BaBar MC Production on the Canadian Grid using a Web Services Approach

A Agarwal[1], P Armstrong[1], R Desmarais[1], I Gable[1], S Popov[1], S Ramage[1], S Schaffer[1], C Sobie[1], R Sobie[1], T Sulivan[1], D Vanderster[1] G Mateescu[2], W Podaima[2], A Charbonneau[2], R Impey[2], M Viswanathan[2], D Quesnel[3]

[1] Department of Physics and Astronomy, University of Victoria, Victoria BC, V8W 2Y2 Canada

[2] Research Computing Support Group, National Research Council Canada, Ottawa, Canada

[3] CANARIE Inc., Ottawa, Canada

E-mail: `aka@uvic.ca`

Abstract. The present paper highlights the approach used to design and implement a web services based BaBar Monte Carlo (MC) production grid using Globus Toolkit version 4. The grid integrates the resources of two clusters at the University of Victoria, using the ClassAd mechanism provided by the Condor-G metascheduler. Each cluster uses the Portable Batch System (PBS) as its local resource management system (LRMS). Resource brokering is provided by the Condor matchmaking process, whereby the job and resource attributes are expressed as ClassAds. The important features of the grid are automatic registering of resource ClassAds to the central registry, ClassAds extraction from the registry to the metascheduler for matchmaking, and the incorporation of input/output file staging. Web-based monitoring is employed to track the status of grid resources and the jobs for an efficient operation of the grid. The performance of this new grid for BaBar jobs, and the existing Canadian computational grid (GridX1) based on Globus Toolkit version 2 is found to be consistent.

1. Introduction

Grid Computing in high energy physics applications is becoming an important and necessary tool to meet the large computational requirements to analyze data and run simulations. Various grid projects such as GridX1 [1, 2], Tera Grid [4], the Open Science Grid [5], the LHC Compute Grid [6], NorduGrid [7] and INFN Grid [8] have already taken part in grid development and deployment. Most of these Grids are based on the grid middleware provided by the Globus Toolkit [9, 10], and also distributed by the Virtual Data Toolkit (VDT) [11].

Recently, GridX1, a Canadian computation Grid, based on Globus Toolkit version 2 and the Condor-G resource broker (RB) [12], has been very successful in running the BaBar and Atlas simulations [1–3]. This grid combines the resources of various Canadian Universities (Alberta, Toronto, McGill, British Columbia, Victoria) and the Canadian research institutions (National Research Council in Ottawa). Though the total number of processors at these sites are about 2500 with disk and tape storage well in excess of 100 TB, GridX1 is given access to a fraction of these resources, with some sites allocating a specific job quota, and others allowing GridX1 to backfill during periods of low utilization. Condor-G provides the resource management and

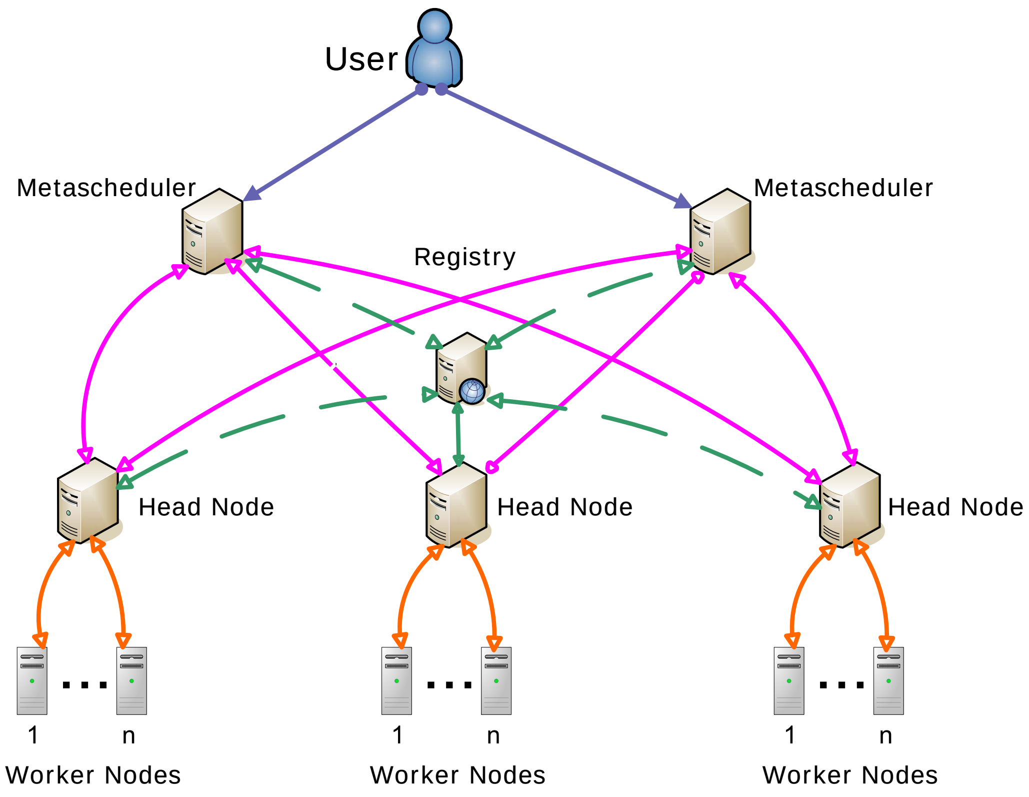


Figure 1. The proposed WSRF-based grid consisting of Condor-G metaschedulers, the central resources registry and the resources

