# Dynamic service selection in workflows using performance data

David W. Walker[*], Lican Huang, Omer F. Rana and Yan Huang
*School of Computer Science, Cardiff University, 5 The Parade, Roath, Cardiff CF24 3AA, UK*

**Abstract**. An approach to dynamic workflow management and optimisation using near-realtime performance data is presented. Strategies are discussed for choosing an optimal service (based on user-specified criteria) from several semantically equivalent Web services. Such an approach may involve finding "similar" services, by first pruning the set of discovered services based on service metadata, and subsequently selecting an optimal service based on performance data. The current implementation of the prototype workflow framework is described, and demonstrated with a simple workflow. Performance results are presented that show the performance benefits of dynamic service selection. A statistical analysis based on the first order statistic is used to investigate the likely improvement in service response time arising from dynamic service selection.

## 1. Introduction

The Grid computing community is converging on a service-oriented architecture in which applications are composed from geographically-distributed, interacting Web services, and are expressed in a workflow description language, typically based on XML. Such applications are often executed under the control of a workflow engine. Workflow techniques provide a means for a collection of services to be combined dynamically. However, although there is broad consensus on the overall architecture of the Grid, there are many unresolved issues that are still active research areas and for which implementations are not publicly available. One such area is the discovery and selection of services that may be combined in a workflow.

In a service-rich environment, it is possible that multiple copies of a service may exist. For instance, there may be multiple semantically equivalent versions of a service on different machines, each making use of a different implementation (such as programming language or algorithm). Selecting an optimal service from this set of equivalent services is a decision that is often undertaken manually by a user. Most techniques for selecting from a set of such services are determined at design time. There is, however, little support for dynamically choosing a Web service as part of a workflow enactment strategy.

This paper presents a mechanism to discover, select, and invoke a Web service at runtime – thereby providing a means for optimizing a workflow by dynamically binding a service name to a service instance. This is also known as "just-in time scheduling", and corresponds to a late binding operation, whereby Web service instances are resolved based on a user-defined set of optimization criteria. The motivations for this work include improving fault-resistance and performance based on factors that cannot be determined at design time. When one service instance fails, the workflow engine should be able to utilize another service instance. For computationally intensive Web services, selecting services with a specific performance profile is beneficial to the execution of the entire workflow. There are various use scenarios that necessitate the choice of "optimal services" only at run-time, such as an image analysis/visualization service that needs to respond within a particular time. Another scenario involves the choice of services in a changing environment, where particular service instances may not persist over long time periods.

[*]Corresponding author: Prof. David W. Walker, School of Computer Science, Cardiff University, 5 The Parade, Roath, Cardiff CF24 3AA, UK. Tel.: +44 (29) 20874205; Fax: +44 (29) 20874598; E-mail: david@cs.cf.ac.uk.

ISSN 1058-9244/07/$17.00 © 2007 – IOS Press and the authors. All rights reserved

Many scientific workflow applications are compute-intensive, and may be long-running – lasting weeks or even months. Here, the selection of "optimal" Web services among the available ones can shorten the computation time. When running a complete scientific workflow application, if one service fails, the whole workflow must be run again. Dynamic Web service selection involves the discovery of a list of candidate services, and if one service fails, then the next one can be tried, thereby avoiding the need to repeat the whole workflow. Scientific workflow applications may require heterogeneous computing resources such as supercomputers, clusters, and networks of workstations/PCs. Some applications need a guarantee of completion within a given period of time, which requires some way of predicting the likely completion time. Therefore, dynamic selection involves choosing a suitable service according to the current conditions (such as workload on the machine where the service is hosted) and service performance models.

In general, it is not necessary to dynamically select every service in a workflow. The services for which dynamic selection will be most effective in reducing the overall workflow makespan are those lying on the critical path, and that are sufficiently long-running for the benefits of dynamic selection to outweigh the overheads incurred. Given a detailed performance model of a workflow it would be possible to determine its critical path, and hence to identify appropriate candidates for dynamic selection. Such a model might be based on the number of operations performed by services and the amount of data transferred between them, although other quality-of-service factors could also be used to weight nodes and edges in the workflow. In such cases the relative importance of the different factors considered (for example, the communication-to-calculation ratio) can be used to determine a single weight for each node and edge in the workflow, or the problem can be treated as a multivariate constraint optimization question [16]. This approach can be used prior to workflow execution to find which services should be selected dynamically. This static approach ignores the fact that determining the critical path of a workflow in a dynamic, service-rich environment can only be done at runtime, and that the critical path may change as the workflow executes. This leads to the predictive scheduling approach, in which an execution schedule is made at the start of workflow execution based on currently known or assumed conditions, but this schedule may change at runtime as conditions change [3,17,21].

In this paper it is assumed that some procedure exists for deciding which services will be selected dynamically – for sufficiently simple workflows this may be done manually. As discussed in Section 4.1, a Proxy Service is used as a placeholder for a service that is to be dynamically selected in a workflow, and service selection is performed through interaction of this Proxy Service with a Discovery Service and an Optimisation Service.

The structure of the rest of this paper is as follows. In Section 2 the motivation of our research is presented, and models of service binding are discussed. Section 3 discusses related work. In Section 4 our architecture for dynamic Web service selection for workflow optimization is presented. Section 5 outlines the implementation of a prototype of this architecture. Section 6 describes service selection experiments performed with our prototype implementation, and discusses the performance results. A best case analysis of dynamic service selection, based on the first order statistic, is presented in Section 7. Finally, Section 8 presents conclusions and ideas for future work.

## 2. Motivation and discussion

Services interacting in a workflow may be specified at an abstract level. By this we mean that the semantics of the service are specified, but not the implementation. It is also possible that in such an abstract workflow the syntax of the service interfaces may also not be fully defined. Thus, an abstract workflow is similar to an algorithm in which the processes that transform the inputs into the desired outputs are specified without referring to the actual software on specific computers that will carry out those processes.

At some point before an abstract workflow can be executed, the abstract services have to be changed into concrete services by binding them to service implementations on particular machines. The details of this conversion from an abstract to a concrete workflow are referred to as a binding model. A key feature of the binding model, that has a large impact on the scope for optimizing the execution of a workflow, is *when* the binding of services actually takes place. We consider three possibilities:

1. Binding takes place at design time (i.e., when the workflow is first composed).
2. Binding takes place immediately before execution of the workflow begins.
3. Binding takes place immediately before a service needs to be executed.

