GRIDCC project is providing a web based workflow editor based on the BPEL4WS [6] 1.1 specification. BPEL4WS is generally regarded as the de-facto standard for composing and orchestrating workflows in the business environment and is now achieving wide adoption within the scientific community. The goal of the BPEL4WS specification is to provide a notation for specifying business process behavior based on Web Services. BPEL4WS

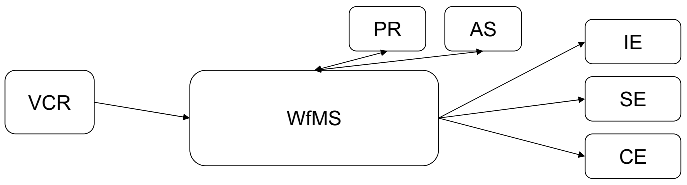


Fig. 2. The GRIDCC Architecture.

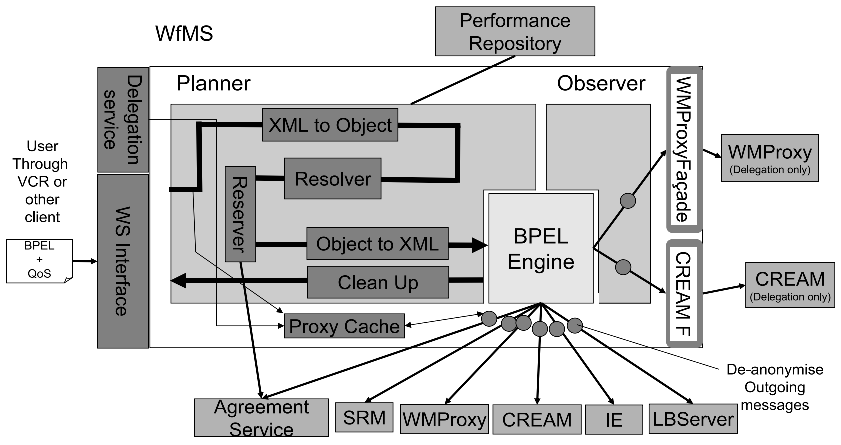


Fig. 3. The architecture of the WfMS.

builds on top of the web service stack and is defined entirely in XML, compatible with other emerging technologies including WS-Reliable Messaging [8], WS-Addressing [9], WS-Coordination [11] and WS-Transactions [17] (BPEL4WS is not limited to only these technologies).

4.1. Business process execution language

The BPEL4WS specification provides a rich vocabulary for the description of business processes supporting two different types of process declaration. An 'Abstract Process' specifies the messages exchanged between participating services without exposing the internal details of the process. An 'Executable Process' extends an abstract process to allow specification of the exact details of how to perform the workflow.

The BPEL4WS 1.1 specification defines various building blocks, known as 'activities'. These activities can be used to specify an executable process' business logic to any level of complexity. Activities can be grouped into: *basic*, such as procedural steps (e.g. invoke, receive and assign), *structured*, which control the flow of the workflow (sequence, switch and while loop) and *special* such as terminate, validate and fault handlers. Within the BPEL4WS workflow process itself, different activities pass information among themselves using global data variables.

Although the BPEL4WS specification is tailored more to the requirements of business processes, there are properties which make BPEL4WS useful in the scientific environment. These properties include Modular Design, Exception and Compensation Handling. Furthermore, scientific services are often subject to change, especially with regard to the data types and service-endpoint locations. Data flexibility is supported using 'generic' or 'loose' web service data types. Finally, the BPEL4WS specification allows for workflow adaptivity, which is mainly facilitated by the 'empty' activity providing a placeholder for future extensions.

However, the BPEL4WS specification does not provide any direct mechanism to support the management and monitoring of a workflow nor does it address web service security issues, such as passing credentials through a BPEL4WS engine. The WfMS uses a "deanonymization" technique in which users identity is first delegated to the WfMS where it can be stored. When the user successfully communicates with the WfMS in the future the outgoing message, from the BPEL4WS engine, is intercepted to attach user credentials and perform a secure call to an external service.

Generating a workflow document is therefore a process of bringing activities together to achieve the end-

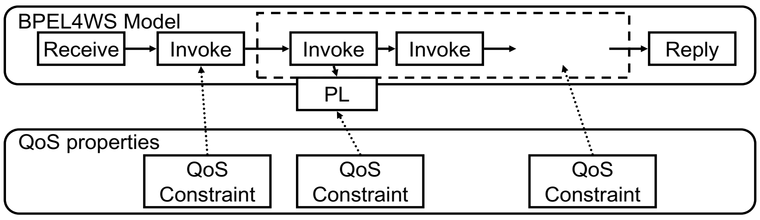


Fig. 4. Connecting QoS Document and BPEL4WS Model.

goal; similar to a low-level programming language this is usually a complex and tedious process. A number of BPEL4WS editors exist, such as the Active BPEL Editor [19] and the Oracle BPEL4WS Process Manager [30]. However, these are developed for computer scientists who wish to develop workflows and have an extensive knowledge of BPEL, and WSDL.

4.2. Aim of GRIDCC editor

The flexibility and richness of features in the BPEL4WS specification brings with it unavoidable complexities. These complexities hinder its use by scientist and researchers in an academic domain. To execute a workflow, the user needs to be able to submit the workflow to a BPEL4WS engine along with documents describing the Web Services instances to use with the workflow, the web service interface (WSDL) for this workflow and other deployment documents. The user then needs to be able to trigger this workflow by using its interface (WSDL). To develop your own workflow also requires a full knowledge of how to compose and manipulate these documents.

Considering the user requirements, we can identify two categories: (1) users executing existing workflow for complex scientific procedures in which they may not be experts; and (2) expert users engineering new or existing processes as workflows. For the first type of users, web applications are appropriate since executing existing workflows means uploading the workflow; supporting parameters and constraints to the server for use by the workflow engine. The second type of user needs rich client software with advanced functionality in order to develop new workflows. This includes comfort features, such as a polished user interface, drag-and-drop, a rich set of keyboard shortcuts, and provision to save/load whole or partial workflows. These whole or partial workflows can then be integrated, as building blocks, to form part of more complicated and sophisticated workflows. The purpose of the GRIDCC editor is to provide a hybrid solution and address some of the limitations of the BPEL4WS specification which are normally ignored by existing open source and commercial editors. The GRIDCC editor differs from existing editors in the following ways:

Portal Based Editor: All open source and commercial workflow editors require installation and configuration of the editor before use. Installation of a workflow editor means access to local file system (normally) as admin, which may not be available. This JSR 168 [1] compliant workflow editor will provide an editing tool on demand without the need to install. Users can edit and save the workflow on the server, this can be performed either from their own web browser or any other (subject to security constraints) at any time in the future. This provides a powerful framework for the sharing of workflows. The use of a JSR 168 compliant portal and portlet interface allows for a mixing of the presentation layer of the editor (running within a web browser) with the back-end workflow editor logic which is implemented in Java and runs on the server side. This allows for a simple Graphical User Interface (GUI) written for a web browser to delegate all of the complicated logic, for such things as validating a workflow, to be handled by the server side. This logic is often referred to as the business logic. Browser based clients have an inherent advantage as they do not need to be upgraded on the client side and provide a pervasive access mechanism. Further, large, complex workflows can be designed by experts then stored and maintained centrally, meaning users need only access and deploy them via the portal.

Drag and Drop: The GRIDCC editor provides a drag and drop facility which allows the user to drag various BPEL4WS activities, Web Services or operations from the web service registry, Quality of Service (QoS) constraints from the QoS panel and variables from the XML Schema registry into the workflow designer pane. Dragging of different components on the designer pane either updates the BPEL4WS script or creates the corresponding QoS elements. The workflow editor is based on ActionScript 3.0 [22] and MXML [15]; the core components of Macromedia Flash. ActionScript is a scripting language supporting various features, such as

```
<?xml,version="1.0",encoding="UTF-8"?> <QoSRequirements>
    <QoSConstraint>
        <XPathReference>process</XPathReference>
        <Resources>
            <CPUSpeed>2000000</CPUSpeed>
        </Resources>
        <MaxDurationTime>100</MaxDurationTime>
        <Reliability>100</Reliability>
    </QoSConstraint>
</QoSRequirements>
```

Fig. 5. A global requirement.

```
<?xml\,version="1.0"\,encoding="UTF-8"?> <QoSRequirements>
    <QoSConstraint>
        <XPathReference>
          /process/sequence[1]/invoke[1]
        </XPathReference>
        ...
    </QoSConstraint>
</QoSRequirements>
```

Fig. 6. A single invoke activity requirement.

drag and drop, within the web browser. These features are normally only available through the use of a desktop (installed) application. MXML is the XML-based markup language for the presentation layer.

Hiding Complexities: A generic BPEL4WS workflow editor demands advanced skill from workflow designers; i.e. a thorough understanding of Web Services architecture and different types and styles of Web Services, expertise in managing XML files from various namespaces, experience of an XML query specification language such as XPath or XQuery; and familiarity with the BPEL4WS specification. Web Services and BPEL4WS specification have dependencies on other related specification e.g. WS-Addressing; which further complicates the design process. The GRIDCC editor hides these complexities from the scientist and researchers by automating different tasks in following way:

1. A workflow process requires global variables for its proper functioning. Whenever a new process is created a global corresponding `<variables>` element is created at a specific location as required by the BPEL4WS specification. The `<variables>` element is a registry of different XML specific data types and variables.
2. A workflow orchestrates various Web Services wrapped in `<partnerLinkType>`, `<partnerLinks>` and `<partnerLink>` elements. Each `<partnerLinkType>` element wraps a `<portType>` of the web service, which itself wraps various operations. The GRIDCC editor creates the `<partnerLinkType>`, `<partnerLinks>` and `<partnerLink>` when any web service is added in the web service registry.
3. Operations on the partner Web Services are called through the `<invoke>` activity. The `<invoke>` activity specifies the `<partner Link>`, `<portType>`, `<operation>`, `<input>` and `<output>` elements defined in WSDL or BPEL4WS script. An operation of the web service can be dropped directly on a workflow process from a web service registry and the editor itself creates the corresponding `<invoke>` activity by relating various required elements.
4. An `<invoke>` element must specify `<input>` and `<output>` variables to store the values of outgoing and incoming messages. If these variables do not exist then the GRIDCC editor creates the `<variable>` required for successful invocation of the operation in the `<variables>` element (item 1).
5. The GRIDCC editor also adds "exception handling" activities with `<empty>` activities as a template for flexibility and extensibility. The

```
<?xml\,version="1.0"\,encoding="UTF-8"?> <QoSRequirements>
    <QoSConstraint>
        <XPathReference>
          /process/sequence[1]/invoke[1]
        </XPathReference>
        <XPathReference>
          /process/sequence[2]/invoke[2]
        </XPathReference>
        <MaxDurationTime>100</MaxDurationTime>
    </QoSConstraint>
</QoSRequirements>
```

Fig. 7. Multiple invoke activity requirements.

designer of the workflow can replace these `<empty>` activities with actual business logic.