metascheduling across the clusters at all sites and it is a part of the Condor job scheduling system ([13–15]). Cluster resources publish ClassAds to a Condor collector periodically through a cron job and the Condor negotiator matches the task and resource ClassAds to choose appropriate resources. A central web-based grid monitoring system is deployed to easily track the status of resources and their jobs which improves the reliability and the efficiency of the grid [2].

Though the usage of GridX1 is successful and efficient in running applications, Globus Toolkit version 2 does not utilize web services, a limitation that has been realized by the Grid computing community. The key limitation of the non-web services approach is that each grid service uses its own service-specific protocols such as Grid Resource Allocation and Management (GRAM), MyProxy [16], Light Directory Access Protocol (LDAP) and Condor. These protocols suffer various limitations: (i) difficult to extend, (ii) compatibility issues due to a lack of protocol standardization, (iii) firewall problems as each service uses its own TCP port and (iv) security vulnerabilities as the GRAM service runs as root, which could be compromised [17]. To avoid these problems, the Globus community has implemented the Web Services Resource Framework (WSRF) [18] in Globus Toolkit version 4. This uses a common protocol (SOAP) [19] to easily accommodate the new WSRF services, and changes to service interfaces can be described in the standard Web Services Description Language (WSDL) [20]. Further, it provides reduced firewall problems as the service container has fewer non-privileged ports and good security as the service container runs as a non-privileged user rather than root.

The aim of the present paper is to build and describe a Canadian computational grid using the WSRF provided by Globus Toolkit version 4. Its basic structure is shown in Figure 1. The main components of the grid are the design and implementation of a metascheduler service around Condor-G and the set up of a central resource registry. Other important features of the grid are automatic registering of resource information to the central registry, ClassAds extraction from the registry to the metascheduler for matchmaking, and the incorporation of input/output file staging. Most of the software developed for this grid is Java based code as a part of a CANARIE (Canada's advanced Internet development organization) funded project instituted in the year 2006. This project was spearheaded by CANARIE, the University of Victoria and the National Research Council in Ottawa. The project's metascheduler service [21] and job submission client [22] components are now being developed further as the Gavia project, which is part of the Globus Incubator programme for new grid technologies. Using these software components, a pre-production grid test bed is built by integrating the resources of two clusters at the University of Victoria. On this grid, the BaBar MC Production is set up and tested. The grid resources use the Portable Batch System (PBS) [23] as the local scheduler. Web-based monitoring is also employed to track the status of grid resources and the grid jobs for the reliable and efficient operation of the computational grid.

2. Metascheduler Design and Implementation

The metascheduler service is built around Condor-G. Though it is not an out-of-the-box metascheduler, Condor-G provides a framework for mapping jobs to resources. It includes a matchmaker (Negotiator) and a resource registry (Collector). Job submitters should set the job attributes, and information providers running on the Grid resources should publish resource information to the resource registry. To build an effective metascheduler Grid service based on Condor, one must accomplish these tasks: (1) expose the Condor job management system as a WSRF service; (2) add matchmaking to the Condor job management system; (3) perform intelligent resource selection.

Figure 2 shows the architecture of the metascheduler service. The Resource ranking plugin and the Job history database are used to accomplish task (3). The next three subsections describe how we accomplished these tasks.

2.1. Expose the Condor job management system as a WSRF service

To accomplish task (1) we have extended the functionality provided by the ManagedExecutable-JobService supplied by Globus 4. This service is part of the WS-GRAM component of Globus 4 and allows one to submit jobs to a Grid resource. Globus supports several types of Grid resources, including Condor pools. The Condor factory type supported by Globus will execute a job on the Condor pool attached to the WS-GRAM service, rather than select another Grid resource and dispatch the job to that resource. However, our goal is to use Condor as a metascheduler and job submission (i.e., as Condor-G) not as a Condor pool. In other words, the Condor factory type supported by Globus does not support the grid universe of Condor.

We have extended the Globus Condor factory type to support the grid universe. To accomplish this, we have extended the Condor scheduler (Condor-G) adapter component of WS-GRAM. The Condor-G adapter is the software layer that is placed between the WS-GRAM job execution service and the local scheduler (e.g., Condor) and accomplishes two main tasks: (1) converts a WS-GRAM job description to a scheduler-specific (e.g., Condor-specific) job description, and (2) submits the job to the local scheduler (e.g., Condor). When a client submits a job, WS-GRAM passes the job description to the Condor-G adapter, which converts it to a Condor job description, with the grid universe attribute, and submits it to Condor. When Condor receives the job, instead of executing it on a Condor pool, it performs matchmaking to select a remote Grid resource, and then submits the job to the selected resource (see Figure 2).