These three binding models will be referred to as the early, intermediate, and late binding models. In general, when binding takes place some form of service discovery and optimization may be performed. However, the longer the time between binding and execution, the greater the possibility that the information used in the service discovery and optimization processes at binding will be out-of-date when execution takes place. For example a service may no longer be available, or the service chosen at bind time may no longer be optimal because resource utilization characteristics have changed. Thus, the late binding model offers the best opportunity for optimizing the performance of individual services because the most recent data is used to make the choice. However, a disadvantage of the late binding model is that although the performance of individual services may be close to optimal, the overall performance of the whole workflow may not be. This is because late binding does not make it possible to take into account the time to communicate data between connected services. For example, consider two services, A1 and A2, in an abstract workflow in which an output port of A1 is connected to an input port of A2. With a late binding model it may be decided that concrete service implementations on machines M1 and M2 are the optimal choice for A1 and A2, respectively. However, if a large amount of data must be transferred between the two services it may be better to choose concrete service implementations on machines with a high bandwidth connection, or where A1 and A2 are hosted on the same machine. Thus, if we want to optimize the workflow as a whole an intermediate binding model is better. Hybrid models can also be considered in which early or intermediate binding is used, but if a service is found to be unavailable at runtime then on-the-fly service discovery and optimization is performed, as in the late binding case. Early binding is usually referred to as static service invocation, and late binding as dynamic (or just-in-time) service invocation.

## 3. Related work

Significant work has already been undertaken in the area of Grid-based workflow systems (see [25] for a survey). The focus of these workflow systems varies – ranging from specialist workflow editors/composition tools, and portal technologies to assess the current status of workflow execution, to semantic annotation tools that treat workflow as a planning problem [4]. The use of metadata to describe workflow elements and workflow graphs [15] is becoming more common as an aid to service discovery and selection. The Service-Globe environment implements dynamic service selection [9] within a context framework that maintains and manages information about the current service environment. Ontologies for describing services can also be used in service discovery. A framework and ontology for dynamic Web service selection is proposed in [13]. The OWL-S ontology [23] provides a description of services that can be used to map an abstract service to a specific service implementation [22]. Mathematical service discovery using a "matchmaking shell" that can be customised is described in [12]. The use of ontological reasoners may in the future play an important role in the discovery and scheduling of services in distributed workflows composed from third-party services.

A software architecture for workflow processing on the Grid is described in [14]. This architecture is largely based on the ICENI environment, and considers both static and dynamic service discovery and selection. Liu and co-workers have proposed the use of quality-of-service criteria to support service selection [11]. The Pegasus system [5] uses AI planning techniques to map abstract workflows to resources, including the reduction of the abstract workflow if intermediate results are already available and the re-mapping of workflows when resources become unavailable or failures occur for other reasons [6,20]. The scheduling of activities through the interaction of a planner and a workflow execution system can also be extended to handle data placement tasks in addition to computational tasks. The Stork system [10] is a scheduler for data placement activities on the Grid that can recover from failures, and through interaction with the DAGMan execution system, make scheduling decisions using both computational and data placement metrics. Zhen and Parashar have developed the Rudder system for dynamic composition of workflows, based on the use of software agents for service discovery and selection [28], and stress the semantic aspects of this process. Adaptive scheduling of workflows has also featured prominently in the Grid Application Development Software (GrADS) project [2,18].

The research presented in this paper differs from that discussed above in that a Proxy Service is used to discover and select services based on the expected service response time, which in turn is assumed to depend on the processor speed and current load of the service hosts. Our current model ignores the time taken to transfer data to, and from, a service. The research of Zangrilli and Lowekamp [26] addresses this issue by

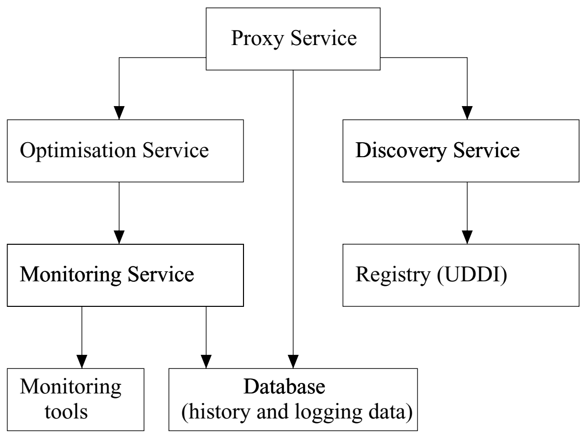

Fig. 1. Dynamic Service Selection for Workflow Optimization.

dynamically selecting services based on network performance. A proxy service is used to passively monitor network performance through the Wren monitoring toolkit [27], and service selection is based solely on the available bandwidth between the proxy and the services.

## 4. Dynamic service selection architecture

In a service-rich environment, multiple copies of a service may co-exist with different performance profiles. As discussed above, usually service selection cannot be done at design time because the service response times cannot be accurately predicted at that time. Figure 1 shows the architecture of our dynamic service selection mechanism, that takes into account monitoring information obtained from service hosts [8]. Currently, the Universal Description Discovery and Integration (UDDI) registry is used to host service descriptions. This registry primarily provides an identifier for a service, service metadata for a very restricted semantic definition, and the location of the WSDL file describing the service interface (via a URL). The database shown in Fig. 1 contains historical data about previous service invocations, such as the response time from a given service. When multiple semantically equivalent copies of a service are found by the Discovery Service, the Optimization Service selects a service based on the history database, and/or real-time data such as the processor speed and current load of the machine hosting the service. Such data are collected using monitoring tools such as Ganglia [19], and/or another locally available recording tool.

### *4.1. Proxy Service*

The Proxy Service is used in a workflow script as an adaptor for dynamically selecting and binding to a Web service. In a workflow application, some activities are critical in terms of their execution time or fault-tolerance properties. We use a Proxy Service to act as a place holder for such services, and allow them to be bound to a physical instance of a service. Among these semantically equivalent services an optimal service is selected and invoked, and its output is passed to the next activity in the workflow. The Proxy Service is itself a Web service. The parameters passed to the Proxy Service include service metadata such as: *queryMethod* which specifies the query mechanism used by the Discovery Service to search for services; *optimizationMeta* which gives a description of the problem-specific optimization model being used; *optimizationMode* which is the mode for selecting a Web service, based on parameters such as execution time, degree of trust in results, etc.; *operation* which corresponds to the actual function performed by the late-binding service and the parameters passed to it. *serviceProxyReturn* is the returned response, and contains a reference to an end point handler for the selected service.

