The GridX1 computational Grid: from a set of service-specific protocols to a service-oriented approach

Gabriel Mateescu[1,3], Wayne Podaima[1]
Andre Charbonneau[1], Roger Impey[2]
Meera Viswanathan[1]
[1] Research Computing Support Group
[2] Institute for Information Technology
National Research Council Canada

Ashok Agarwal, Patrick Armstrong
Ron Desmarais, Ian Gable, Sergey Popov
Simon Ramage, Randall Sobie, Daniel C. Vanderster
Department of Physics and Astronomy
University of Victoria, Victoria BC
V8W 2Y2 Canada

Darcy Quesnel
CANARIE Inc.
Ottawa, Canada

Abstract

GridX1 is a computational Grid designed and built to link resources at a number of research institutions across Canada. Building upon the experience of designing, deploying and operating the first generation of GridX1, we have designed a second-generation, web-services-based, computational Grid. The second generation of GridX1 leverages the Web Services Resource Framework, implemented by the Globus Toolkit version 4. The value added by GridX1 includes metascheduling, file staging, resource registry and resource monitoring.

1. Introduction

GridX1 is a collaborative Grid computing project spearheaded by CANARIE (Canada's advanced Internet development organization), the University of Victoria and the National Research Council. GridX1 has established a computational grid infrastructure across Canada. The GridX1 project has been successful in enabling the execution of scientific computing applications such as the CERN ATLAS experiment [6], and the BaBar collaboration [4, 13], led by the Stanford Linear Accelerator Center. The results of executing these applications on GridX1 are detailed in [2].

The GridX1 project has produced two Grid implementations: (1) the first generation of GridX1 [1], based on a set of service-specific protocols and built upon version 2 of the Globus Toolkit; (2) the second generation of GridX1, which is the focus of this paper, based on a service-oriented approach and built upon version 4 of the Globus Toolkit.

This paper describes the techniques applied in the second generation of GridX1 middleware developed for achieving effective and reliable job submission, execution and monitoring. The paper is organized as follows: Section 2 describes the GridX1 architecture and its first-generation implementation. The limitations of the bags-of-services approach are pointed out in Section 3. Section 4 discusses the advantages of a web-services-based solution. The implementation of the second-generation of GridX1, including the web-services-based metascheduler, resource registry, and resource monitoring components, is described in Sections 5 and 6, while its early deployment is discussed in Section 7. Concluding remarks are made in Section 8.

2. First Generation of GridX1

The first generation of the GridX1 infrastructure has provided a production-grade Grid [1]. Its architecture is shown in Figure 1, and it incorporates the Globus Toolkit [8], the MyProxy credential management service [11], and the Condor-G job management system [9]. The Globus Toolkit provides the services of security (based on public key cryptography and X.509 certificates), resource management, information services, and data management. The MyProxy service supports the execution of jobs with long turn-around times, whose credentials would otherwise expire.

A Grid containing more than a few resources needs a *metascheduler* service. The Condor-G [9, 15] resource management system is used as the basic building block for

[3]Correspondence to: Gabriel Mateescu, National Research Council, Research Computing Support Group, 100 Sussex Drive, Ottawa, K1A 0R6 Canada. Email: gabriel.mateescu@nrc.gc.ca