Ease of use: The GRIDCC editor is easy to use due to its built in error handling. The workflow editor validates the actions of the designer and makes sure different workflow activities are arranged according to predefined specifications. Different activities are arranged in the logical groups and different activities are enabled only when they can be used. The BPEL4WS Designer maps structured activities of the BPEL4WS language e.g. Sequence, Process, Flow as Graphical User Interface containers such as a Panel. This mapping was a natural choice as structured activities of BPEL4WS can wrap simple activities like invoke, receive or reply and similarly GUI containers can contain simple widgets like buttons, text boxes and labels. The main advantage of this mapping is the enhanced user interface such as when the container is dragged or re-positioned visually then all of its inner components are re-positioned relatively, which can be likened to re-arranging of a sequence activity along with all its sub activities within the BPEL4WS process.

Web service registry: The Editor provides a very basic Web Services registry to arrange Web Services for later use. When a user adds a new web service to the registry the editor creates corresponding `<partnerLinkTypes>`, `<partnerLinks>` and `<partnerLink>` elements for later use in the `<invoke>`, `<reply>` or `<receive>` activities. The Web Services registry hides the inner details and complexities of the BPEL4WS script and WSDL extensions (required by the BPEL4WS specification) from the user and the designer can concentrate more on the business logic rather than wrapping the partner services in various elements. The web service registry can also be used as a pool of semantically equivalent but geographically dispersed Web Services.

XML Schema Registry: When any web service is added into the Web Services registry, the editor parses the web service and extracts the data types declared in the `<wsdl:type>` and `<wsdl:message>` elements and build the registry of various data types. XML data in the registry is namespace aware, however, the user doesn't need to address the namespace issues and conflicts. At design time the user only drags the required data type and the editor creates the respective variable with correct structure in the BPEL4WS script.

QoS Constraint: The GRIDCC editor provides a pallet grouping of various QoS constraints. The BPEL4WS specification does not define the quality issues related to overall workflow nor individual Web Services. QoS constraints can be coupled with different BPEL4WS activities particularly `<invoke>` activities or linked to a particular service through its `<partnerLink>`. QoS constraints for the workflow are specified in the separate file rather than embedding them within the BPEL4WS script.

Workflow Monitoring: Some popular workflow engines describe the state of a workflow in XML, but not all. If a workflow state is available in XML format then it can be exposed and can be queried by the client application. It is also possible to retrieve a workflow snapshot at run time in XML; which can be transformed using XSLT into user acceptable formats. However, it must be stated that extracting an XML snapshot of the current state of the executing workflow requires some understanding of the underlying workflow engine. Consequently, it is easier to create a workflow monitoring web service which is independent of the underlying workflow engine. This involves monitoring nodes being introduced within the workflow script which 'call back' the monitoring service, allowing the service to keep a record of how far the workflow has progressed. This has the advantage that the monitoring service is unaffected by the underlying workflow en-

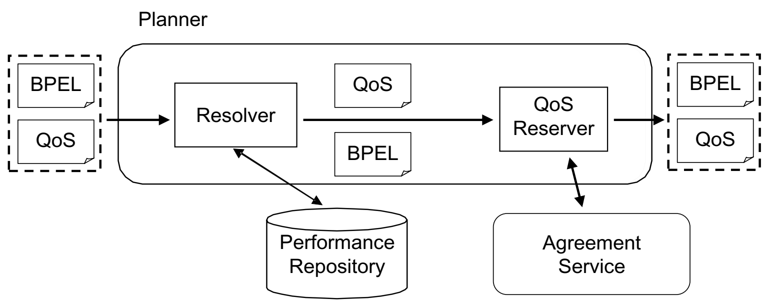


Fig. 8. QoS Components.

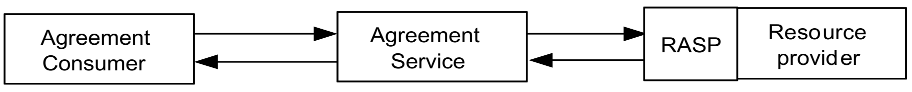


Fig. 9. Agreement Service interactions.

gine implementation and allows the user to select which elements within the workflow they wish to monitor, ultimately giving the user a greater level of control. Development of a custom monitoring service can be especially important when the status of multiple workflow activities are crucial in deciding the execution path of the workflow.

5. QoS aware workflow optimization

Quality of Service (QoS) is a broad term that is used in this context to denote the level of performance and service that a given client will experience at a specific point in time, when invoking a given specific operation on a web service instance.

QoS support is paramount given the inherent shared nature of the Grid. This is compounded through the limited capability of certain high demand services. Furthermore, QoS is essential to deal with the requirements for real time interactions, such as guaranteeing the streaming and storage of data from an instrument. In this environment to satisfy users requirements will often lead to resource starvation or a situation no better than first come first served processing due to users making exclusive reservations for all stages of their workflow. This effect can be alleviated through the client specifying loose (soft) and strict (hard) QoS requirements as appropriate.

The provisioning of loose guarantees consists of the capability of clients, to select service instances that can potentially provide a best-effort QoS profile meeting the client's requirements. This is based on previous measurements of that service. Loose guarantees are delivered on a best-effort basis, and for this reason, they generally do not require the prior establishment of a Service Level Agreement (SLA).

Strict QoS requires the certainty of the delivery of the prescribed level of service, which needs to be agreed upon through signaling and negotiation processes involving both the client and the service provider. For example, the reservation of a given portion of a resource such as RAM, disk space, network bandwidth or instrument. The reservation service provider is responsible for keeping information about resource availability over time, for ensuring that no resource is overcommitted and for supporting resource-specific locking mechanisms.

QoS provisioning of our work relies on both strict and loose QoS guarantees. These QoS requirements can be made either by a user through the client interface as discussed in Section 4, or as part of the workflow manipulation process. While hard QoS requires the making of reservations on the resources to be used, soft QoS requires the user (or planner acting on the users behalf) to model the execution of the services that may be used to satisfy the users requirements to determine if these resources are appropriate. These models may vary from the simple, when only a single service is required to the extremely complex when several services are required as part of a workflow. In such cases it may be required that a reservation is used even to satisfy loose QoS constraints.

A user submits a BPEL4WS document, which is likely to contain a number of service invocations – referred to as tasks. A separate document is used to describe the QoS requirements placed on the BPEL4WS document. Although a number of languages exist for

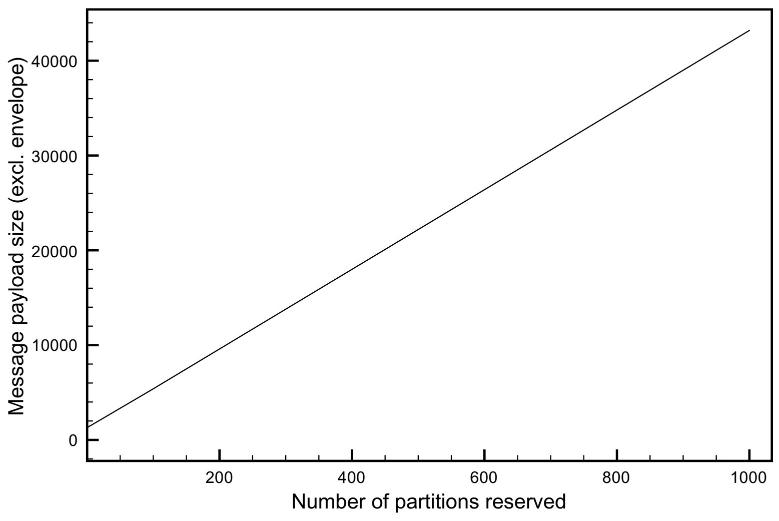


Fig. 10. Instrument reservation invocation messages size, in Bytes.

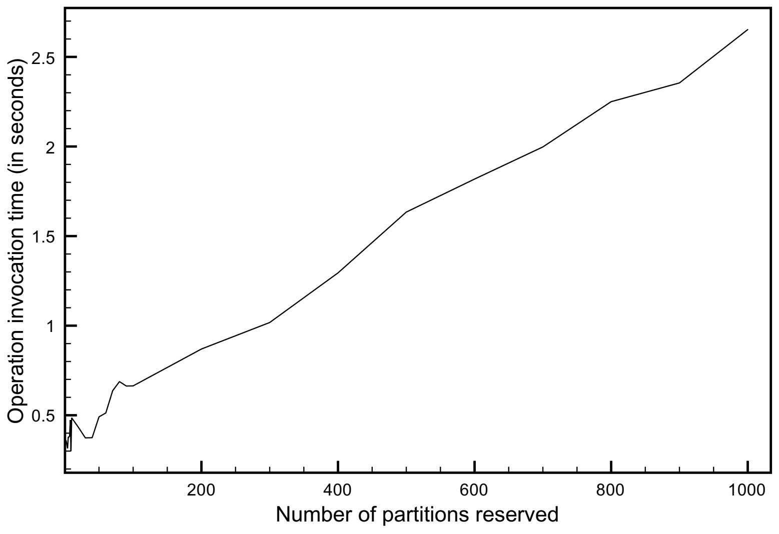


Fig. 11. Instrument reservation completion time (linear scale).

describing QoS requirements, none exist which is suitable for describing QoS requirements with a workflow description. A common practice emerging within the XML community is to have one document per use, and for these documents to cross reference each other, for example a BPEL4WS document describes the abstract workflow, another document describes the deployment details and a third defines the skeleton of the workflow's WSDL. In this spirit of the 'one-document-one-use' paradigm, the QoS requirements are themselves defined within their own document, which references the associated workflow, in our case a BPEL4WS document. This allows us to use existing BPEL4WS tooling to develop and execute workflows, and further it means future development of QoS support for other workflow languages would be more straightforward.