Therefore, we have empowered the Condor job execution service to be a job metascheduling and submission service, rather than an interface to a Condor pool. To handle file staging between

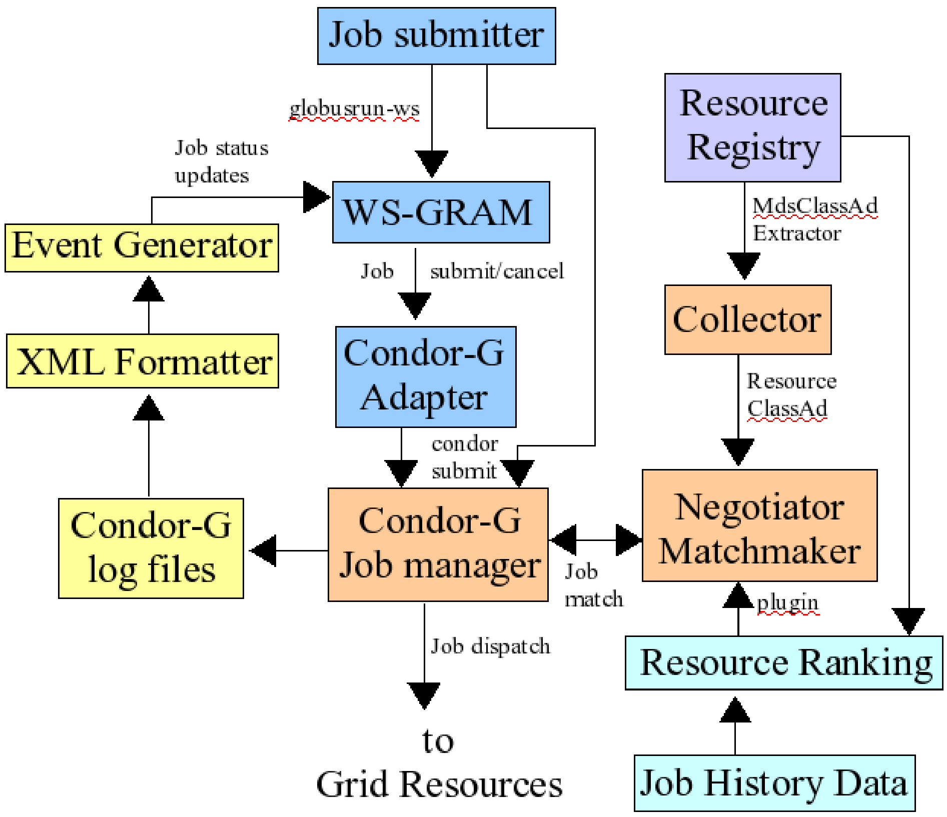


Figure 2. Metascheduler Block Diagram

the metascheduler and the Grid resource, our extended scheduler adapter converts Globus file staging directives to Condor file transfer commands. Note that the user can also create the Condor-specific job description file and submit it directly to the Condor-G job manager as shown in Figure 2. The other part of matchmaking is based on the job attributes. In addition to the job attributes specified by the user, we have added a Rank job attribute which is essential in performing intelligent resource selection.

2.2. Adding matchmaking to the Condor job management system

For metascheduling, we use the Condor matchmaking mechanism, which selects a resource for a job by matching the job ClassAd with the resource ClassAd. So, in order to perform matchmaking, Condor needs to have information about the available Grid resources.

In a Condor pool, the central manager gets resource information directly from the compute nodes using Condor-specific, non web-services protocols. On the other hand, for the Condor-G based metascheduler we need to generate Grid resource information rather than execution node information, and we need to communicate this information to the Condor Collector using web services and clients rather than Condor-specific protocols. For this purpose, a Java-based MDS client program (called the MdsClassAdExtractor) has been developed by us and runs on the metascheduler machine; this program extracts the resource ClassAd from the central resource registry and sends it to the Condor Collector (Figure 2). The setup of the central resource registry and the MdsClassAdExtractor is described in detail in Sections 3.1 and 3.2, respectively.

2.3. Intelligent resource selection

Two important ClassAd attributes used in Condor matchmaking are *Requirements* and *Rank*. Requirements is a boolean expression that defines the constraints that the job imposes on the resource. The Rank attribute defined by the job is a real number that defines the preferences that the job has about the resources. For a given job j, Condor will compute the ranks $Rank(j, r)$ for all the resources $r \in R$, where R is the set of the resources from the registry for which the Requirements attribute evaluates to true. Then Condor will match the job with the resource with the highest value of the Rank. Another important feature in the Condor matchmaking is the use of external ranking capability, which allows one to write the Rank expression as $Rank = external\_rank(script; TARGET.Name;)$, where *external_rank* is a callout function defined in a dynamic shared object loaded by Condor at run-time. This callout is the mechanism by which we can invoke an external program, e.g., *script* to compute the Rank, and pass this program arguments such as the name of the resource (TARGET.Name). This feature has been used in GridX1 to reduce the ATLAS job failures due to network problems while reading input data from a storage element [2].