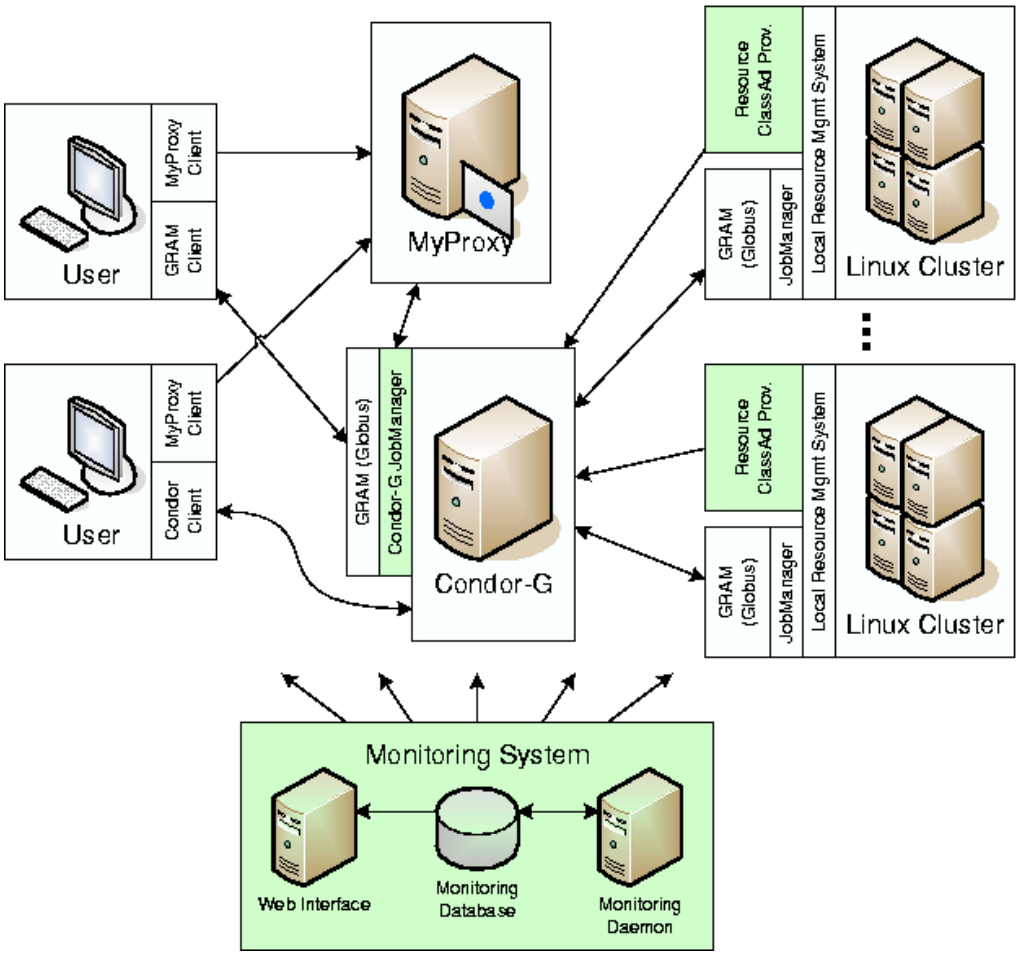

Figure 1. The GridX1 architecture: middleware developed in-house shown in gray boxes

metascheduling. Condor-G is the part of the Condor job scheduling system [14, 16, 5] that supports dispatching a Grid job to a Grid resource (e.g., cluster), which is accessed via the Globus job service. Condor-G is not an out-of-the-box metascheduler, but it provides a framework for mapping jobs to resources, based on matching job attributes with resource attributes. It includes a matchmaker and a resource registry; job submitters should set the job attributes, and information providers running on the Grid resources should publish resource information to the registry.

The GridX1 metascheduling architecture is shown in Figure 2. Users can submit jobs in two ways: (1) via the Grid Resource Allocation and Management (GRAM) interface to the Globus job execution service, or (2) via a Condor client. In the first case, a Globus job description is submitted, which is expressed in the *resource specification language*. The GRAM service and the GRAM-Condor adapter convert this description to a Condor job in the *globus* universe. In the second case, a Condor job description is submitted, with a job attribute to indicate to Condor that the job is in the *globus* universe. In both cases, jobs are persisted in a Condor queue and are mapped to a Grid resource using the Condor matchmaking [15] service.

The matchmaking service acts as a broker on behalf of a job, matching it to the most suitable resource. In Condor, job and resource attributes are expressed as ClassAds, with the attributes *Requirements* and *Rank* being used to define respectively constraints and preferences that the matched entity must meet. The Rank is a number that measures the suitability of a resource for a job: the higher the rank, the more suitable the resource is for a job. For every job that has to be mapped to a resource, the matchmaker performs two steps: (1) the job's Requirements attribute is evaluated to select the resources that meet the job's constraints; (2) the job's Rank attribute is evaluated for each of the resources selected at step (1), and the resource with the highest rank is selected for the job.