If multiple workflows are executed concurrently and use the same Proxy Service, it is possible that the Proxy Service will become a bottleneck and any performance advantage arising from dynamic service selection may be lost. In such cases more than one Proxy Service could be made available for use by the workflows. The Proxy Service instance used by a workflow could be fixed at design time, or by using a simple algorithm (such as random selection) at runtime.

### *4.2. Discovery Service*

The queryMethod is used by the Discovery Service to find and filter the services available. There are three methods: byNAME, byMETA and byONTOLOGY. The byNAME method is used to query all semantically equivalent services with a specified name. The byNAME method would typically be used where the services are registered by the same business entity but with different ways to access the service (i.e., the existence of different bindingTemplates in UDDI). No extensions are needed to the information contained in the UDDI registry. The byMETA method is used to query all services by examining the service metadata. The byMETA method would typically be used where all service providers conform to a particular metadata specification, and have the ability to publish their own services in the UDDI registries. By using such metadata, the Discovery Service can find semantically equivalent services. This method needs to register metadata information in the UDDI registry by a method method similar to that of Miles et al. [15]. Service properties include service name, service ID, list of operation names, operation IDs, and their input, as well as output, and data types. These items are registered in the vector of description entities in the business service entity. The byONTOLOGY method is used to search for all semantically equivalent services based on a description processed by an ontological reasoner. This is usually referred to as semantic matchmaking, and would typically be used where there are many service providers who publish their service descriptions according to the schema encoded in a service ontology. The service providers are loosely connected or without any relationship. Currently only the byNAME method has been implemented in the Discovery Service of our prototype implementation.

### 4.3. *Optimization service*

The input to the Optimization Service is the list of semantically equivalent services – and the output is the "best" service based on the criteria identified by the user. The Optimization Service uses real-time data or previously recorded historical data to make this selection from the candidate services.

When the Proxy Service receives the end point reference of the selected service returned by the Optimization Service, it invokes the selected service dynamically. The WSDL file is downloaded and parsed. The input/output parameters to/from the selected Web service are marshalled via the Proxy Service. The actual contents of the input and output data structures are described in XML. We may specify adaptors to transform the String type of input or output data of a Proxy Service into various data types to match the ports of other Web services which link the Proxy Service to other services in the workflow script.

The Proxy Service supports the fault-tolerant execution of workflow scripts. If a service fails, then an alternative semantically equivalent service will be substituted. As the Proxy Service is independent of the workflow engine used, specific logging data can be obtained and stored. Such data includes the identity of the Web service(s) that were invoked and the source of the input data, for instance. Scientists can judge the trustworthiness of the scientific conclusions obtained

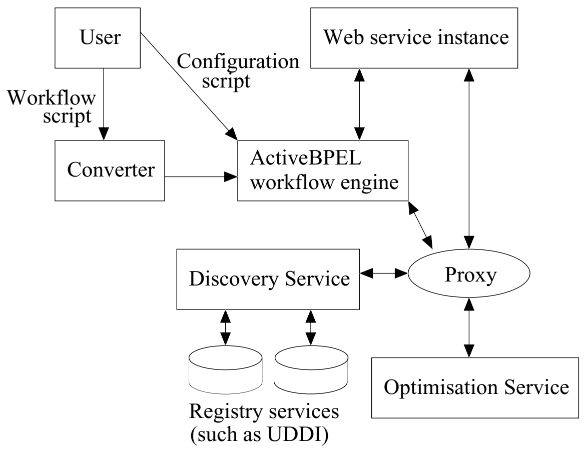

Fig. 2. Architecture of the WOSE system.

through the execution of the workflow by examining the logged data.

Figure 2 describes the overall architecture of the system. A user provides a workflow script and a configuration script. The workflow script may be translated into an XML description that is appropriate for the particular workflow engine being used – in this instance ActiveBPEL.[1] The workflow engine then interacts via the Proxy Service with the Discovery and Optimization Services.

Figure 3 shows the sequence diagram for workflow execution, which integrates the Proxy Service into the workflow and performs service discovery and selection. As shown in Fig. 3, the workflow is first deployed into the workflow engine. Optionally the workflow script may first be converted by an XSLT script into a form that can be understood by the workflow engine – this capability has previously been described in [7]. When a client invokes the workflow some of the services are invoked through the Proxy Service and some are not. In the latter case the workflow engine invokes the service directly and receives back the result, as shown by interactions 2A and 3A in Fig. 3. If a service is to be invoked through the Proxy Service then the workflow engine first invokes the Proxy Service by passing it the information needed to perform the service discovery and selection phases. This is shown by step 2 in the figure. The Proxy Service then invokes the Discovery Service (step 3), which returns a list of services capable of satisfying the service request (step 4). The Proxy Service then invokes the Optimisation Service and passes to it the list of candidate services (step 5). In

[1] http://www.active-endpoints.com.

Fig. 3 the Optimisation Service queries the Monitoring Service and receives back performance data about the service hosts (steps 6 and 7), which it then uses to select the service to be invoked from the list of candidates. An end-point reference to the selected service is then passed back to the Proxy Service (step 8). The Proxy Service then invokes the selected service and receives back the result (steps 9 and 10), which is then returned to the workflow engine (step 11). On completion of the workflow the final output is returned to the client that initiated the execution of the workflow (step 12).

The fact that the Proxy Service acts as an intermediary in passing the results from the invoked service back to the workflow engine imposes requirements on the capabilities of the Proxy Service – for example, it must have sufficient storage to handle the data from any service it is required to invoke. Optimizations in the transfer of data between services have not been examined in the work presented here. However, there is clearly scope to improve this aspect of workflow execution by avoiding unnecessary steps in data handling.

## 5. Implementation

Our framework for dynamic Web services includes service discovery, selection, and invocation. Web services are discovered that match the requirements identified by the user in the configuration file sent to the workflow engine (see Fig. 2). Then the best service is selected according to the performance data, and the selected service is invoked. This service selection scenario happens at run-time, during the execution of a workflow application. As discussed in Section 4.1, the Proxy Service plays an important role in the service selection scenario. The invocation of the Proxy Service is done by passing it an XML description document that contains an abstract definition of the required service.