Though the Condor matchmaking has many advantages mentioned above, we noticed that if the Rank expression depends on multiple attributes of the resource, a job-resource pairwise computation does not provide a correct global order of the resources. Hence it does not select the best resource. To see this, assume that we want to order the resources by two resource attributes such as number of cpus and estimated waitingtime, $a1$ and $a2$. If we were to sort only by $a1$ (or $a2$) then we could write $Rank = a1$ (or $Rank = a2$). However, to sort by both of these attributes with $a1$ as the first key and $a2$ as the second key, we must know the maximum value of $a2$, call it $A2$. If we knew it, we could write $Rank = a1 \times A2 + a2$ to get a multiple key ordering. However, the matchmaking mechanism has neither the capability to "store" the value of $A2$ for use in the Rank expression, nor can the Rank expression depend on job-specific functions such as $A2 = max_{r \in R}(a2)$. To solve this problem and achieve intelligent resource selection, we employ a solution that contains two components: (1) we persist the resource attributes in a database; (2) we use the Condor external ranking capability by utilizing an external program in the ranking expression as described above. The external program will extract any number of resource attributes from the database and compute a Rank value that is an arbitrary function of these attributes. The database is used to obtain fast access to the resource attribute values (as opposed to extracting them directly from the registry). Furthermore, with the database we can quickly compute functions such as the maximum value of an attribute. The information is inserted in the database by a background process. Therefore, using external ranking and a database we achieve a dual goal: (1) support correct ordering of resources when the rank expression depends on multiple attributes of a resource; (2) make the computation of the Rank expression fast.

Another feature we used for intelligent resource selection is to predict the waiting time of the job on the candidate resource and include this time in the Rank expression. The prediction is based on keeping a record for each resource of the waiting times of the previous jobs submitted to that resource. This information is kept in the database in the job history table. The external program can then simply fetch this information from the job history table and include it in the computation of the Rank as shown in Figure 2.

3. Registry Design and Implementation

The registry is designed to keep information about the resource attributes, metascheduler, reliable file transfer service (RFT) and the local resource management services (LRMS). The 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 here is designed so that information can be propagated from the Grid resource to the matchmaker using web services to communicate

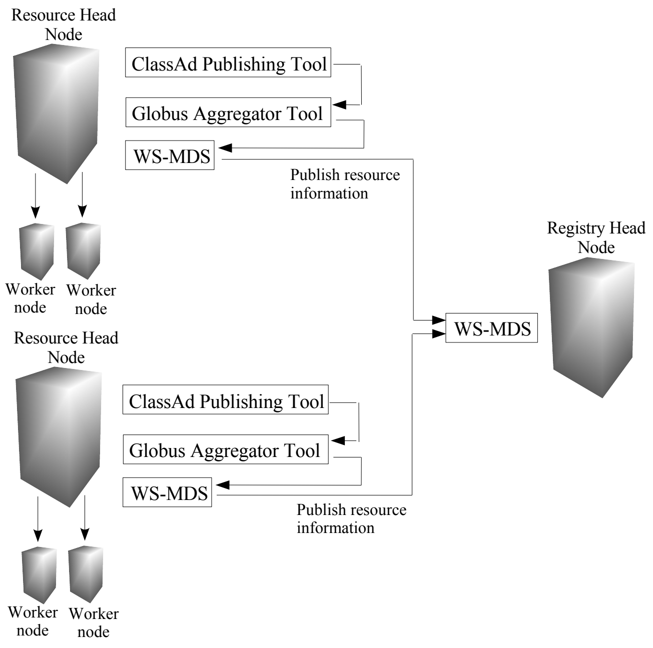


Figure 3. The ClassAd Publishing Tool

between components running on different machines. The resource attributes are made available to the Condor matchmaker on the metascheduler using the following design:

(i) An information provider script is written, which runs on every Grid resource and generates the resource ClassAd in a GLUE 1.2 XML schema. The GLUE XML schema is designed to model grid resources [24].

(ii) On each Grid resource, a resource information provider inserts the resource information in the form of XML-formatted Condor ClassAds into the local WS-MDS and also publishes the resource information into the registry (discussed in detail in section 3.1).

(iii) On the metascheduler machine, a MdsClassAdExtractor tool is designed to extract the XML-formatted information about the resources that advertise their attributes to WS-MDS (discussed in detail in section 3.2).

Unlike the Globus Toolkit version 2, where Grid resources advertised their status directly to the Condor information service, this scheme uses Web services for inter-host communication. This way, propagating the resource information only requires access to the WS-MDS service.

3.1. ClassAd publisher on the resources

Since resource information is needed for job matchmaking on a periodic basis, a ClassAd publishing tool is designed and implemented to automate this process. This process is shown in Figure 3. The tool works with the local job schedulers (PBS, Condor) to extract the cluster information such as the total number of cpus available, the number of free cpus, memory and the number of jobs in the queue etc. This information is converted into the XML format (GLUE

1.2 schema) and aggregated into its local WS-MDS by using the Globus aggregator tool. This information is then upstreamed to the registry where the information from all the resources are gathered. The MDS Index registration mechanism is used which allows one to register an information provider with the MDS index service.