A web-based grid monitoring system was developed to

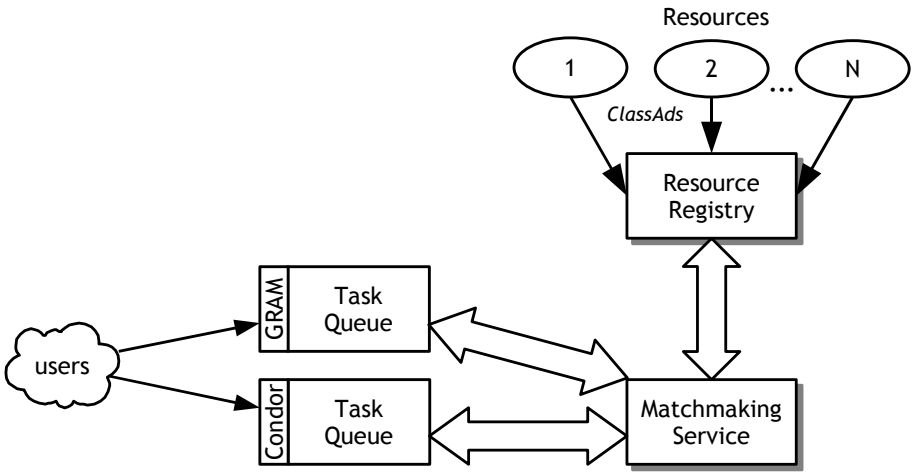

Figure 2. The GridX1 metascheduling architecture

allow users to track the status of resources and jobs, and to allow grid operators to detect and diagnose faults in the grid. Resource and job status information is obtained by querying the Condor system. Site functional testing is done using a daemon which periodically determines the status of the grid components, by performing an authentication test, a GRAM job submission test, and a GridFTP data transfer test. The results of the tests are archived in a MySQL database and a web interface presents the results to users.

3. Beyond the First Generation of GridX1

In the first generation of GridX1, Grid services were using service-specific protocols. For example, the job service used the Grid Resource Allocation and Management (GRAM), and the information service used the Light Directory Access Protocol (LDAP). There are several limitations of this approach, some of which are:

(i) difficult to extend: given that services use ad-hoc and evolving protocols, such as GRAM for the job service, adding a new service requires either modifying an existing protocol or adding a new protocol;

(ii) compatibility issues between different versions of the service and the clients: new versions of the service tend to use a protocol slightly modified from previous versions, which breaks backward compatibility with the old clients;

(iii) firewall problems: each service uses a TCP/IP port which must be open to all the client machines accessing that service. Because of the dynamic nature of the grid, managing host-based access control lists is cumbersome. Moreover, the requirement to open some ports may conflict with the local site policy.

(iv) security vulnerabilities: the job service runs as a privileged user, which opens the door to potentially compromising the system on which the service runs. This makes it even more problematic for the site firewall policy to allow wide access to the service, which in turn restricts the usefulness of the service.

The key limitation is that *each Grid service uses its own custom protocol*. This limitation has been realized by the Grid computing community. The solution has been to define a protocol framework to be used by the Grid middleware: the Web Services Resource Framework [10], which combines Web services with the associated resources.

Web services are building blocks for distributed applications. Two key features characterize Web Services: (1) their interface is described in a standard language, the Web Services Description Language (WSDL) [18]; (2) they are invoked using standard protocols: typically, SOAP [17] specifies the service invocation, and HTTP is used as the transport protocol for SOAP. WSDL describes the service interface, while XML schemas [19] describe the data types used by the interface. WSDL provides a standard mechanism to describe the interface of a Web service, and makes it possible to develop the machinery for building clients that are guaranteed to interface correctly with a certain service. This solves the long-standing issue of ensuring the correct interface between clients and services.