The following list presents the scenario of a typical dynamic Web service selection procedure.

- Step1: The Proxy Service receives an XML document describing a required service.
- Step2: The document is sent to a Discovery Service which returns a list of matching services.
- Step 3: If the list of matching services is empty go to step 7.
- Step 4: Send the service list to the Optimization Service which selects the service with the best performance using its own performance selection mechanism and available performance data. The Optimisation Service returns the selected service.
- Step 5: Invoke the service. If the invocation is successful, go to step 7.
- Step 6: Remove the failed service from the service list and then update the logging data by adding the failure record into it. Go to step 3.
- Step 7: Stop

The Optimization Service selects a service from a list of services by using the service performance data collected by a Monitoring Service. A Monitoring Service obtains performance data for a particular service using monitoring tools. A database may be used to keep historical performance data for use in performance prediction. Currently the Optimization Service selects the service with the highest “performance factor”. The performance factor, which involves only the CPU speed and load of the machine that hosts a particular service, is defined as follows:

$$\frac{CPUspeed}{(LoadAverage + 1)}$$

The appearance of the CPU speed in the performance factor takes into account its processing power. The load average is a measure of the number of active jobs on a system. When the load average is high a system is expected to respond more slowly.

It should be noted if a failed service instance is removed from the service list in step 6, this means that it will be not be considered further in the current service selection process. The next time the Proxy Service is used to select a service any matching services that had previously failed will be considered again for selection. Thus, if a service is temporarily unavailable it will automatically become a candidate for selection once it becomes available again. To permanently exclude a service from future use it must be removed from the service registry.

## 6. Dynamic workflow experiments

The example used here makes use of three Web services. The invokebrowser service retrieves raw protein data from an input URL. This data is then passed to the getproteinseq service which extracts a protein sequence from it. This protein sequence is next input to the blastall service which searches for matches to the sequence in a protein database by using the Basic Local Alignment Search Tool (BLAST). These matches are then returned to the client that invoked the workflow. Thus, the workflow consists of a linear arrangement of three services. The invokebrowser and getprotein-

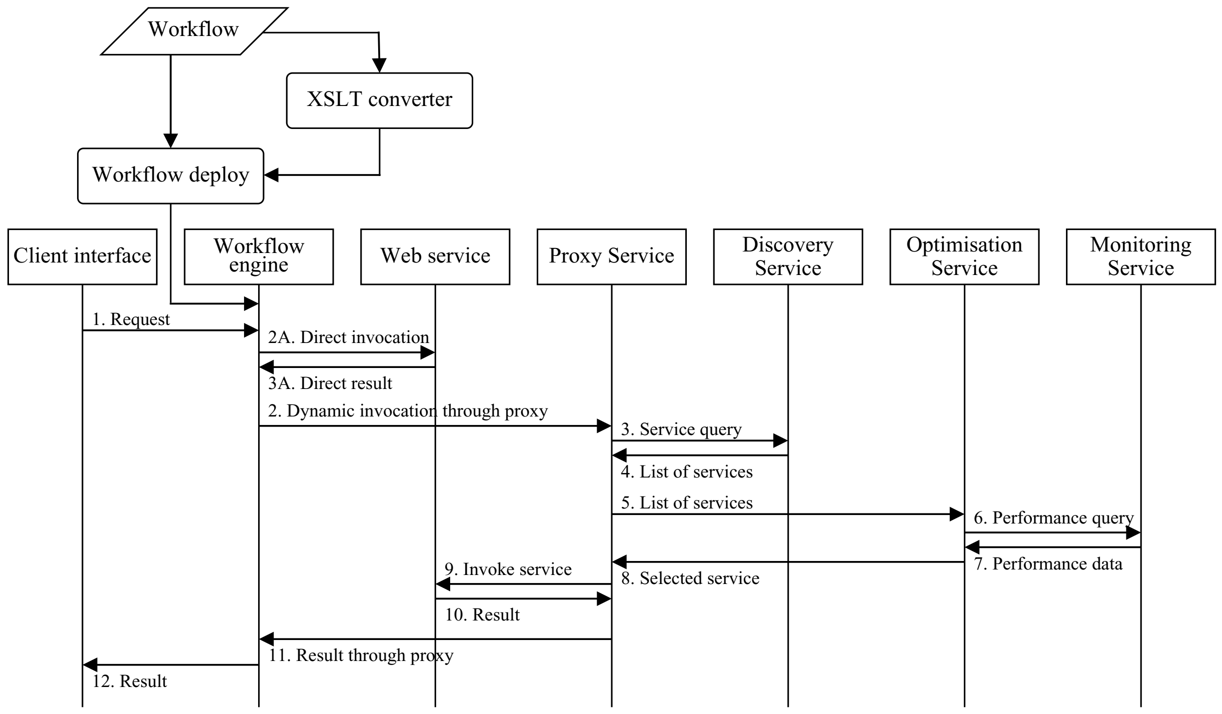

Fig. 3. Sequence of messages for workflow incorporating dynamic Web service selection.

seq perform simple tasks that are quickly executed and input and output small amounts of data. The blastall service is long running, taking of the order of tens of minutes to execute in our experiments. It also inputs and outputs quite small amounts of data (typically a few kbytes). In the experiments described below, the invokebrowser and getproteinseq services are invoked directly by the workflow engine, and the blastall service is invoked through a Proxy Service. Multiple instances of the blastall service were installed on different machines, and a blastall service is chosen dynamically from these based on realtime and recorded data obtained from Ganglia [19]. In our workflow script, a Proxy Service is initially used as a placeholder for a blastall service and by interaction with the Discovery and Optimization Services, a service is selected and dynamically executed.

## 6.1. Setup

In the experiments Tomcat v5.0.12 and Axis v1.2.1 were used as Web service containers. The performance history database used is a mySQL database (version 4.1.12). The JUDDI registry maintained by the Welsh e-Science Centre is used as the UDDI registry.[2] The invokebrowser and getproteinseq services were installed on a Windows XP laptop. An instance of the blastall service was installed on each of six identical Linux workstations running Redhat 7.0. All instances of the blastall service were registered in the JUDDI registry. The blastall service wraps the Java-based BLAST algorithm[3] by executing the blastall command with default arguments. The database queried by the blastall service was downloaded from the same web site, and contains a collection of protein sequences. The Monitoring Service uses Ganglia v3.01 to retrieve realtime performance data from service hosts. Ganglia is used due to the significant interest in this tool within the Grid community – however, our implementation is not restricted to the use of this tool.