3.2. ClassAd extractor tool on the metascheduler

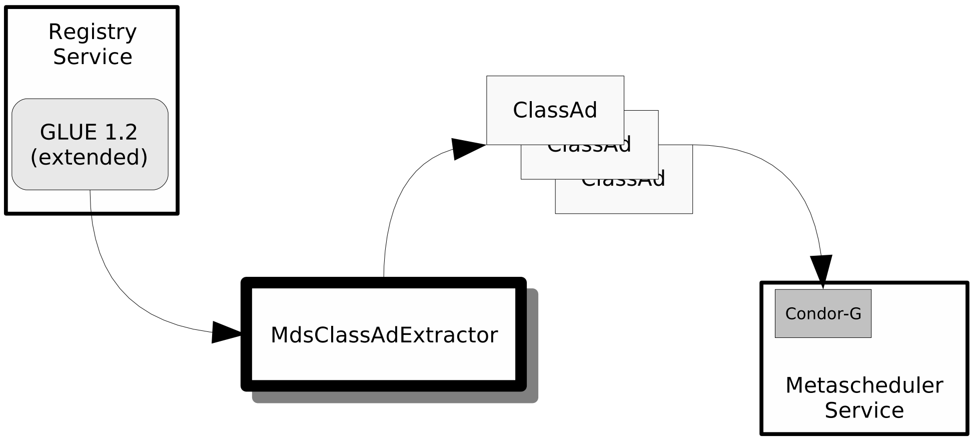


Figure 4. The MdsClassAdExtractor Tool

Information about the compute resources of the grid is stored in the information registries (Globus MDS) using an extended version of the GLUE 1.2 XML schema. Each compute resource has an associated XML document in the central registry service. These XML documents describe each compute resource's capabilities, configuration and availability. Because the metascheduler is implemented using Condor-G, which has its own information registry, the compute resource information needs to be periodically extracted from the Globus central registry service and published to the Condor collector. This information will then be used by Condor during the matchmaking process to assign jobs to resources. But Condor works with ClassAds and is unable to process information that is in the GLUE 1.2 XML format. As a result of this, a tool is needed to not only extract the information from the registry, but also to convert this data from GLUE 1.2 XML to Condor ClassAds and publish it to the Condor collector.

For this purpose, the MdsClassAdExtractor tool as shown in Figure 4 has been implemented. MdsClassAdExtractor is a Java application which queries a Globus MDS to extract the compute resources's GLUE 1.2 information, converts this XML data to Condor ClassAds for all resources and then publishes these ClassAds to the Condor collector used by the metascheduler. This software runs on each of the metaschedulers on a periodic basis and extracts the data from the MDS using the Globus Java libraries. The publishing of ClassAds to the Condor collector is done using the Condor command-line utilities. Since the compute resource information in the Globus MDS registry has been formatted in XML, XSLT is chosen as the means of data conversion between the GLUE 1.2 XML format and Condor ClassAds. Using XSLT to implement the data conversion decouples the transformation process from the rest of the application – making it more flexible and easier to adapt to changes in either the XML schema or the ClassAd definitions. This XSLT transformation process is embedded in the MdsClassAdExtractor tool and is implemented using Xalan-Java.

4. Setup of BaBar Monte Carlo Production on Canadian grid testbed

A pre-production WSRF based Canadian grid testbed has been deployed to test out the approach and the software discussed above. On this grid, the implementation and operation of a real high energy physics application such as BaBar Monte Carlo Production is highlighted. A schematic illustration of this set up is shown in Figure 5.