As such a simple QoS language has been defined which uses XPath [14] references into the BPEL4WS document to tag activities, both basic and structured activities as shown in Fig. 4. These XPath tags are then annotated with the QoS requirement such as time to execute, disc space required or memory required. Alternatively services that are used may be tagged by their `<partnerLink>`. This has the advantage that these can be processed without the need to decompose the BPEL4WS document. However, if the service is used

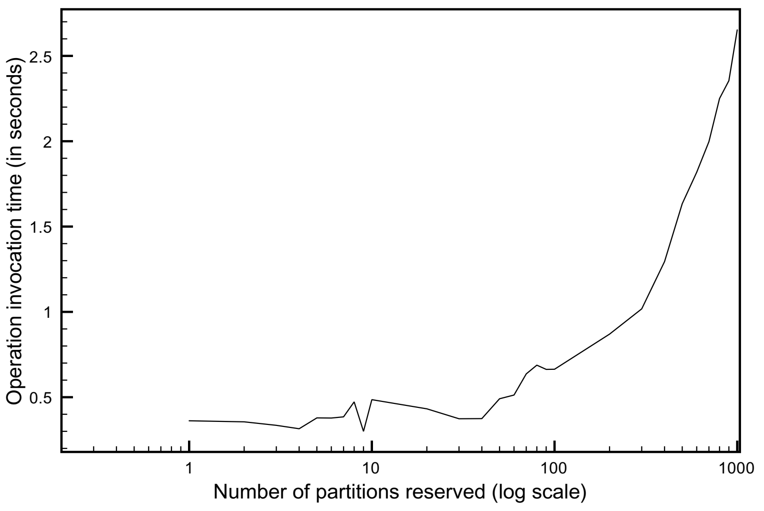


Fig. 12. Instrument reservation completion time (logarithmic scale).

multiple times within the document it is not possible to distinguish between these. Thus all calls will share the same QoS requirements.

When using the XPath approach QoS requirements fall into three categories according to the range of activities within the BPEL4WS document it refers to: global, single invoke and multiple invoke.

A **global requirement** is a QoS element that specifies single global QoS requirements. A simple example is given in Fig. 5 (for simplicity, all the unnecessary technical details are omitted).

In the above example, there is a single *XPathReference* pointing to the entire process of the BPEL4WS document. Thus everything within this process must match these QoS requirements. In this case the overall time should be less than 100 seconds, all CPU's should be 2Ghz and all resources should be fully reliable.

Single invoke activity requirement is a QoS element that specifies requirements on a particular BPEL4WS activity. Only that activity needs to satisfy the QoS requirement specified as shown in Fig. 6.

In this case there is a single *XPathReference* pointing to a single invoke element of the BPEL4WS document. Thus everything within this invoke must match these QoS requirements.

Multiple invoke activities requirement is a QoS element that specifies requirements over a set of activities – i.e. all must satisfy (jointly) the QoS requirements. Within a single QoS constraint, several *XPathReferences* pointing to different activities in a BPEL4WS document are defined. See Fig. 7 where the time for both invoke activities must be less than 100 seconds.

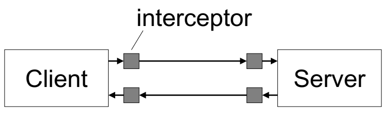


Fig. 13. Inline SOAP message interceptors.

It should be noted that multiple QoS elements may exist within the same QoS document and that these elements may have overlapping effects on the BPEL4WS document. For example there may be a requirement for the entire workflow to complete within an hour while several of the individual tasks may need to be completed in less than five minutes each.

5.1. QoS components

The Planner is responsible for ensuring QoS adherence. This is achieved through the use of a number of stages – see Fig. 8. Namely *constraint resolver, basic resolver, performance repository and QoS reserver*. These components are chained together with the QoS and BPEL4WS documents being passed between them. We explain each of these steps in more detail:

Performance Repository is central to being able to perform any quality of service decisions. Information is held within the PR as to how different entities within the Grid behave. This may be information about how reliable a service is or how long it is expected to take to execute. Information is retrieved from the PR in order to determine if a workflow can be executed within the requested QoS requirements and if so which are the most appropriate resources to select.

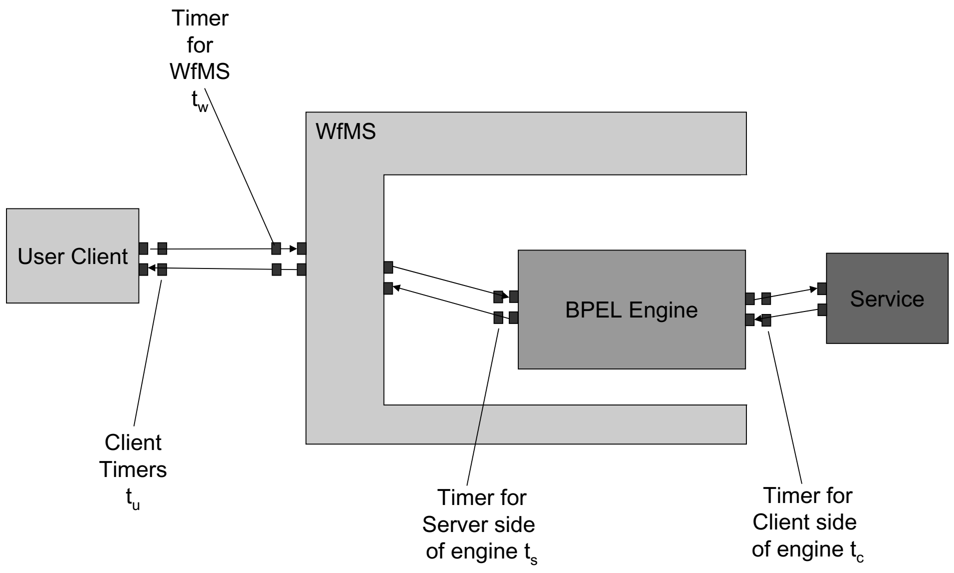


Fig. 14. Locations of timings within architecture.

Resolver Services. Here a chain of services are used to modify an abstract workflow in which potentially not all endpoints are defined into a concrete workflow in which the endpoints are defined. The selection of resources to use is an area of active research and we seek to develop alternative approaches for this stage, allowing for comparison. Here we discuss some of the approaches that we have been evaluating. The most simple approach is to use a round-robin service which selects the next available resource until one is found matching the QoS and other requirements. A constraints resolver based on a constraints equation method can also be used. The workflow along with the QoS requirements is converted into a set of constraint equations in the form of linear equations which can be solved by using Mixed Integer Linear Programming [31]. Information from the Performance Repository is used to help solve these constraint equations. Another approach is to use a priori information to pre-allocate resources to a specific set of (common) task thus making the process of resolving a mapping exercise [4]. It should be noted that a QoS element without a named resource is then changed into an element requesting a named resource. This can then be passed onto the QoS Reserver for making the actual reservation if required.

QoS Reserver inspects incoming QoS documents looking for requests for making reservations with known resources. The Agreement Service is then contacted in order to make these reservations. The QoS document is then updated to indicate that the reservation has been made and records the unique token used to access the reservation. All requests for reservations are processed here. In the current implementation if reservations can’t be satisfied then the whole document will be thrown back to the user to select new reservation times. Once we have components capable of selecting timings for reservations internally the workflow and QoS will be returned to this component.

5.2. Observer

The Observer receives a completed copy of the QoS document, and the BPEL4WS document, at the same time that the workflow engine receives the BPEL4WS document. This QoS document contains timing information as to how the workflow is expected to execute. The Observer is then able to monitor the progress of the executing workflow, through status calls to the workflow engine, in order to ensure that the workflow executes as desired. If the workflow deviates from the expected plan, in either direction, the observer is able to invoke the Planner to re-compute the workflow in order to achieve the desired QoS requirements. As each workflow engine implements status calls in a different way not only is this approach difficult to implement but also unclear as to the performance hit that will be incurred. The ActiveBPEL editor seems to just expose its internal state and as such calls seem to have little overhead. However, this call should only be made as needed and is an area of open research.