4. Second Generation of GridX1

The second generation of GridX1 is based on the Web Services Resource Framework (WSRF) technology [12] and its implementation in the Globus Toolkit version 4. WSRF solves the issues that impacted the first generation of GridX1 as follows:

(i) easy to extend: given that all services use SOAP-based web services, adding functionality can be done by creating a new WSRF service, whose interface is made known to clients using the WSDL service description;

(ii) seamless support of upgrades: if the interface of a service changes, the client can be easily upgraded, using the the new WSDL description of the service;

(iii) reduced firewall problems: Grid services use non-privileged TCP/IP ports, such as 8080, that most sites allow to be open;

(iv) good security: under the web-services approach, services are started by a service container which runs as a non-privileged user. A controlled method for executing jobs as another user is provided.

Unlike Web services, which are typically stateless, Grid services must often maintain persistent objects, such as jobs and user credentials. To manage this kind of objects, WSRF defines objects called *resources* which are composed of a number of *resource properties*, that are in turn managed (created, updated, queried, destroyed) by a Web service. A Web service combined with one or more resource properties is called a *WS-resource*. A single web service can manage multiple instances of the resource properties.

5. The Metascheduler Service

The metascheduler service receives a job description from a client and determines a suitable resource for the job to execute on. Unlike a *local scheduler*, a *metascheduler* does not have control over the resources to which it maps jobs. The local schedulers to which the jobs are ultimately dispatched do not always support *advance resource reservation* (which is defined as a set of resources reserved for a certain job for a well-defined time interval). Because of this, the metascheduler can only observe the status of the resources and the profile of pending jobs at a local scheduler, then make an estimate of when the job will be run at that local scheduler. Some metaschedulers, such as the VIOLA/Unicore metascheduling service [7], are designed to interact with local schedulers that support advance resource reservation, so the metascheduler will bind the job to a resource for a time interval. To make the metascheduler useful for the general case, we do not use advance reservation.

5.1. Metascheduling and Job Management

A metascheduler service can be exposed in two main ways, as shown in Figure 3: (1) as a service that receives a job description from a job submitter and returns a resource for the job; or (2) expose the metascheduler service as if it were a job execution service that also performs scheduling.

Under the first approach, the metascheduler is a standalone service that maps a job description to a resource name. This approach is used, for example, in the VIOLA metascheduling framework [7]. A disadvantage of this approach is that it requires a job submitter that is aware of the metascheduler service: upon receiving a job, the submitter first consults a metascheduler service from which it gets the resource for the job, then it invokes the job execution service on that resource. Another disadvantage is that it makes resource allocation more complex, because it breaks up the related operations of allocating resources for a job and running a job on the allocated resource, into two separate operations that are implemented by different modules: (1) resource allocation is done by the metascheduler; (2) job execution is done as a separate action by the job execution service, upon request from the job submitter. Assuming that in step (1) the metascheduler performs advance resource reservation, then were step (2) to fail for some reason, the metascheduler will not revoke the reservation unless the job submitter informs it about the failure.

The second approach is to expose the metascheduler service as if it were a job execution service. However, unlike a job service, which simply forwards the job to the local scheduler to which it is attached, a metascheduler exposed as a job execution service forwards the job to a remote resource that it determines to be most appropriate for the job. The advantage of this approach over the first approach is that the job submitter can treat the metascheduler as a regular job execution service.

On the other hand, the second approach to designing the metascheduler service requires it to handle file staging within the service. The job submitter cannot handle file staging, since the target grid resource is not known at submission time, but it is determined by the metascheduler after the job is submitted. In fact, the metascheduler service does not have to be actually involved in file staging; it only needs to make sure that the job description includes the appropriate file staging commands.

5.2. Enhanced Metascheduler service

In the second generation of GridX1, the metascheduling service has been improved to do more intelligent resource selection and to support file staging. Figure 4 shows the architecture of the enhanced metascheduler service. The new components, as compared to Figure 2, are the Resource ranking plugin, the Job history database, and the Condor-G job submission and file staging module; all but the latter component – which is part of the Condor distribution – have been developed in-house.