## 6.2. Results

In the workflow used in the experiments the parameters passed to a proxy service are assigned as follows: serviceMeta is set to “blastallservice”, queryMethod to “byNAME”, operation to “serviceblastall”, optimizationMode to “performance”, and optimizationMeta to

[2]http://www.wesc.ac.uk/services/uddi/index.html.

[3]Available from http://ncbi.nih.gov/blast/.

"NONE". The URL parameter input to the invokebrowser service is http://us.expasy.org/uniprot/P05131.fas from which the service retrieves the KAPB1_BOVIN protein sequence from the UniProt database (this is described at http://www.expasy.org/uniprot/KAPB1_BOVIN). The input to the getproteinseq service is the output of the invokebrowser service. The getproteinseq service extracts the protein sequence consisting of 351 characters, where each character represents an amino acid. The getproteinseq service then passes this character string as an input parameter to the Proxy Service.

The process of workflow execution is as follows. The client (a simple web interface) invokes the ActiveBPEL workflow engine. When the Proxy Service is invoked, the six blastall service instances are retrieved from the JUDDI registry by the Discovery Service. The Optimization Service then selects one of these services based on performance data from Ganglia. The Proxy Service then invokes the selected blastall service, and returns results to the next activity. Finally, the ActiveBPEL engine outputs the result to the client interface for display to the user.

The relationship between service response time and the load on the service host was investigated with three different experiments. Experiment 1 was conducted using a single service on a remote host. The aim of this experiment is to test if there is a correlation between the service response time and the performance factor. To this end we try to keep the load on the service host constant by executing a number of long-running jobs to create a background workload. Varying the number of jobs allows different loads to be created on the service host. For each instance of Experiment 1, the load on the service host was fixed by executing a suitable number of long-running jobs to create a background workload. While these jobs were running the workflow was invoked on the client machine and the response time for the blastall service was recorded. This time is the period from the invocation of the Proxy Service by the workflow engine to the return of the results from the service, and thus includes the time to run the discovery and optimisation services. The load average was recorded soon after the invocation of the proxy service. The results of Experiment 1 are shown in Fig. 5 in which each symbol plotted represents one workflow invocation. Although there is a correlation between the service response time and the load average – in general a larger load average results in a longer service response time – there is considerable scatter in the values for any particular load average. This arises because the experiments were not carried out on dedicated machines, and hence the background workload could not be fully controlled. This reflects the type of environment in which service hosts are often used.

Experiment 2 investigated the impact on the service response time of a varying workload on the service hosts. A randomly varying synthetic workload was run on each of the six identical service hosts, and the workflow was executed repeatedly within a loop (only one instance of the workflow was executing at any one time). As in Experiment 1, the blastall service was invoked through the proxy service, and the service response time measured from the invocation of the proxy service to the return of the results was recorded, together with the load average at the start of the proxy service invocation. The results, shown in Fig. 8, are similar to those of Experiment 1 in the sense that there is a general trend for a larger load average at invocation time to result in a longer service response time. Although in both experiments there is a large scatter in the results, it is reasonable to expect that service selection based on the load average at invocation time would result in a faster response than selecting a service at random.

No method can accurately predict future load if no restrictions are placed on the use of the service hosts, and hence any method of estimating which service host will complete execution sooner will give the wrong answer sometimes – which accounts for the scatter in the results for Experiments 1 and 2. Experiment 3 sought to determine if service selection based on the load average at invocation time results in a faster response than selecting a service at random. Experiment 3 consists of two phases. In the first phase a varying synthetic workload was run on one of the service hosts, and the service response time was recorded for several successive executions of the workflow. The average service response time was then computed. In the second phase the same synthetic workload was run on three service hosts. The workflow was executed and the proxy service was used to dynamically select the service host based on the performance factor (or equivalently the load average since the three service hosts were identical). This was repeated several times, and the average service response time was computed. The average service response time for phase 1 and phase 2 was 4252 seconds and 932 seconds, respectively. This demonstrates that dynamic selection based on the load average at invocation time can result in better performance.

The performance factor identified here could be modified based on data supplied by a monitoring service. For example, if services require a large amount of memory to execute we could also use the currently available

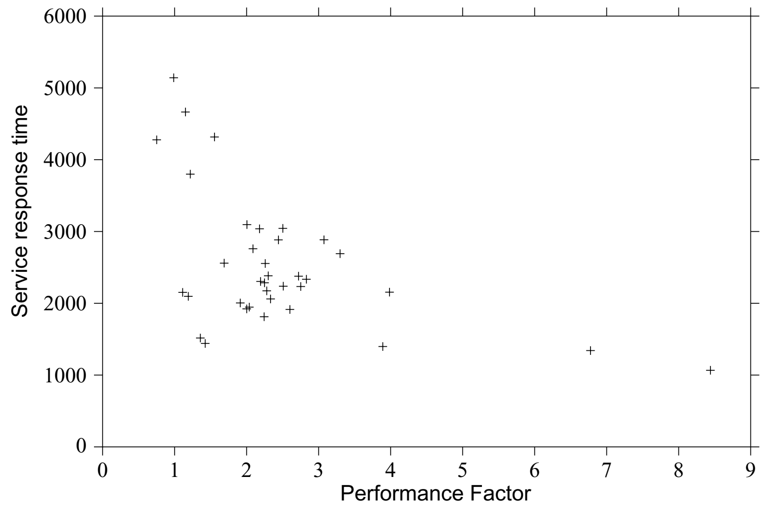

Fig. 4. The correlation between service response time (in seconds) and the performance factor based on the one-minute load average.

memory, alongside load average and processor speed, as a way to chose a host. It should be noted that the Optimization Service provides a reference to the service host with the largest performance factor *at the current time*. Hence, future use of a service on such a machine may not provide identical behaviour. It may, therefore, be a good idea to make use of a long term history of service host performance to support service selection decisions (also measurable with Ganglia).