We currently support the dynamic changing of endpoints within the BPEL4WS documents. All end-

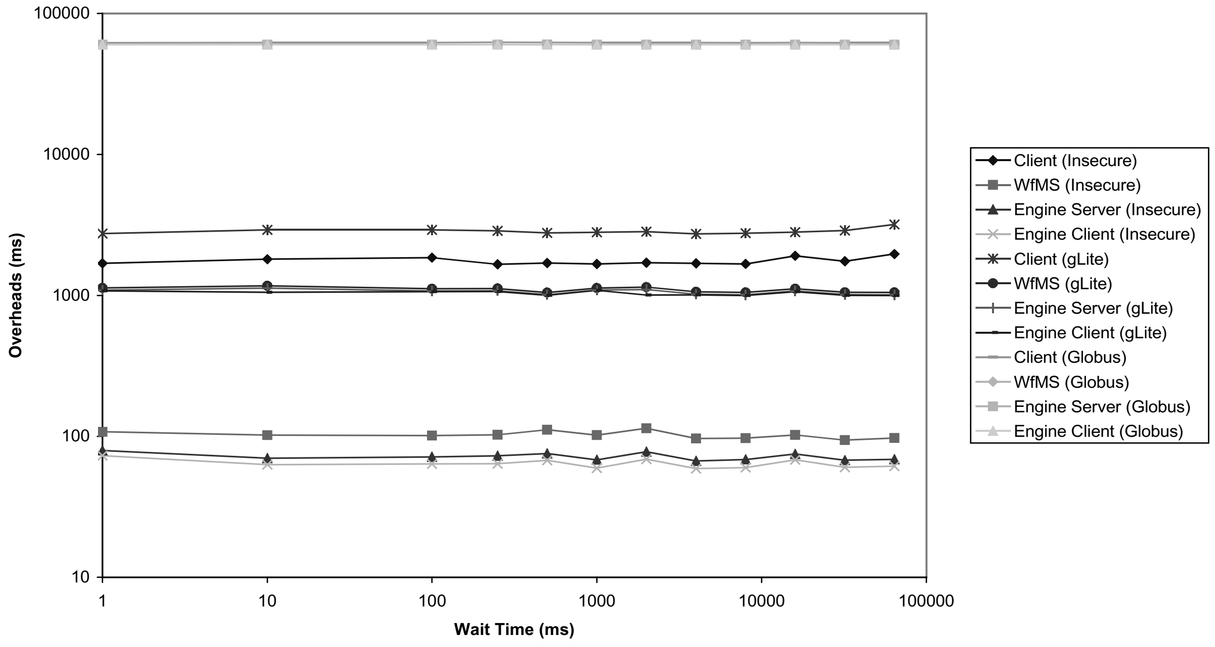


Fig. 15. Overheads vs wait time.

points within the BPEL4WS document are defined by variables within the workflow, which are assignable through calls to the workflow at initialization time, via a call to a WSDL operation. For additional control, we are investigating approaches to dynamically alter the endpoints during workflow execution. One such approach would be to provide interrupt operations on the workflow, which would update these endpoints while the workflow itself is running, and would allow for endpoints to be selected for optimization without being forced to compute them all at the time the workflow is first triggered.

6. Performance repository

The Performance Repository (PR) is the central source of information about how services within the Grid behave. This information includes: reliability, availability, accessability and expected sojourn time. The PR provides two main functions, those of gathering and analyzing information about other services and that of providing responses to queries about these services.

The PR is implemented as a database and contains two main types of data: modeling and collected. Model information provides mathematically modeled descriptions on how a service operates. This is produced through off-line analysis of the service. It should be noted that the architecture of the PR is agnostic to the modeling approach used. Models are held within the PR as separate objects with the requirement that they can respond to the same set of queries that are supported by other data sources within the PR such as historically collected data. Alternatively a service may present information at regular intervals to the PR. This collected information can be data-mined in order to obtain estimates for the service. We seek to extend the PR such that it may derive models from collected information as well as use seasonality to help predict more accurately future state. The intention is to provide a service comparable with the Network Weather Service [32] for services within the Grid. There are two customers for the PR's information, the individual user and the WfMS planner.

Requests for information from the PR may not be handled by the PR in isolation. For example a query of "What 3 Ghz Pentium computers are more than 80% reliable?" would first make use of the standard resource selection service to determine which resources are 3 Ghz Pentium processors and then check this candidate set against their reliability. However, this only holds for static information. Queries about current workload and reservations held is not within the remit of the PR.

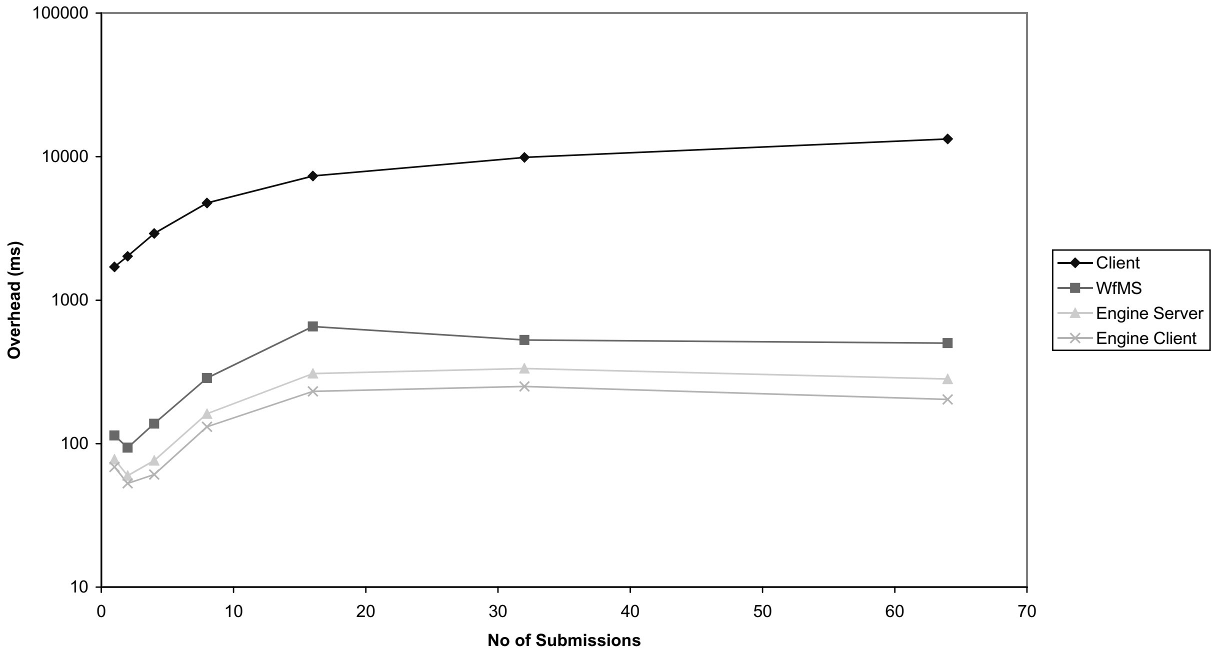


Fig. 16. Overheads vs Number of submissions.

7. Reservation services

The GRIDCC *Agreement Service* (AS) is the service implementing resource reservation for those resources which support this functionality, i.e. storage and instruments. The name stems from the fact that the AS is using ideas and, to a certain extent, data types and interfaces from a WS-Agreement [7] specification draft. As such, the AS establishes agreements for resource reservation: It stands in the middle between agreement consumers and *Reservation and Allocation Service Providers* (RASPs), as shown in Fig. 9. The agreement consumer is the entity requesting the reservation, and may or may not be the same entity with the reservation consumer. The RASP is a gateway to the resource, for reservation purposes. In this ecosystem of resource consumers and brokers, the AS is the entity accepting *agreement offers* which comply to specific templates, parses and interprets the terms, and eventually contacts the RASP to request the reservation on behalf of the agreement consumer.

Essentially, the AS is an intermediary which implements a control protocol for establishing digital contracts concerning reservations of storage space and instrument elements. By "control" we refer to the fact that the domain-specific reservation information (for instance start/end time) is not visible to the AS and is not a part of the protocol (language) that the AS understands, but rather carried in the message payload – the *Service Description Terms* (SDTs), in AS terminology. External processors are responsible for interpreting the SDTs and converting them to a language that the RASPs understand.

As there may be multiple Agreement Services establishing reservation contracts for different administrative domains, and a single domain may be served by more than one Agreement Services, resource state monitoring and resource locking cannot be realistically performed by the AS. Thus, the AS only holds the role of that component which proxies requests and performs necessary translations, so that agreement consumers need only speak one language for resource reservations. It is up to the resource itself to honor the contract and ensure the availability of what has been reserved at the specified time and for the specified duration.

7.1. Advance reservations for overall performance enhancement

Advance reservations can affect the overall performance of a workflow in two different ways, depending on whether it is computing facilities, network resources, storage space, or instruments being reserved. Here, we do not discuss the case of reserving data in digital libraries or other repositories.

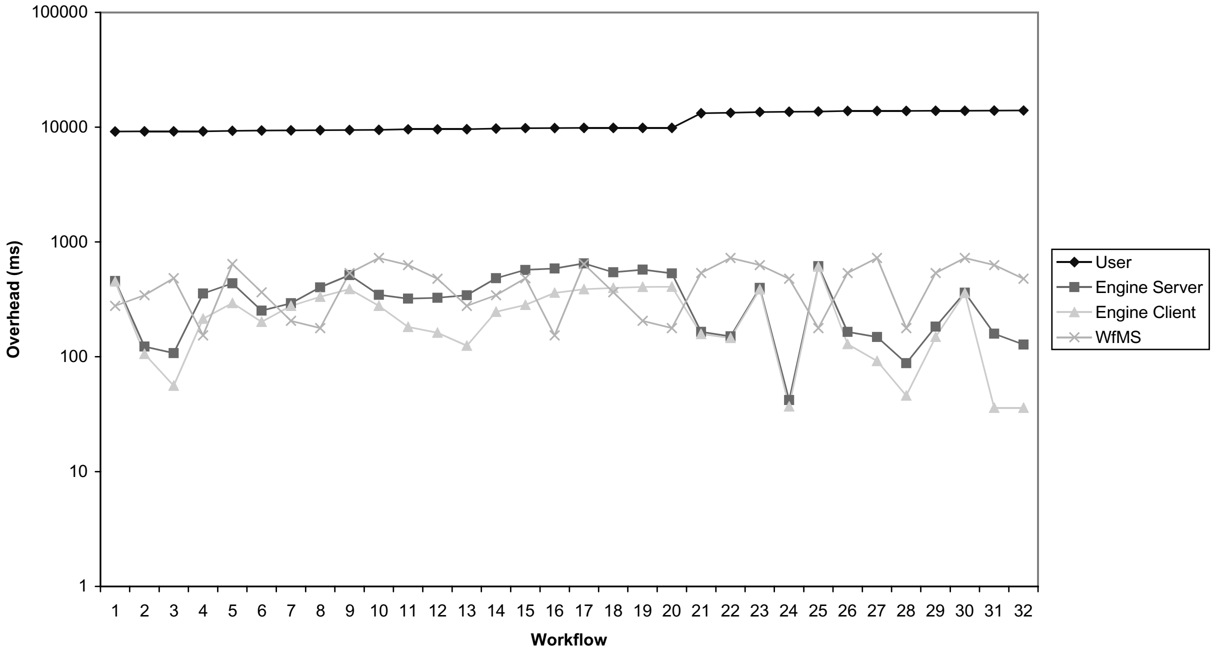


Fig. 17. Overheads for a multiple submission run.