The resource ranking plugin is a callout module that the

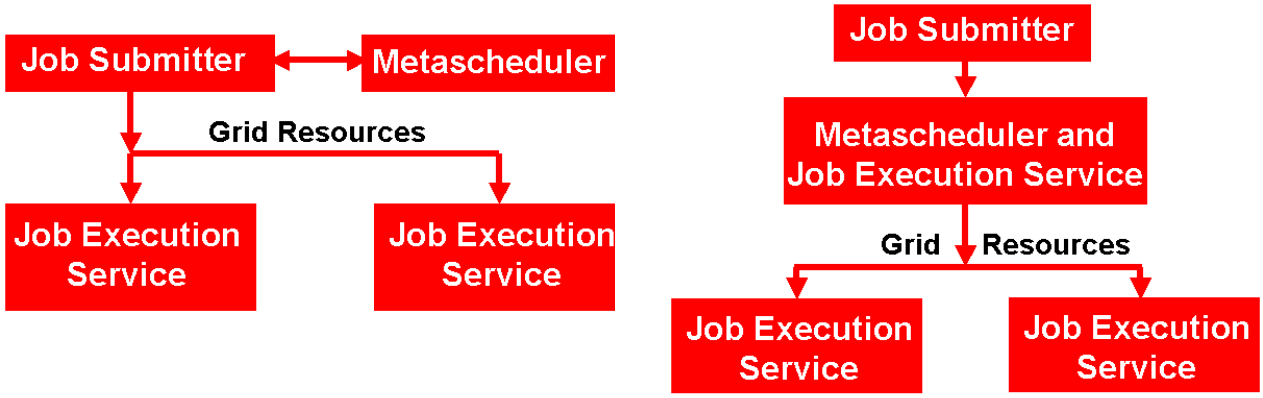

Figure 3. Exposing the Metascheduler as a stand-alone service (left) or as a job scheduling and execution service (right)

matchmaker invokes to compute the *Rank* of a resource for a job. With the plugin, we overcome the limitation of the matchmaking procedure described in Section 2. There, the Rank attribute was computed by the matchmaker, by considering each job-resource pair in turn, irrespective of the attributes of the other resources. So the Rank value was the result of a *pairwise* computation. This kind of computation could provide a global order of the resources for a job when the job is specifying a Rank expression that depends on *only one resource attribute*. However, if the job specifies a Rank expression that depends on *multiple resource attributes*, a pairwise computation does not provide an accurate global order of the resources.

The resource ranking plugin eliminates this problem by using global information about the resources while computing the rank. The global information is extracted from the *Resource Registry*. In addition, the plugin supports making the Rank value depend on the predicted turnaround time of the job. This time is in turn predicted using information about the previous jobs that have executed on a resource. This information is fetched by the resource ranking plugin from the job history database.

The Resource Registry module provides the matchmaker with static and dynamic information about the Grid resources, as described next.

5.3. Grid Resource Registry

The Resource Registry maps a resource name to a *resource classAd*, i.e., a set of static and dynamic attributes describing a resource. The resource registry implemented in the second generation of GridX1 is designed so that information can be propagated from the Grid resource to the matchmaker using WSRF to communicate between components running on different machines.

The resource attributes are made available to the matchmaker using the following design:

(i) On each Grid resource, a *resource information provider* inserts the resource information – in the form of XML-formatted condor classAds – into the local *WSRF Monitoring and Directory Service* (WS-MDS, for short);

(ii) The WS-MDS running on the Grid resource registers with the replicated GridX1-level WS-MDS (replication is used to achieve high availability of the GridX1-level WS-MDS);

(iii) On the matchmaker machine, a WS-MDS client queries the GridX1-level WS-MDS and extracts information about all resources that advertised their attributes to WS-MDS;

(iv) The XML-formatted resource information is translated to native Condor classAd format and is pushed into the Condor internal information service maintained by the Condor *collector* daemon.