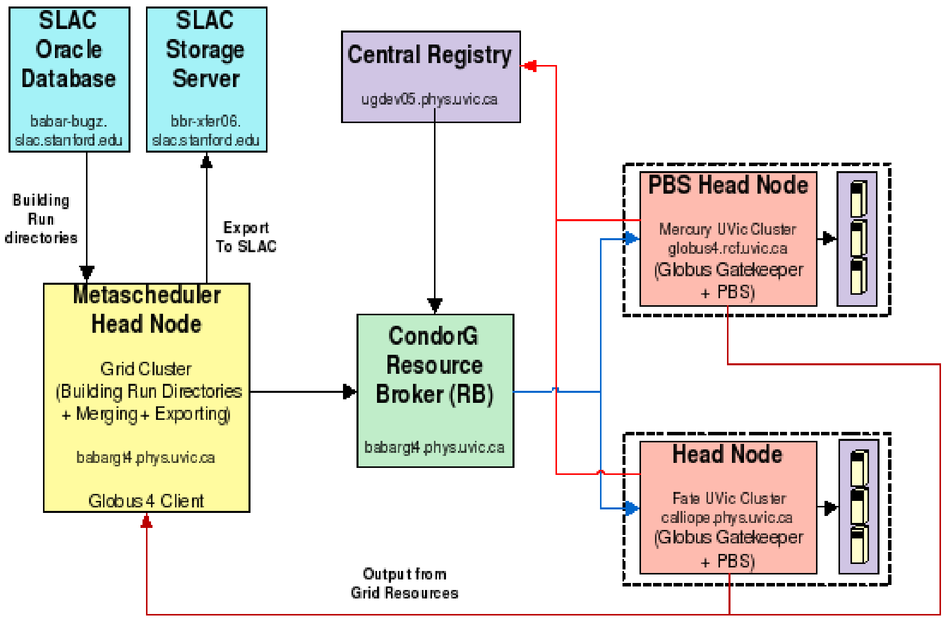


Figure 5. BaBar MC Production set up using the Condor-G Resource Broker

The grid testbed consists of a metascheduler head node, central registry head node and the two grid resources, Fate and Mercury Linux clusters, situated at the University of Victoria. The primary requirement is the installation of Globus Toolkit version 4 on all the machines including worker nodes. On the metascheduler, Condor is installed and configured to accept and run the Condor jobs. The Condor-G adapter software to expose the Condor job management system as a WSRF service (discussed in section 2.1 and shown in Figure 2) and the MdsClassAdExtractor software to extract the resource ClassAds for job matchmaking to the Condor job management system (discussed in section 3.2 and shown in Figure 4) have also been installed. The Central registry machine needs the local WS-MDS to be set up properly so that it can store the information about the resource attributes, metaschedulers, RFT and LRMS. An information provider script on both Fate and Mercury resources is installed which will publish information in the GLUE CE 1.2 schema, inserts into local WS-MDS and publishes the resource information into the central registry (as discussed in section 3.1). The resources use PBS as the local scheduler and they are accessible as a WS-GRAM service provided by Globus Toolkit version 4. The BaBar jobs can be submitted to the metascheduler either as a WS-GRAM service or as a Condor-G submit client. The metascheduler accepts the jobs, selects the best resource by matching the job and resource ClassAds, and performs file staging and job dispatching to the selected resource.

Before running the BaBar Monte Carlo Production on this grid, it is necessary to make sure that the BaBar software is installed on the metascheduler head node and the local BaBar Production on both Fate and Mercury clusters is set up and is fully functional. The metascheduler is the head node of the grid resources where BaBar jobs are built, submitted, merged and exported to SLAC (see Figure 5). To run BaBar Production through the web-services based grid, two directories are created on the metascheduler (i) outgoing and (ii) incoming. As usual, the run directories are built from SLAC using the spbuild script supplied by SLAC and the run directories are tarred up and placed in the outgoing directory. The BaBar

Table 1. Performance tests.

Sr. No.	SP8 Validation	Globus Toolkit 4				Globus Toolkit 2			
	Run Number	fate		mercury		fate		mercury	
		CPU Time (HH:MM)	% Eff	CPU Time (HH:MM)	% Eff	CPU Time (HH:MM)	% Eff	CPU Time (HH:MM)	% Eff
1	9941582	4:26	99	4:59	85	4:23	99	4:06	98
2	9941585	4:24	99	5:01	80	4:40	98	4:08	98
3	9941587	4:31	98	4:55	84	4:26	99	4:14	97
4	9941589	4:09	98	4:40	85	4:41	98	4:20	97

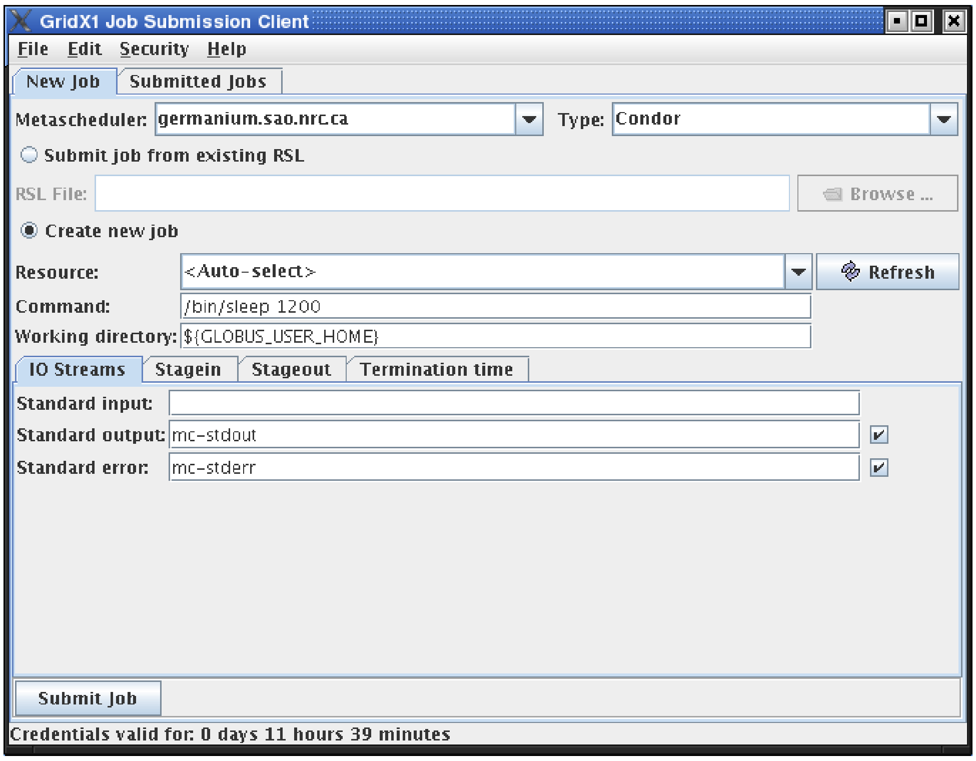
jobs are then submitted to the metascheduler either as a WS-GRAM or Condor job submission. The job ClassAds are matched with the resource ClassAds by the Condor matchmaking process depending on the rank and requirement of the job ClassAds and the best resource is selected. On the selected resource, the tarred up run directories on the metascheduler are transferred to the resource using the globus-url-copy, untarred and executed on it. After completion of the BaBar job, the output run directories are tarred up again and brought back to the submission metascheduler head node in the incoming directory using globus-url-copy. An automatic script running periodically as a cron job on the metascheduler head node keeps untarring the output run directories and puts them back in their usual place to the allrun directory used by SLAC for merging and exporting. In this way, all the grid resources are unified to look like a single combined resource controlled by the metascheduler head node. Merging and exporting is done in its usual way on the head node like local production and using the same spmerge and spexport scripts supplied by SLAC (see Figure 5). The running of the BaBar jobs for the SP8 validation runs on this web-based grid is found to be successful and as efficient as on the existing Canadian grid (GridX1) based on Globus Toolkit version 2. The histograms for the SP8 validation runs compare very well with the corresponding runs at SLAC giving credence to the results.