## 7. Best case statistical analysis

The experiments discussed in Section 6.2 support the view that dynamic service selection can reduce service response times (and hence also the workflow makespan), even when the service hosts are not dedicated to running under the control of the workflow environment. In such circumstances third parties can impose loads on the service hosts, making it difficult to accurately predict the load on a host during the execution of a service. We now consider the maximum performance improvement that can be expected when selecting services on non-dedicated resources. This analysis is based on the first order statistic.

Suppose there is one service on each of $N$ identical hosts, and that the service response time for each service follows a probability function $f(t)$. Thus, the probability that the service response time is between $t$ and $t + dt$ is $f(t)dt$. The probability distribution is sampled N times, giving the service response time values $T_i$ for $i = 1, \ldots, N$. This sampling corresponds to measuring the service response time on each of the $N$ service hosts at some time. The service response times are then relabelled as $Y_i$ ($i = 1, \ldots, N$), such that $Y_1 \leqslant Y_2 \leqslant \cdots \leqslant Y_N$. $Y_i$ is termed the $i$th order statistic. Now suppose the sampling procedure is repeated $K$ times – that is, the following steps are repeated $K$ times:

1. Sample the probability distribution $N$ times.
2. Relabel the resulting service response times so that $Y_1 \leqslant Y_2 \leqslant \cdots \leqslant Y_N$.

Consider the expected value of the first order statistic, $Y_1$, as $K$ becomes large. This represents the best performance improvement that could be achieved through dynamic service selection, in the sense that, this would be the average service response time if the service selection procedure always chose the fastest service host for each of the $K$ service invocations. This is, therefore, a best case analysis, based on the key assumptions that the probability distribution of service response times is known, and is the same for all $N$ service instances.

It can be shown that the probability function of the $i$th order statistic is given by [24]:

$$f_i(t) = \frac{N!}{(N-i)!\,i!}[F(t)]^{i-1}[1-F(t)]^{N-i}f(t) \quad (1)$$

where $F(t)$ is the cumulative density function (also known as the distribution function) defined by:

$$F(t) = \int_{-\infty}^{t} f(x)dx \quad (2)$$

It is instructive to carry through the evaluation of the expected value of the first order statistic for a particular

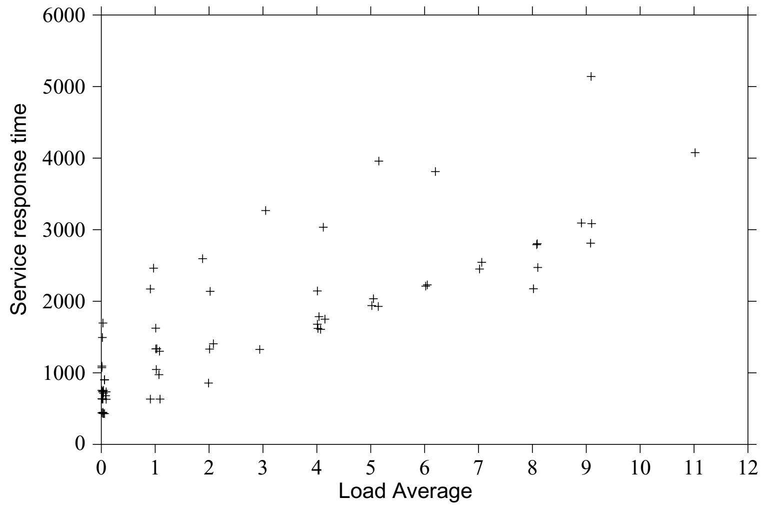

Fig. 5. Service response time in seconds as a function of the one-minute load average for Experiment 1.

probability function. This will show the relationship between the shape of the probability function (in particular its width) and the best performance improvement that could be expected through dynamic service selection. For this purpose it will be assumed that the cumulative density function takes the form of the Gamma distribution:

$$F(t) = 1 - \frac{\Gamma(\alpha, t/\theta)}{\Gamma(\alpha)} \tag{3}$$

where $\Gamma(\cdot)$ is the Gamma function, $\Gamma(\cdot,\cdot)$ is the upper incomplete Gamma function, and $\Gamma(\alpha, 0) = \Gamma(\alpha)$. In this case, the probability function takes the following form:

$$f(t) = \left(\frac{1}{\Gamma(\alpha)\theta^{\alpha}}\right) t^{\alpha-1} \exp\left(-t/\theta\right) \tag{4}$$

It should be noted that the mean and variance of the Gamma distribution are given by $\mu = \alpha\theta$ and $\sigma^2 = \alpha\theta^2$, respectively. Substituting the expressions for $F(t)$ and $f(t)$ in Eqs (3) and (4) into Eq. (1), and taking $i = 1$, the following expression is obtained for the expected value of the first order statistic:

$$E_1(N, \alpha, \theta) = N\theta \int_0^{\infty} x \left(\frac{\Gamma(\alpha, x)}{\Gamma(\alpha)}\right)^{N-1} \frac{x^{\alpha-1}\exp\left(-x\right)}{\Gamma(\alpha)}\, dx \tag{5}$$

Equation (5) shows that the first order statistic depends linearly on $\theta$, and nonlinearly on $N$ and $\alpha$.

Since $\alpha = (\mu/\sigma)^2$ and $\theta = \sigma^2/\mu$, the expected value of the first order statistic can also be expressed as a function of the number of service instances, $N$, and the mean and variance of the Gamma distribution, that is, as $E_1(N, \mu, \sigma)$. The behaviour of $E_1(N, \mu, \sigma)$ will now be examined when the mean, $\mu$, is kept constant and the variance, $\sigma^2$, is varied, for different values of $N$. Table 1 shows how $E_1(N, \mu, \sigma)$ decreases as the number of available services, $N$, increases for $\sigma = 50$, 100, 200, and 300. A mean of $\mu = 1000$ was used, although the actual value of the mean is not important here since if both $\mu$ and $\sigma$ are scaled by the same factor, then $E_1(N, \mu, \sigma)$ is also scaled by that factor. Table 1 illustrates two important points:

1. As the width of the probability function increases with the mean fixed, then $E_1(N, \mu, \sigma)$ decreases for a given value of $N$. This is to be expected since the wider the probability function the greater the chance of the smallest of $N$ samples lying far below the mean. When the probability function is narrow even a large value of $N$ results in only a small decrease in $E_1(N, \mu, \sigma)$ below the mean value.
2. $E_1(N, \mu, \sigma)$ falls relatively quickly for the first few values of $N$, but thereafter falls only slowly. For example, for $\sigma = 300$ the expected value of the first order statistic falls by over $40\%$ from the mean value for $N = 10$. However, using $N = 20$ service instances decreases the expected value by only an additional $7\%$.