Unlike the first generation of GridX1, where Grid resources were advertising their status directly to the Condor information service, this scheme uses Web services for

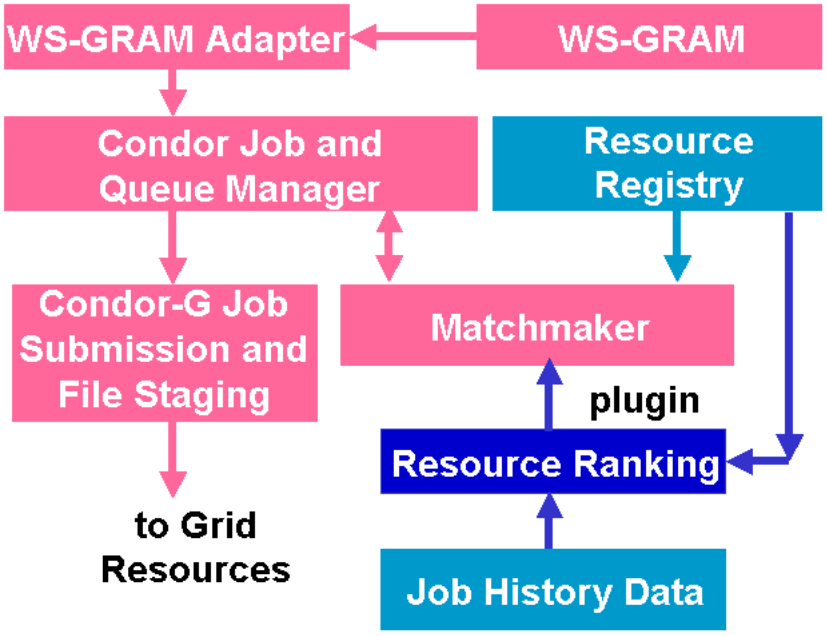

Figure 4. The new GridX1 metascheduling architecture

inter-host communication. This way, propagating the resource information only requires access to the WS-MDS service, unlike the previous method which required access to the TCP/IP port used by the Condor information service on the matchmaker machine; this port had to be open to all Grid resources, which complicated firewall management and created potential vulnerabilities.

The chain of WSRF services and clients described above, which is used to assemble the resource information, is a good example of the power of service-oriented architectures such as GridX1.

6. Grid Monitoring

The first generation GridX1 monitoring system has been useful to grid users and operators alike [1]. By monitoring the basic grid functionalities, such as the job execution service (GRAM) and the data transfer service (GridFTP), problematic clusters were easily identified and they could be removed from the set of available resources, thereby ensuring the reliability of the Grid. A similar system is required for the second generation of GridX1.

Because the new GridX1 metascheduling system is still built around Condor, the job and resource monitoring components from the first generation system can be reused. In particular, a web interface to the Condor tools for determining the resource status and job information is easily ported to the new system. However, for the site functionality testing, a new system needed to be built, one that uses the Globus Toolkit 4 API. This system has been developed to monitor the service availability and functionality, using periodic queries. For example, the system can ping sites and can also test if the service container is listening by attempting to connect to the TCP port 8443. If a service passes the availability test, a functionality test can be run, which attempts to use the service. Functionality tests include credential delegation, WS-GRAM job submission, and file transfer using the Reliable File Transfer (RFT) service. Finally, the monitoring system exercises all of the GridX1 services by submitting a job to the metascheduler service and observing the job as it uses Grid services. A web-interface allows grid operators to not only observe the results of the service tests, but also provides a mechanism to add, delete, and modify the tests to be performed.

7. Testbed Deployment

A prototype of the service-oriented second generation of GridX1 has been deployed in a national testbed using resources at the National Research Council in Ottawa and the University of Victoria. The testbed is composed of a small number of Linux clusters, each of which uses the Portable Batch System [3] as its local resource management system (LRMS). The LRMS of each cluster is accessible as

a WS-GRAM service provided by the Globus Toolkit version 4. Each cluster advertises its resource information (formatted as an XML ClassAd) to a redundant pair of WS-MDS registry servers, one at each institution. Similarly, the metascheduler, implemented as described in section 5.1, is replicated at the two institutions to improve reliability and scalability. Being exposed as a WS-GRAM service, the metascheduler accepts jobs from standard Globus Toolkit clients, matches the jobs to a cluster, and performs the file staging and job dispatching to the selected cluster. The job load is distributed evenly between the replicated metaschedulers and registries using a round-robin DNS approach.