The performance change is directly related to the shared (or not, thereof) nature of resources to be reserved. If a resource can be used in a shared manner, as is the case for instance with computing facilities and network links, advance reservation can boost or lower workflow execution times when demanding processing or transfer of information is involved. In the former case, for instance, the user locks a slice of the computing facilities for themselves, so the processing tasks submitted will finish as soon as they are expected to. If, however, there is no reservation in place, the user will either have to compromise for slower performance, to wait until the resources are available to them, or to re-route the task thus introducing another discovery-matchmaking-submission cycle. Under situation of low utilization it may even be possible for the task to complete sooner without reservation as a greater proportion of the resource than would have been reserved is used. This is a risky approach and is in no way a reliable mechanism. The case is similar for network resources.

When it comes to non-sharable resources, the workflow performance may suffer from the need to wait until the resources are made available from others. For instance, if a workflow must use a specific instrument to retrieve data for the region it is installed in, there is no option other than to wait for the instrument to become available for use. Should this requirement be known well in advance, the reservation of the resource would allow its immediate use at a pre-specified time, thus removing any queuing delays.

In both cases, the usefulness of reserving resources in advance is directly related to the specific application (workflow) taken into consideration. An application for the batch-processing of historic data can typically accept delays introduced from shared use of resources. For real-time applications which demand immediate and efficient access to resources, the ability to reserve resources in advance may be of critical importance, as they may have to block and suffer significant delays should they not have reserved them when needed.

8. Evaluation and performance

In this section we provide an evaluation of the real-time services used within the GRIDCC project. Due to the nature of this work being based on bringing workflow support to Grid middleware and for the support of instruments within the Grid it is difficult to give a quantitative analysis of this work. It is difficult to evaluate the benefit of providing workflow support within the Grid. Instead we focus here on the quantitative features of the system such as the effect on performance of using reservations, the chance of a workflow being accepted, the influence of the size of the PR on the planning time and the overheads incurred from using this approach.

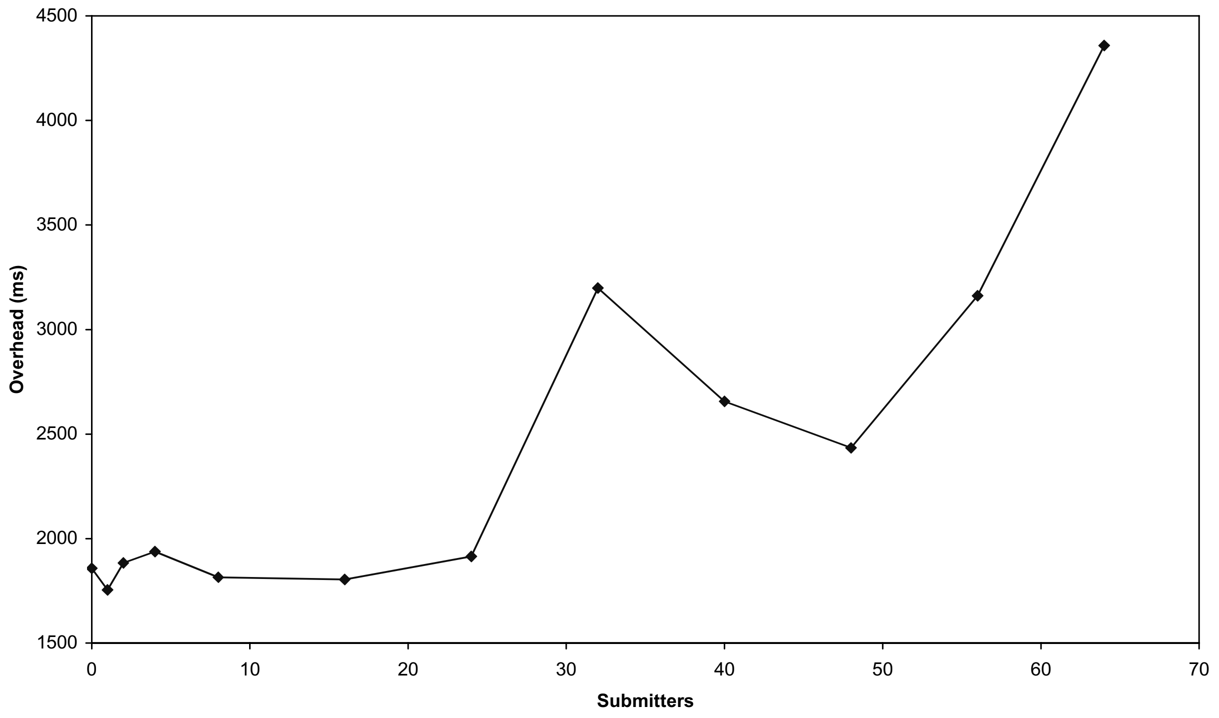


Fig. 18. Overheads vs multiple clients.

8.1. Reservation jobs

In the current implementation, reserving storage is always dependent on the resource being used, with regard to the completion time of the operation. The only thing that changes from the AS perspective is the amount of space to be reserved, which is passed on to the storage resource. As such, efficiency of the AS for storage reservations is always the same. It is interesting, however, to see how the AS scales with instrument reservations.

Instruments and instrument collections are typically virtualized using the same interface. As such, an Instrument Element may control a single instrument or multiple instruments of various types. The internal organization of instrument collections is represented by a tree structure. Each sub-tree of such a tree is considered to be a *partition* of the instrument, which can be reserved. As such, reserving the partition corresponding to the top node (root) is equivalent to reserving the whole instrument. Reservation of a partition involves inserting a relevant entry in the *instrument reservations database*; Then when a user contacts the instrument and tries to use it by setting its properties, they are not authorized if a reservation entry for this instrument partition exists in its reservations database and is not owned by the user.

In the case of instrument reservations, the message payload increases linearly as the number of partitions to be reserved increases (Fig. 10). We conducted a number of experiments over a local network, so that networking delays do not affect our measurements. The actors involved in the experiments were a client to the AS, running on the same workstation as the AS. The remote instrument is simulated by a service running on a low-end computer on the same Local Area Network, connected at 100 Mbps. We ran a series of 10 experiments with varying number of instrument partitions to be reserved and came up with their averages. As shown in Figs 11 and 12, the AS return time scales linearly as the message payload size increases.

This is not the case when reserving instruments by quoting their type or functional and non-functional characteristics. In this case, the message size changes insignificantly, and it is the RASP and resource implementation that provides the biggest effect on performance. Similarly, when reserving CPUs, the AS performance depends largely on whether reservations take place by naming the exact resources to be reserved, or by grouping them according to their properties and reserving them in this way. Computing facilities reserva-

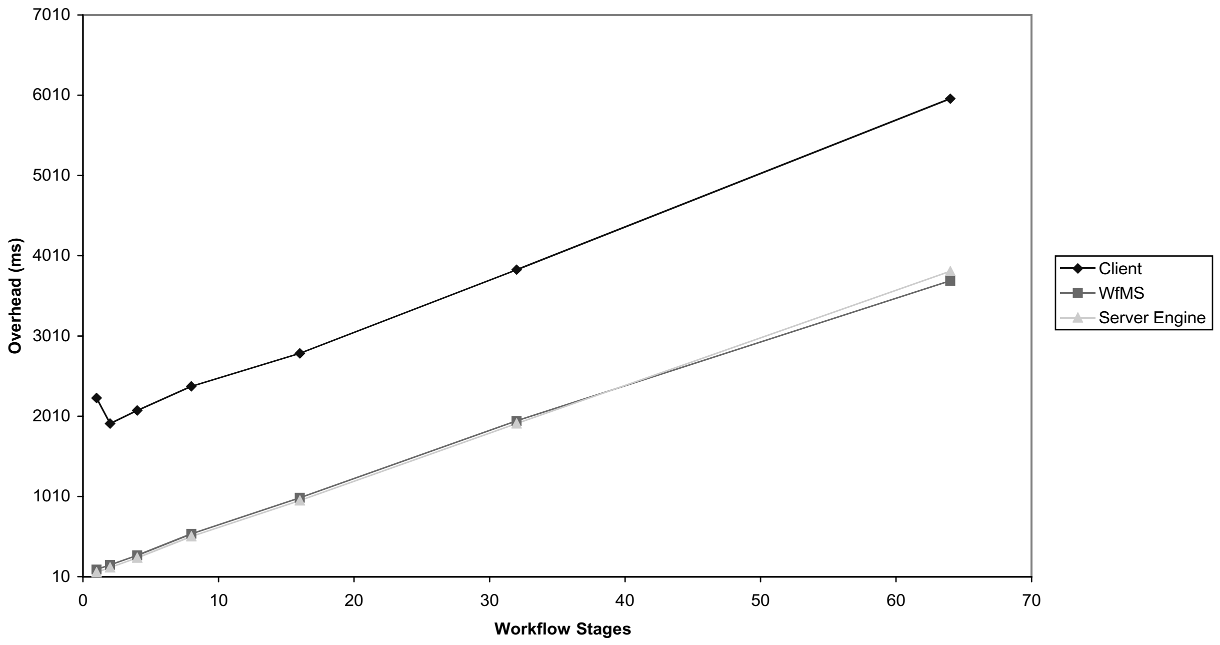


Fig. 19. Overheads vs number of workflow stages.

tion has not been implemented in our work yet, and is currently work in progress.

8.2. *WfMS overheads*

Here we present the results of experiments to evaluate the real-time performance of the WfMS (release WfMS_D4_3_RC1). The experimental set-up comprised of different computational resources which belong to a medium level network at Imperial College London distributed between the departments of Computing and Physics. We measure response times at different points within our architecture and establish the sampling distributions of the mean.