Thus, Table 1 shows that for a narrow probability function no method of service selection will result in significantly faster performance when compared with random selection. In addition, for a wider probability function, when service selection based on perfor-

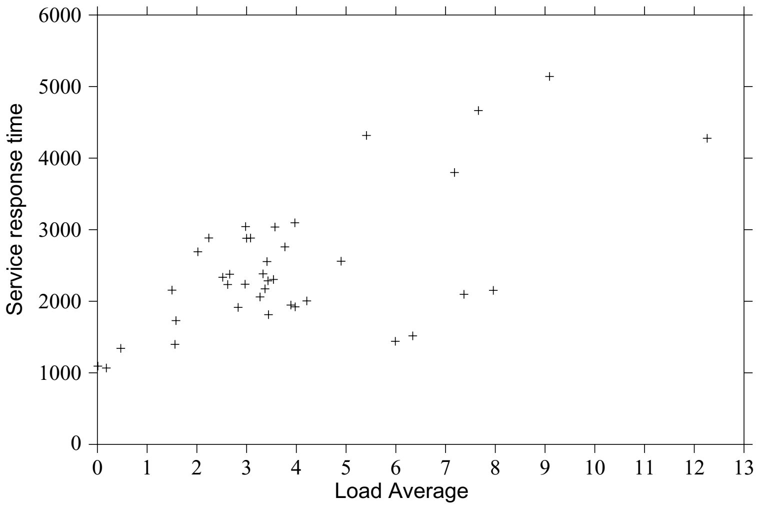

Fig. 6. Service response time in seconds as a function of the one-minute load average for Experiment 2.

mance prediction is likely to be worthwhile, adding more than a certain number of service hosts will not improve performance by very much, and so will not be cost-effective. The scalability of the service discovery and selection procedures is not, therefore, much of an issue, because unless the probability function is very broad only a modest number of service hosts (maybe 10 to 20) need to be used. However, in other circumstances the addition of extra service hosts may be worthwhile. For example, if the total amount of work performed per unit time by the set of service hosts is constant, then the addition of service hosts will reduce the mean service response time, $\mu$, and hence result in improved performance. Similarly, if multiple workflows are concurrently accessing the same set of equivalent services, adding more service hosts may result in improved throughput. This improvement arises because while a particular service instance is running the load on that host increases, so it is less likely that the same service instance will be selected by a different workflow for that period. Thus, the service selection mechanism ensures a degree of load balancing across the service hosts, and if there are more service hosts it is less likely that any particular one will be selected to concurrently execute a service instance for more than one workflow. It should be noted, however, that because the performance gain from dynamic selection is O($\sqrt{N}$) and the time to discover candidate services is O($N$), it is always true that if $N$ is sufficiently large dynamic selection will actually degrade performance.

The above discussion and analysis assumes that the service response times are identically and independently distributed for the $N$ service hosts, and that they follow the Gamma distribution. However, the following tight lower bound on the expected value of the first order statistic exists (see [1] and references therein):

$$E_1(N,\mu,\sigma) \geqslant \mu - \sigma\frac{(N-1)}{\sqrt{2N-1}} \tag{6}$$

This bound (usually referred to as the Hartley-David-Gumbel bound) holds for any distribution, and shows that the slow rate of decrease in the service response time observed for the Gamma distribution applies quite generally, since the bound decreases as $O(\sqrt{N})$. It is readily confirmed that the results in Table 1 obey the bound in Eq. (6).

Table 1
Expected value of the first order statistic for $\mu = 1000$

| $N$ | $\sigma = 50$ | $\sigma = 100$ | $\sigma = 200$ | $\sigma = 300$ |
|---|---|---|---|---|
| 1 | 1000.00 | 1000.00 | 1000.00 | 1000.00 |
| 2 | 971.80 | 943.65 | 887.73 | 832.64 |
| 3 | 957.93 | 916.40 | 835.25 | 757.17 |
| 4 | 949.01 | 899.02 | 802.45 | 710.97 |
| 5 | 942.54 | 886.51 | 779.13 | 678.60 |
| 6 | 937.51 | 876.83 | 761.28 | 654.11 |
| 7 | 933.43 | 868.99 | 746.96 | 634.64 |
| 8 | 930.01 | 862.45 | 735.08 | 618.61 |
| 9 | 927.07 | 856.85 | 724.97 | 605.06 |
| 10 | 924.51 | 851.98 | 716.21 | 593.39 |
| 11 | 922.24 | 847.67 | 708.50 | 583.17 |
| 12 | 920.21 | 843.82 | 701.64 | 574.11 |
| 13 | 918.37 | 840.34 | 695.46 | 566.00 |
| 14 | 916.69 | 837.17 | 689.85 | 558.66 |
| 15 | 915.15 | 834.26 | 684.73 | 551.98 |
| 16 | 913.73 | 831.58 | 680.02 | 545.86 |
| 17 | 912.41 | 829.10 | 675.66 | 540.21 |
| 18 | 911.18 | 826.79 | 671.62 | 534.98 |
| 19 | 910.03 | 824.62 | 667.84 | 530.17 |
| 20 | 908.95 | 822.59 | 664.30 | 525.57 |

## 8. Conclusions

This paper has presented a framework for dynamic workflow management in a service-oriented environment. Services capable of fulfilling a service request are discovered dynamically, and selection is undertaken using performance data. A Proxy Service is used that acts as an adaptor to support the dynamic service selection. In the first instance, semantically equivalent services are chosen, based on the use of a registry service (supported by extending UDDI). An Optimisation Service then selects the service with the lowest predicted response time, by accessing realtime performance data provided by a monitoring service. From the experimental results presented, it can be inferred that the strategy proposed here can improve workflow performance through the use of realtime performance data. A best case analysis, based on the first order statistic, shows how the improvement in service response time that results from dynamic service selection is related to the width of its probability function. For a sufficiently narrow probability function dynamic service selection is unlikely to result in better performance than selecting a service at random. The analysis also shows that as the number of service hosts increases the additional performance benefit becomes progressively less so that using more services is unlikely to be cost-effective.