Table 1 shows the comparison of the CPU time and the percent efficiency (ratio of CPU time/WallTime) for the few SP8 validation runs used in the production at SLAC. It is evident that the results are quite stable and consistent between this new web-based grid (Globus Toolkit version 4) and the existing Canadian production grid GridX1 (Globus Toolkit version 2). There is a slight loss of percentage efficiency on the mercury cluster for the Globus Toolkit version 4. This is attributed to network congestion since mercury is a busy shared cluster. This new grid set up will be used for running the production in future.

5. Job Submission Tool and Grid Monitoring

Grid toolkits, such as Globus 4, already provide command-line tools to submit, manage and monitor jobs on a grid. Although these tools usually provide all the features and functionality needed, often they can be error prone and difficult to use for the novice user. Also, the typical use of the grid will only require a subset of the numerous features provided by these command-line tools, making their interface seem overly complex to some users. In view of this, the graphical user interface (GUI) for the BaBar application is designed and implemented by using the most commonly used features in the Globus 4 command-line tool. This is shown in Figure 6. With this GUI, the user is able to create grid credentials, discover available compute resources and metaschedulers through the registry, submit jobs to the grid resources by selecting the metascheduler and then add, delete and monitor the jobs status. Having an easy to use graphical interface to a subset of the most commonly used features found in a grid toolkit not only makes the grid more accessible to the end user, but also renders the job submission and monitoring processes less error prone.

Note that this GUI uses the Gavia Job Submission and Management (JSAM) library. Acting



Metascheduler: germanium.sao.nrc.ca
Type: Condor
Submit job from existing RSL
RSL File:
[✓ Create new job]
Resource: <Auto-select>
Command: /bin/sleep 1200
Working directory: ${GLOBUS_USER_HOME}
Standard input:
Standard output: mc-stdout ✓
Standard error: mc-stderr ✓
Credentials valid for: 0 days 11 hours 39 minutes

Figure 6. The Job Submission Client interface to web-services based grid

as a facade [25], this Java library encompasses the job submission and management tasks such as querying an MDS for a list of resources, or refreshing the credentials of a long-running job, and presents a higher-level interface to the Globus libraries.

Though the GUI discussed above is useful for job submission and monitoring, it does not incorporate the results of site functional tests such as pinging the resource to check if the server is listening at TCP port 8443, credential delegation test, WS-GRAM job submission test, GridFTP test and file transfer test using the Reliable File Transfer (RFT) service. The site functional tests are very important for knowing the health of the grid. The failure of any test allows the grid operators to remove the problematic cluster from the grid and thus improves the overall reliability and efficiency of the grid by reducing the number of failed jobs. In the existing Canadian grid GridX1, the results of such tests were archived and stored in a MySQL database to allow monitoring of the status of grid resources and their jobs [2]. Condor tools such as Condorq, and Condor history were used to present the jobs status and historical plots of failed/success jobs. All these results were displayed through a web interface. Since the present metascheduling system is still built around Condor-G, the above useful features are retained in the new WSRF based grid. The only difference in the new grid is that the site functional tests are remodelled by using the Globus Toolkit version 4 commands. These tests at each resource are performed by a series of Java classes controlled by a main class called Collector which reports all the results into the MySQL database for storage.

Further, we have enabled Condor Quill to provide useful information about the job queue status and the jobs history. Condor quill is a method of storing the job queue and job historical data in the postgres relational database. The advantage of Condor Quill over Condor history is to provide fault tolerance and performance enhancements in very large and busy pools.