Here we concern ourselves with the overheads incurred from using the WfMS as a workflow enactment service. This can be from the WfMS itself or the use of the BPEL engine. In all cases communication between services and the serialization and de-serialization of messages along with the processing of these messages are not individually distinguished and form part of the overheads. In order to test these overheads we need a mechanism for timing parts of our architecture. Due to the distributed nature of the workflow service it is not possible to directly time calls between different resources at the level of accuracy required – on the order of milliseconds. This is due to the fact that clocks between computers are often miss-aligned and are rarely accurate to this extent. Instead our approach is to perform comparative timings between events on the same resource. This allows for round-trip timings to be made though not uni-directional timings. In order to add timings to services which are not part of our code base and to make timings of our code without the need to explicitly write the timings into the code we use the inline SOAP message interceptors approach. This is illustrated in Fig. 13 below.

We use this approach in a number of places as depicted in Fig. 14 with Table 1 describing the timings used. Note that these interceptors record timings for all messages that pass through them. Thus the output from the WfMS was omitted as this would have also recorded all other calls and made it virtually impossible to determine which were calls to the BPEL engine. Timings of the end service were omitted as these closely matched the execution time of the service. All results are generated from averages of ten iterations. The specification of the resources and their locations are given in Table 2. All resources were connected to 100 MB networks. It should be noted, however, that there is no specialized link between the departments and all experiments were done during normal load. For some of the tests multiple client boxes were used. These were identical resources from an open cluster room. All calls between the Client service and the WfMS were performed securely using the gLite https tooling. The communication between

Table 1
Meaning of timings in Architecture

Time	Name	Description
t_u	User time	Timing observed by the user for invoking a workflow.
t_w	WfMS overall time	Time observed by the WfMS for invoking a workflow.
t_s	BPEL Server	Time observed by the BPEL engine for an incoming workflow.
t_c	BPEL Client	Time observed by the BPEL engine for each call out to external services.

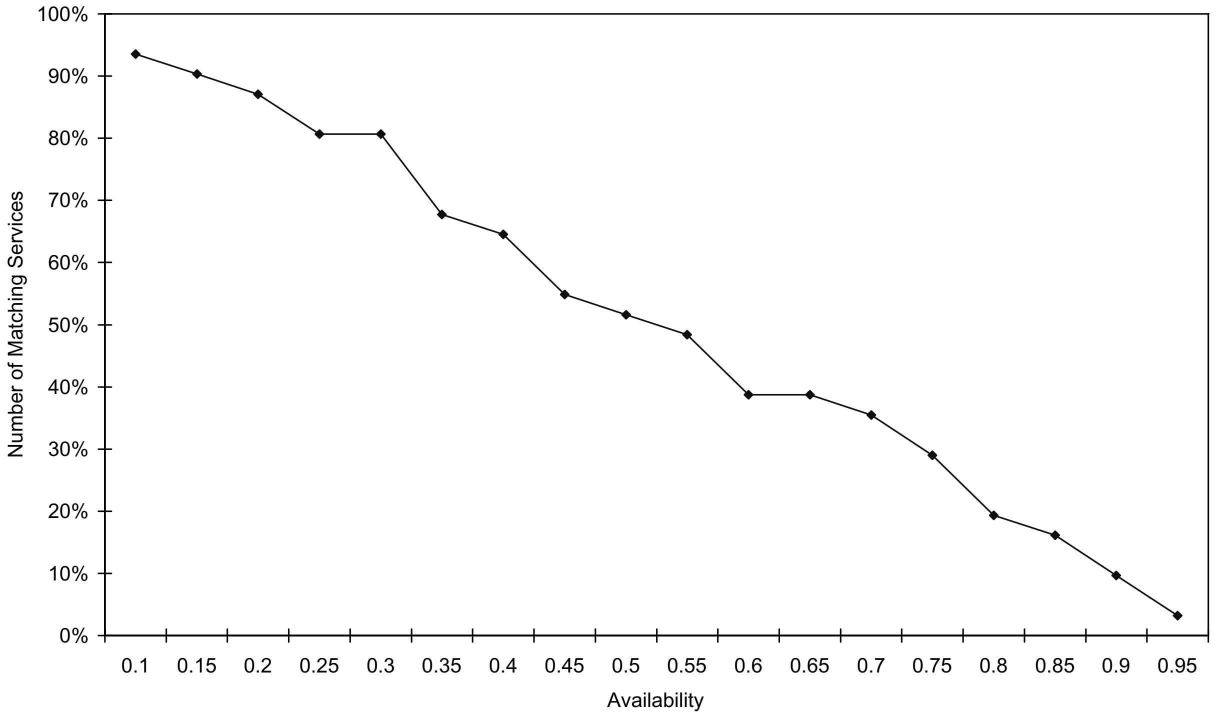


Fig. 20. Matching services vs availability.

the WfMS and BPEL engine was unencrypted as both services were running on the same box. Three forms of security were used between the BPEL engine and the service. Those of unencrypted, gLite (https) and Globus (https).

As the main emphasis of this section is to evaluate the WfMS and BPEL engine we have constrained the workflows to ones that can be controlled precisely – the only services which is used is a wait service, which waits for a pre-determined number of milliseconds – thus allowing for easy computation of overheads. We used two workflows; the first workflow consists of just one call to the wait service (request $\rightarrow$ wait $\rightarrow$ respond) and the second consists of repeated calls to the wait service (request $\rightarrow$ wait n times $\rightarrow$ respond). The first workflow allows for the testing of the overheads incurred when executing a workflow. While the second indicates the additional overheads incurred from the number of components within your workflow.

Overheads vs Wait time. In this experiment a single wait workflow was executed with varying wait times being used and was conducted with all three security mechanisms (unencrypted, gLite, Globus). Figure 15 depicts the overheads incurred when running workflows through the system which showing that they are independent of the wait time. The unencrypted communication operated the quickest with an average client overhead of 1.7 seconds and WfMS overhead of 0.1 seconds. The gLite security added on average 1 second to the overheads seen by the WfMS and user. The larger overhead for security between the User and WfMS as opposed to the BPEL engine and the wait service can be attributed to the fact that the user is located on a remote network to the WfMS and as such the handshaking op-

Table 2
Experimental Resource specification

Resource Use	Location	Specification
Client	Computing	Intel Pentium 4 (3.2 GHz) with 2 MB cache and 1 GB memory. Ubuntu Linux 7.10.
WfMS, BPEL engine	Physics	AMD Sempron (3100+) with 256K cache and 512MB memory. RedHat Enterprise Linux 4. Apache Tomcat 5.5.23.
Wait Service	Physics	AMD Sempron (3100+) with 256K cache and 512MB memory. RedHat Enterprise Linux 3. Apache Tomcat 5.0.28.

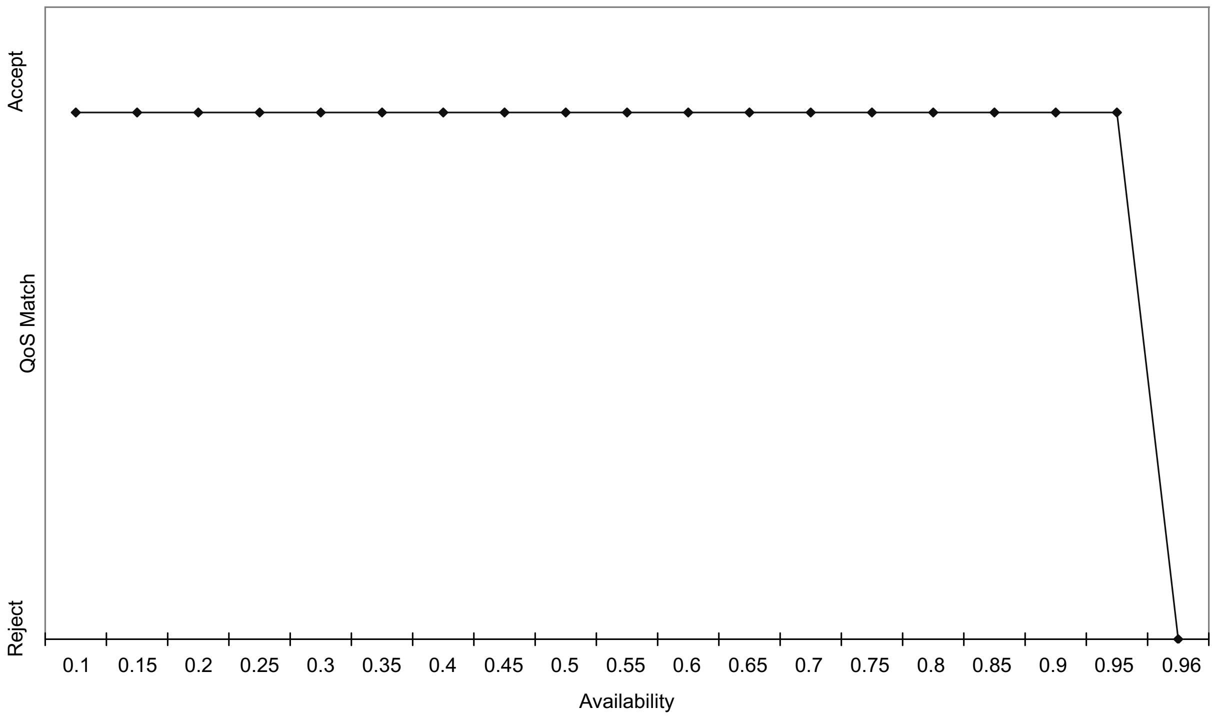


Fig. 21. Workflow acceptance vs availability.

erations will take longer than those on a local network. The Globus security mechanism adds approximately 60 seconds to the overheads seen by the user and WfMS. The reason for such a high overhead is not fully understood. One problem was exposed with the gLite security libraries in that there appears to be a memory leak. After approximately fifty workflow submissions the memory is exhausted. This issue was not observed with the other two security approaches. This is currently under investigation. Experiments were also conducted to determine the effect of deploying the workflow to the BPEL engine for each submission compared with just triggering it each time. The process of deploying the workflow was found to add an extra overhead of 2.2 seconds.