## Acknowledgements

The research described in this paper was performed as part of the Workflow Optimisation Services for e-Science Applications (WOSE) project, funding for which was provided by the Core e-Science Programme of the UK's Engineering and Physical Sciences Research Council (grant reference GR/S24886/01), and is gratefully acknowledged. It is also a pleasure to acknowledge helpful discussions about bounds on the first order statistic, and related matters, with Professor Antonia Jones of the Cardiff School of Computer Science.

## References

[1] N. Balakrishnan, C. Charalambides and N. Papadatos, Bounds on Expectation of Order Statistics from a Finite Population, *Journal of Statistical Planning and Inference* **113** (2003), 569–588.

[2] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, Jignesh M. Patel, D. Reed, Z. Shi, O. Sievert, H. Xia and A. YarKhan, New Grid Scheduling and Rescheduling Methods in the GrADS Project, *International Journal of Parallel Programming* **33**(2) (June 2005), 209–229.

[3] C. Chapman, M. Musolesi, W. Emmerich and C. Mascolo, *Predictive Resource Scheduling in Computational Grids*, in Proceedings of the 21st International Parallel and Distributed Processing Symposium, pub. IEEE Computer Society Press, 2007.

[4] Y.-H. Chen-Burger, K.-Y. Hui, A.D. Preece, P.M.D. Gray and A. Tate, Workflow Collaboration with Constraint Solving Capabilities, *Expert Update* **8**(1) (2005), 48–60.

[5] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob and D.S. Katz, Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems, *Scientific Programming* **13**(3) (2005), 219–237.

[6] E. Deelman, T. Kosar, C. Kesselman and M. Livny, What Makes Workflows Work in an Opportunistic Environment? *Concurrency and Computation: Practice and Experience* **18**(10) (2006), 1187–1199.

[7] L. Huang, A. Akram, D.W. Walker, R.J. Allan, O.F. Rana and Y. Huang, *A Workflow Portal Supporting Multi-language Interoperation and Optimisation*, Concurrency and Computation: Practice and Experience (submitted December 2005, accepted February 2007).

[8] L. Huang, D.W. Walker, Y. Huang and O.F. Rana, *Dynamic Web Service Selection for Workflow Optimization*, in proceedings of the UK e-Science All Hands Meeting, 2005.

[9] M. Keidl and A. Kemper, *Towards Context-aware Adaptable Web Services*, in Proceedings of the 13th International World Wide Web Conference, pub. ACM Press, 2004, 55–65.

[10] T. Kosar and M. Livny, *Making Data Placement a First Class Citizen in the Grid*, in Proceedings of the 24th International Conference on Distributed Computing Systems, 2004, 342–349.

[11] Y. Liu, A.H. Ngu and L. Zeng, *QoS Computation and Policing in Dynamic Web Service Selection*, in Proceedings of the 13th International World Wide Web Conference, pub. ACM Press, 2004, 66–73.

[12] S. Ludwig, O.F. Rana, W. Naylor and J. Padget, Matchmaking Framework for Mathematical Web Services, *Journal of Grid Computing* **4**(1) (2006), 33–48.

[13] E.M. Maximillien and M.P. Singh, A Framework and Ontology for Dynamic Web Services Selection, *IEEE Internet Computing* **8**(5) (2004), 84–93.

[14] A.S. McGough, J. Cohen, J. Darlington, E. Katsiri, W. Lee, S. Panagiotidi and Y. Patel, An End-to-end Workflow Pipeline for Large-scale Grid Computing, *Journal of Grid Computing* **3**(3–4) (2005), 259–281.

[15] S. Miles, J. Papay, T. Payne, M. Luck and L. Moreau, Towards a Protocol for the Attachment of Metadata to Grid Service Descriptions and its Use in Semantic Discovery, *Scientific Programming* **12**(4) (2004).

[16] Y. Patel and J. Darlington, *A Novel Approach to Workload Allocation of QoS-constrained Workflow-based Jobs in a Utility Grid*, in Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, 2006.

[17] R. Prodan and T. Fahringer, *Dynamic Scheduling of Scientific Workflow Applications on the Grid: a Case Study*, in Proceed-

ings of the 2005 ACM Symposium on Applied Computing, pub. ACM Press, 2005, 687–694.

[18] D.A. Reed and C.L. Mendes, Intelligent Monitoring for Adaptation in Grid Applications, *Proceedings of the IEEE* **93**(2) (February 2005), 426–435.

[19] F.D. Sacerdoti, M.J. Katz, M.L. Massie and D. Culler, *Wide Area Cluster Monitoring with Ganglia*, in proceedings of the 2003 IEEE International Conference on Cluster Computing, 2003, 289–298.

[20] G. Singh, C. Kesselman and E. Deelman, Optimizing Grid-Based Workflow Execution, *Journal of Grid Computing* **3**(3–4) (2005), 201–219.

[21] D.P. Spooner, J. Cao, S.A. Jarvis, L. He and G.R. Nudd, Performance-aware Workflow Management for Grid Computing, *The Computer Journal* **48**(3) (2005), 347–357.

[22] N. Srinivasan, M. Paolucci and K. Sycara, Semantic Web Service Discovery in the OWL-S IDE in Proceedings of the 39th Annual Hawaii International Conference on System Sciences, 2006.

[23] W3C web site, OWL Web Ontology Language for Services, http://www.w3.org/Submission/2004/07/ (visited 10 April 2007).

[24] E.W. Weisstein, Order Statistic, from *MathWorld – A Wolfram Web Resource*. http://mathworld.wolfram.com/OrderStatistic.html.

[25] J. Yu and R. Buyya, A Taxonomy of Scientific Workflow Management Systems for Grid Computing, *Journal of Grid Computing* **3**(3–4) (September 2005), 171–200.

[26] M. Zangrilli and B. Lowekamp, *Transparent Optimization of Grid Server Selection with Real-Time Passive Network Measurements*, in proceedings of the Third International Workshop on Networks for Grid Applications (GridNets2006), 2006.

[27] M. Zangrilli and B. Lowekamp, *Using Passive Traces of Application Traffic in a Network Monitoring System*, in Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC13), 2004, 77–86.

[28] Z. Li and M. Parashar, *An Infrastructure for the Dynamic Composition of Grid Service*, Technical Report, Center for Advanced Information Processing, Rutgers University, April 2007. Available at http://www.caip.rutgers.edu/TASSL/Papers/rudder-tr-07.pdf.