6. Conclusions

We have presented our approach to build an efficient and productive grid, which uses the WSRF-based architecture provided by the Globus Toolkit version 4. Further, it is shown how any real and complicated high energy physics application such as BaBar can be set up and implemented in the framework set out by the WSRF. It may be noted that the main features of this new grid are the Condor-G metascheduler service, and the central registry to keep information about the resource attributes, metaschedulers, and the RFT service. By adding the Condor-G adapter software layer between the WS-GRAM job execution service and the Condor scheduler, the BaBar jobs can be submitted to the metascheduler by using the WS-GRAM service. The metascheduler accepts the jobs, selects the resource by matching the job and resource ClassAds, and performs file staging and job dispatching to the selected resource. An automatic registering of the resource ClassAds to the central registry, and ClassAds extraction from the registry to the metascheduler for matchmaking are useful features of this grid. An added attraction is the design of the user grid submission tool using the Gavia JSAM library and the Globus Java API. The performance of BaBar jobs on this new grid and its comparison with the corresponding runs executed on the existing Canadian grid (GridX1) based on Globus Toolkit version 2 is successful and reliable. A web-based grid monitoring system to provide the status of grid resources and the grid jobs is retained as in the previous Canadian computational grid, GridX1. Only the site functional tests are remodelled using the commands based on Globus Toolkit version 4. Further, the web interface to Condor queue, pool status and job history now use Condor quill rather than Condor history providing performance enhancements in very large and busy pools. In future work, high scalability and reliability of BaBar job submission across the grid will be achieved by using multiple metaschedulers and multiple registries with duplicate information about the available resources and services.

Acknowledgments

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

References

[1] Agarwal A et al 2006 GridX1: A Canadian Particle Physics Grid *Proc. of Comp. in High Energy Physics (Mumbai)* vol 2 ed S Banerjee (Delhi: Macmillan India Ltd.) pp 993-6
[2] Agarwal A et al 2007 GridX1: A Canadian Computational Grid *Future Generation Computer Systems* **23** 680-7
[3] Walker R, Vetterli M, Agarwal A, Sobie R and Vanderster D 2006 A Grid of Grids using Condor-G *Proc. of Comp. in High Energy Physics (Mumbai)* vol 2 ed S Banerjee (Delhi: Macmillan India Ltd.) pp 881-3
[4] The TeraGrid Project 2005 *http://www.teragrid.org/*
[5] The Open Science Grid 2006 *http://www.opensciencegrid.org/*
[6] The LHC Computing Grid Project 2005 *http://lcg.web.cern.ch/LCG/*
[7] The NorduGrid Collaboration 2005 *http://www.nordugrid.org/*
[8] INFN Grid 2006 *http://www.infngrid.org*
[9] Foster I and Kesselman C 1997 Condor-G: A metacomputing infrastructure toolkit *The Int. J. of Supercomputer Applications and High Performance Computing* **11**(2) 115-28
[10] Globus Toolkit 2006 *http://www.globus.org*
[11] Virtual Data Toolkit 2005 *http://www.cs.wise.edu/vdt/*
[12] Frey J, Tannenbaum T, Livny M, Foster I and Tuecke S 2001 Condor-G: A Computation Management Agent for Multi-Institutional Grids *Proc. of 10th Int. Symp. on High Performance Distributed Computing (HPDC-10)* IEEE Press
[13] Tannenbaum T, Wright D, Miller K and Livny M 2001 Condor - A Distributed Job Scheduler *Beowulf Cluster Computing with Linux* MIT Press
[14] Thain D, Tannenbaum T and Livny M 2003 Condor and the grid *Computing: Making the Global Infrastructure a Reality* ed F Berman et al (Toronto: John Wiley & Sons Inc) pp 16-74

[15] Thain D, Tannenbaum T and Livny M 2005 Distributed Computing in Practice: The Condor Experience *Concurrency and Computation: Practice and Experience* **17**(2-4) 323-56
[16] Novotny J, Tuecke S and Welch V 2001 An online credential repository for the grid: Myproxy *Proc. of the Tenth Int. Symp. on High Performance Distributed Computing (HPDC-10)* IEEE Press
[17] Mateescu G et al 2007 The GridX1 computational Grid: from a set of service-specific protocols to a service-oriented approach *21st Int. Symp. on High Performance Computing Systems and Applications* p 8
[18] Gawor J, Humphrey M, Jackson K, Lang S and Meder S 2005 Options for building WSRF compliant services *GlobusWorld*
[19] W3C 2006 SOAP version 1.2 *http://www.w3.org/TR/soap12-part1*
[20] W3C 2006 The web services description language WSDL *http://www.w3.org/2002/ws/desc*
[21] Globus Incubator Project Website 2007 *http://dev.globus.org/wiki/Incubator/Gavia-MS/*
[22] Globus Incubator Project Website 2007 *http://dev.globus.org/wiki/Incubator/Gavia-JSC/*
[23] Portable Batch System Professional 2005 *http://www.altair.com/pbspro/*
[24] W3C 2006 XML schema *http://www.w3.org/XML/Schema*
[25] Gamma E, Helm R, Johnson R and Vlissides J 1995 *Design Patterns: Elements of Reusable Object-Oriented Software* (New York: Addison-Westley) pp 85-93