Overheads vs Number of submissions. Here the same workflow is used with the client submitting multiple workflows at the same time. As the gLite and Globus security mechanisms had both exhibited problems in the previous test, only unencrypted communication was used here. The wait time in all cases was held at 2000 milliseconds. Figure 16 illustrates the overheads incurred during multiple submissions to the WfMS. It can be seen that as the number of simultaneous submissions increase so does the overheads. With the client overheads continuing to increase while the others flatten out. This could be due to the fact that our submission client was not able to handle multiple simultaneous submissions. Alternatively this could be another manifestation of the memory leak observed with the gLite secure submissions in the previous ex-

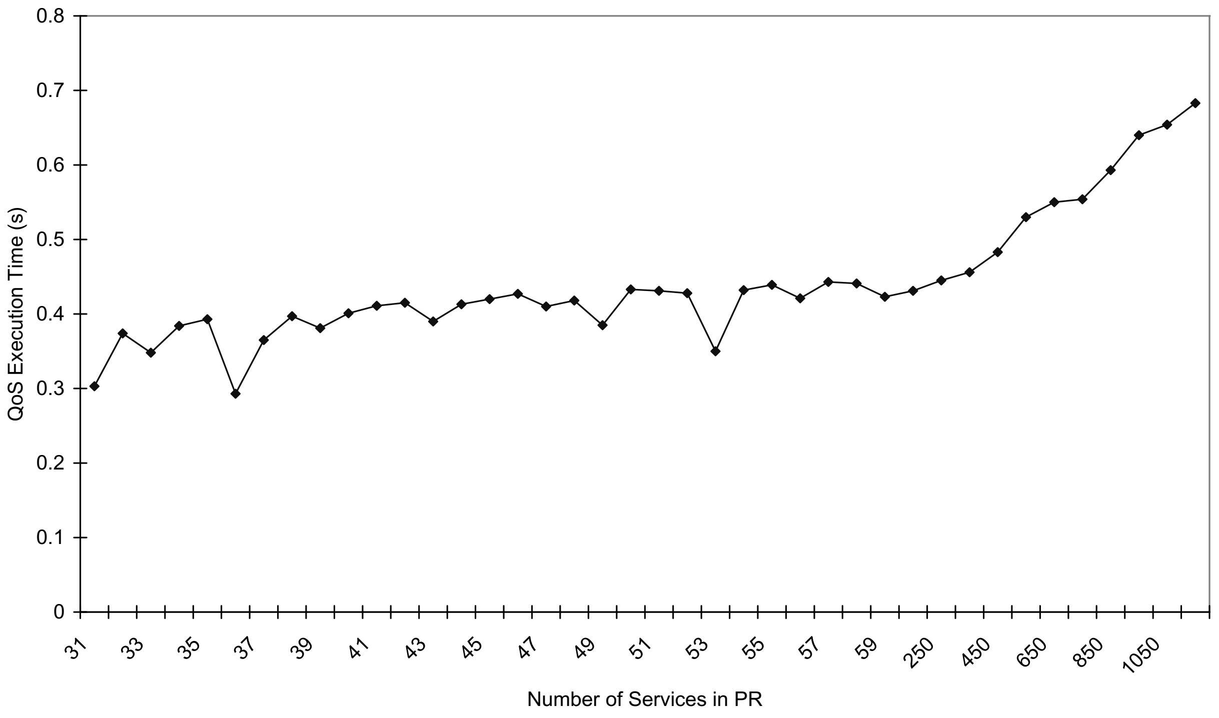


Fig. 22. Planning time vs PR size.

periment. Both the WfMS and BPEL engine appear to cope well under reasonable load. However, both exhibit an increase in overhead from multiple submissions. In both cases this can partly be attributed to the bursty nature of job submissions. Figure 17 illustrates the overheads from one run of this experiment. The X-axis represents the workflow submission number. As can be seen the overheads observed with the client jump sharply at around 21 submissions. This could be due to memory exhaustion or overloading of the submission client. The WfMS, Engine Server and Engine Client remain fairly consistent.

Overheads vs Number of submitters. Here rather than having multiple submissions from the same client multiple clients were used each repeatedly submitting the single workflow, thus removing the busty nature and load on the client. Figure 18 depicts the overheads as seen by one of the clients for varying numbers of clients. The figure shows the client observation time increases as the number of concurrent submitters increase. It can, however, be seen from this graph that the overhead is far lower than in the previous experiment as the clients are fully independent and do not interfere with each other. Suggesting that in fact the bursty nature of arrivals coupled with overheads in the multi-submission client were affecting the previous results.

Effect of the length of a workflow on overheads. In this experiment we vary the length of a workflow to determine the effect on overheads. Here the (request $\rightarrow$ wait n times $\rightarrow$ respond) workflow is used with varying values for n. The wait time for each call was held at 2000 milliseconds. Figure 19 illustrates the results from the experiment. As expected, as the number of stages in the workflow increases so do the overheads. It can be determined that the overhead per workflow stage is approximately 60 milliseconds, the majority of this time coming from the BPEL engine.

8.3. Planner experiments

Here we present experiments to evaluate the functionality and timings of the planner service within the WfMS. Rather than executing workflows we evaluate how the Planner can determine workflow placement and where the overheads come from.

Availability of services. In this experiment we preload the PR with a set of auto generated resource descriptions where the availability is generated from a uniform distribution. The submission document, to the WfMS, is augmented with a QoS requirement of the availability of the service and the Planner is set to output the percentage of resources which have been found matching the required level of availability. Figure 20 illustrates the percentage of resources matching the availability requested in the submission document. As expected as the availability is increased the proportion of matching resources decreases.

Ability to run a workflow. Here we look at the planners ability to run a workflow with QoS. Using the same PR information from the previous workflow we repeat the same experiment but this time look at the ability for the planner to accept the workflow – be able to locate resources matching the users requirements. Figure 21 shows the relation between the availability property of QoS requirements and possibility of finding matching services and hence being able to execute the workflow. As can be seen it is nearly always possible to find a resource with the desired level of availability. Though as can be derived from the previous experiment as the users availability requirement goes up the number of matching services goes down until it is no longer possible to find a matching resource. At which point the workflow is rejected.

Performance repository size vs planning time. In this experiment we determine how the size of the PR affects the planner. The PR is loaded with different numbers of records and the amount of time the planner takes to execute is recorded. From Fig. 22 we can see that as the number of available services in performance repository increases so does the execution time of the planner. When the number of services in the Performance Repository is low (less than 250) this effect is small. However, beyond this number the time increases rapidly.

9. Conclusion and future work

In this paper we have presented the real-time services used within the GRIDCC project to allow for integration of existing Grid technology with that of instruments. These include the end-to-end workflow pipeline which takes a user's design and implements it within the Grid, reservation services and performance repository. Workflows are defined through an editor which allows the augmentation of QoS requirements, defining the users expectations for the execution. The WfMS provides a mechanism for building QoS on top of an existing commodity based BPEL4WS engine. Thus allowing us to provide a level of QoS through resource selection from a priori information along with the use of advanced reservation.

The workflow editor and observer are areas of current development within the project. We are working with application scientists from the GRIDCC project to abstract the editor away from the BPEL4WS/QoS languages and make them more accessible to the scientist. We are investigating other techniques which will allow us to dynamically change the execution of the BPEL4WS workflow once deployed to the engine. Thus allowing for real time adaption of the workflow in light of the changing state of the Grid.

References

[1] A. Abdelnur and S. Hepper, *Porlet specification* (*jsr-168*), http://jcp.org/aboutJava/communityprocess/review/jsr168/.

[2] Active Endpoints. ActiveBPEL Engine (2.0). http://www. activebpel.org.

[3] A. Afzal, J. Darlington and A.S. McGough, *Stochastic Workflow Scheduling with QoS Guarantees in Grid Computing Environments*, in Proceedings of the Fifth International Conference on Grid and Cooperative Computing, Changsha, China, October 2006, 185–194.

[4] A. Afzal, A.S. McGough and J. Darlington, *Capacity planning and scheduling in grid computing environments*, Future Generation Computer Systems, doi:10.1016/j.future.2007.07.004, 2007.

[5] S. Andreozzi, T. Ferrari, S. Monforte and E. Ronchieri, *Agreement-based Workload and Resource Management*, in Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia, December 2005. IEEE Computer Society.

[6] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic and S. Weerawarana, *Business Process Execution Language for Web services version 1.1*, (BPEL4WS). http://www6. software.ibm.com/software/developer/library/ws-bpel.pdf, May 2003.

[7] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke and M. Xu, *Web Services Agreement Specification* (*WS-Agreement*), http:// www.ogf.org/documents/GFD.107.pdf.

[8] R. Bilorusets, D. Box, L. Felipe Cabrera, D. Davis, D. Ferguson, C. Ferris, T. Freund, M.A. Hondo, J. Ibbotson, L. Jin, C. Kaler, D. Langworthy, A. Lewis, R. Limprecht, S. Lucco, D. Mullen, A. Nadalin, M. Nottingham, D. Orchard, J. Roots, S. Samdarshi and J. Shewchuk, *Web Services Reliable Messaging Protocol* (*WS-ReliableMessaging*), ftp://www6. software.ibm.com/software/developer/library/ws-reliablemessaging200502.pdf, February 2005.

[9] D. Box, E. Christensen, F. Curbera, D. Ferguson, J. Frey, M. Hadley, C. Kaler, D. Langworthy, F. Leymann, B. Lovering, S. Lucco, S. Millet, N. Mukhi, M. Nottingham, D. Orchard, J. Shewchuk, E. Sindambiwe, T. Storey, S. Weerawarana and S. Winkler, *Web services Addressing* (*WS-Addressing*), August 2004.

[10] D. Burdett and N. Kavantzas, *WS Choreography Model Overview*, http://www.w3.org/TR/ws-chor-model/.

[11] L.F. Cabrera, G. Copeland, M. Feingold, R.W. Freund, H.T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, M. Little, A. Nadalin, E. Newcomer, D. Orchard, I.

Robinson, J. Shewchuk and T. Storey, *Web Services Coordination (WS-Coordination) 1.0*, ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf, August 2005.

[12] J. Cao, S.A. Jarvis, S. Saini and G.R. Nudd, *GridFlow: Workflow Management for Grid Computing*, in Proceedings of 3rd International Symposium on Cluster Computing and the Grid (CCGrid), Tokyo, Japan. IEEE CS Press, Los Alamitos, 12–15 May 2003.