At present, the testbed has been fully deployed and the registry and metascheduler functionalities have been verified using simple jobs which exercise the matchmaking, job dispatching, and input/output file-staging. Work is underway to thoroughly evaluate the performance of the presented grid services and the reliability achieved using our service-replication strategy.

8. Conclusions

We have presented the architecture of the GridX1 computational grid, which has evolved from a bag-of-services technology to a WSRF-based architecture. Within the framework set out by WSRF, we have developed a metascheduling service, along with information providers and a client that extracts resource status from the Index service and makes it available to the metascheduler.

The WSRF-based design of GridX1 provides a robust and flexible Grid solution. A WSRF-based testbed Grid has been deployed, with initial results showing that the approach is viable.

The results of the work described here have been submitted as a Globus Incubator Project (*Gavia* Metascheduler), and will be available for distribution.

Acknowledgements

The authors acknowledge the support of CANARIE, the National Sciences and Engineering Research Council, and the National Research Council of Canada. Support from CANARIE has been given under the CANARIE Intelligent Infrastructure Program.

References

[1] A. Agarwal. *et al.* GridX1: A Canadian Computational Grid. *Future Generation Computer Systems*. to appear.

[2] A. Agarwal, M. Ahmed, B. Caron, A. Dimopoulos, L. Groer, R. Haria, R. Impey, L. Klektau, C. Lindsay, G. Mateescu, Q. Matthews, A. Norton, D. Quesnel, R. Simmonds, R. Sobie, B. S. Arnaud, D. Vanderster, M. Vetterli, R. Walker, and M. Yuen. GridX1: A Canadian Particle Physics Grid. In *Proceedings of Computing in High Energy Physics 2006, Mumbai, India*, February 2006.

[3] Altair Engineering. PBS Professional Documentation. http://www.altair.com/software/pbspro.htm, 2006.

[4] B. Aubert. The BaBar Detector. *Nuclear Instruments and Methods in Physics Research Section A*, 479(1):1–116, February 2002.

[5] J. Basney, M. Livny, and P. Mazzanti. Utilizing widely distributed computational resources efficiently with execution domains. *Computer Physics Communications*, 140, 2001.

[6] CERN. The ATLAS Data Challenge Experiment. http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/DC/, 2005.

[7] K. Cristiano, R. Gruber, V. Keller, P. Kuonen, S. Maffioletti, N. Nellari, M.-C. Sawley, M. Spada, T.-M. Tran, O. Wäldrich, P. Wieder, and W. Ziegler. Integration of ISS into the VIOLA meta-scheduling environment. In S. Gorlatch and M. Danelutto, editors, *Proceedings of the Integrated Research in Grid Computing Workshop*, pages 357–366. Università di Pisa, November 2005.

[8] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[9] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, Aug. 2001.

[10] J. Gawor, M. Humphrey, K. Jackson, S. Lang, and S. Meder. Options for building WSRF compliant services. In *GlobusWorld*, 2005.

[11] J. Novotny, S. Tuecke, and V. Welch. An online credential repository for the grid: Myproxy. In *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*. IEEE Press, August 2001.

[12] OASIS. Web services resource framework (WSRF). http://www.oasis-open.org/committees/wsrf, 2006.

[13] SLAC. The BaBar Workbook documentation. http://www.slac.stanford.edu/BFROOT/www/doc/, 2006.

[14] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor – A Distributed Job Scheduler. In T. Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, 2001.

[15] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., March 2003.

[16] D. Thain, T. Tannenbaum, and M. Livny. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, 17(2–4):323–356, 2005.

[17] W3C. SOAP version 1.2. http://www.w3.org/TR/soap12-part1, 2006.

[18] W3C. The web services description language (WSDL). http://www.w3.org/2002/ws/desc, 2006.

[19] W3C. XML schema. http://www.w3.org/XML/Schema, 2006.