[13] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, *Web services Description Language (WSDL) 1.1*, http://www.w3.org/TR/wsdl, March 2001.

[14] J. Clark and S. DeRose, *Xml path language (xpath) version 1.0*, http://www.w3.org/TR/xpath, 1999.

[15] C. Coenraets, *An overview of MXML: The Flex markup language*, http://www.adobe.com/devnet/flex/articles/paradigm.html, March 2004.

[16] D.J. Colling, L.W. Dickens, T. Ferrari, Y. Hassoun, C.A. Kotsokalis, M. Krznaric, J. Martyniak, A.S. McGough and E. Ronchieri, *Adding Instruments and Workflow Support to Existing Grid Architectures*, In Lecture Notes in Computer Science, volume 3993, Reading, UK, April 2006, 956–963.

[17] WCox, F. Cabrera, G. Copeland, T. Freund, J. Klein, T. Storey and S. Thatte, *Web Services Transaction (WS-Transaction) 1.0*, http://dev2dev.bea.com/pub/a/2004/01/ws-transaction.html, January 2004.

[18] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, A. Lazzarini, A. Arbree, R. Cavanaugh and S. Koranda, Mapping Abstract Complex Workflows onto Grid Environments, *Journal of Grid Computing* **1**(1) (2003), 9–23.

[19] Active endpoints. ActiveBPEL Designer. http://www. activeendpoints.com/products/activebpeldes/index.html.

[20] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto Jr. and H.L. Truong, ASKALON: a tool set for cluster and Grid computing, *Concurrency and Computation: Practice and Experience* **17**(2–4) (2005), 143–169.

[21] I. Foster and C. Kesselman, eds, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, July 1998.

[22] G. Grossman and E. Huang, *ActionScript 3.0 overview*, http://www.adobe.com/devnet/actionscript/articles/actionscript3, overview.html, June 2006.

[23] L. Huang, D.W. Walker, Y. Huang and O.F. Rana, *Dynamic Web Service Selection for Workflow Optimisation*, In Proceedings of the UK e-Science All Hands Meeting, September 2005.

[24] M. Welsh, D. Culler and E. Brewer, *SEDA: An architecture for wellconnected scalable internet services*, In Eighteenth Symposium on Operating Systems Principles (SOSP-18), October 2001.

[25] A. Mayer, A.S. McGough, N. Furmento, W. Lee, S. Newhouse and J. Darlington, *ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time*, In UK e-Science All Hands Meeting, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, Sep. 2003, 894–900.

[26] A.S. McGough, J. Cohen, J. Darlington, E. Katsiri, WLee, S. Panagiotidi and Y. Patel, An End-to-endWorkflow Pipeline for Large-scale Grid Computing, *Journal of Grid Computing* (February 2000), 1–23.

[27] A.S. McGough, L. Young, A. Afzal, S. Newhouse and J. Darlington, *Performance Architecture within ICENI*, In UK e-Science All Hands Meeting, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, Sep. 2004, 906–911.

[28] A.S. McGough, L. Young, A. Afzal, S. Newhouse and J. Darlington, *Workflow Enactment in ICENI*, In UK e-Science All Hands Meeting, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, Sep. 2004, 894–900.

[29] D. Nickull and F. McCabe, *SOA Reference Model*, http://www.oasis-open.org/committees/tc home.php?wg abbrev=soa-rm.

[30] Oracle. Oracle BPEL Process Manager, http://www.oracle.com/technology/products/ias/bpel/index.html.

[31] Y. Patel, A.S. McGough and J. Darlington, *QoS Support for Workflows in a Volatile Grid*, In Procedings of the 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, September 2006.

[32] San Diego Supercomputing Centre. Network weather service. http://nws.cs.ucsb.edu/ewiki/, 2005.

[33] T. Tannenbaum, D. Wright, K. Miller and M. Livny, *Condor – A Distributed Job Scheduler*, Beowulf Cluster Computing with Linux. The MIT Press, MA, USA, 2002.

[34] I. Taylor, M. Shields and I. Wang, Resource Management for the Triana Peer-to-Peer Services, in: *Grid Resource Management – State of the Art and Future Trends*, J. Nabrzyski, J.M. Schopf and J. Węeglarz, eds, Kluwer Academic Publishers, 2004, 451–462.

[35] W.M.P. van der Aalst, L. Aldred, M. Dumas and A.H.M. ter Hofstede, *Design and Implementation of the YAWL system*, In Proceedings of The 16th International Conference on Advanced Information Systems Engineering (CAiSE 04), Riga, Latvia, june 2004. Springer Verlag.

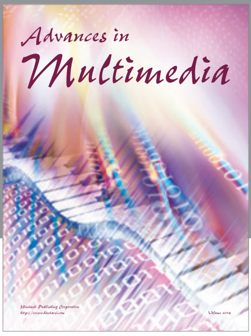
[36] W3C. Web Service. http://www.w3.org/TR/ws-arch/.

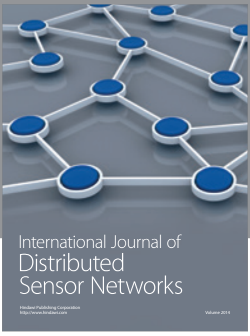
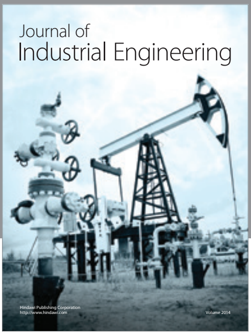
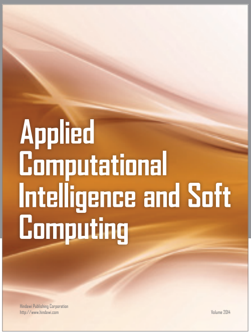
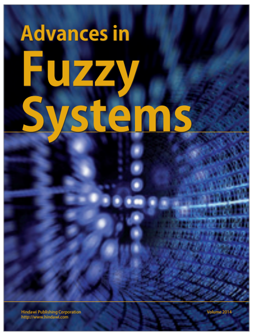
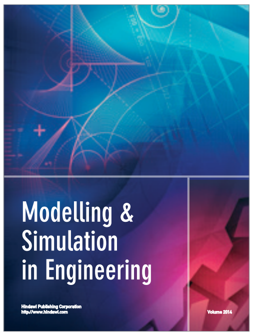
[37] L. Young and J. Darlington, *Scheduling in the Grid using High Level Policies*, PhD thesis, Imperial College London, University of London, 2004.

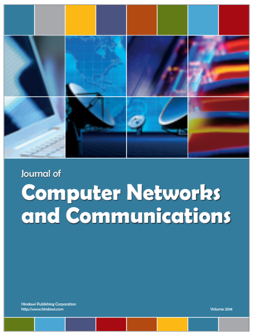
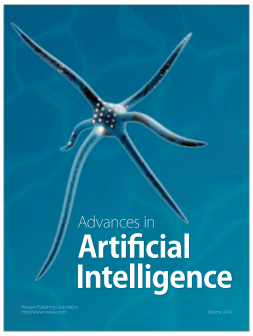
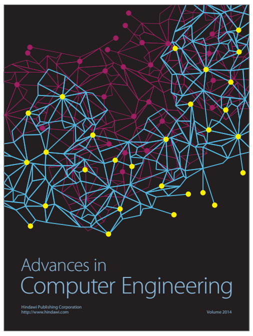
[38] L. Young, A.S. McGough, S. Newhouse and J. Darlington, Scheduling Architecture and Algorithms within the ICENI Grid Middleware, in: *UK e- Science All Hands Meeting*, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, Sep. 2003, 5–12.

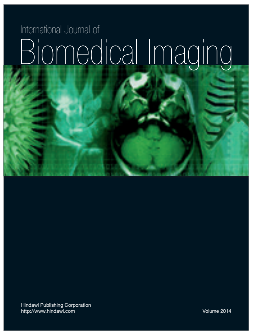
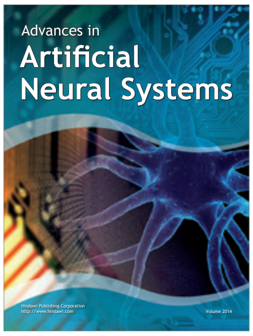
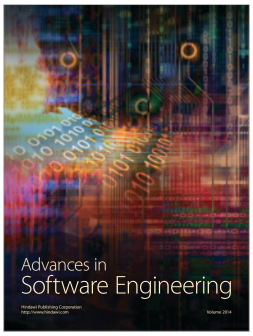
[39] J. Yu and R. Buyya, *A Novel Architecture for Realizing GridWorkflow using Tuple Spaces*, In Proceedings of 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004), Pittsburgh, USA. IEEE CS Press, Los Alamitos, 8 Nov. 2004.

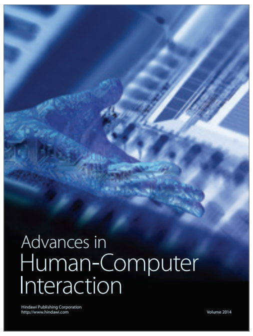
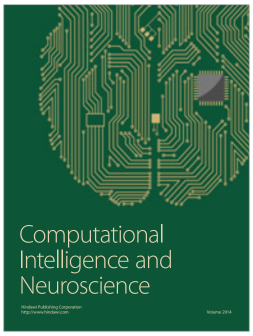
[40] J. Yu and R. Buyya, *A taxonomy of workflow management systems for grid computing*, GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10, 2005.


Advances in
Multimedia

The Scientific World Journal

International Journal of
Distributed Sensor Networks

Journal of
Industrial Engineering

Applied Computational Intelligence and Soft Computing

Advances in
Fuzzy Systems

Modelling & Simulation in Engineering


Journal of
Computer Networks and Communications

Advances in
Artificial Intelligence

Advances in
Computer Engineering

International Journal of
Computer Games Technology

International Journal of
Biomedical Imaging

Advances in
Artificial Neural Systems

Advances in
Software Engineering

Journal of
Robotics

Advances in
Human-Computer Interaction

Computational Intelligence and Neuroscience

International Journal of
Reconfigurable Computing

Journal of
Electrical and Computer Engineering