A Lightweight Campus Grid Resource Broker and Job Submission Tool

David Wallom and Tiejun Ma
Oxford e-Research Centre
University of Oxford
{david.wallom. tiejun.ma}@oerc.ox.ac.uk

Abstract

Resource brokers are generally part of large complicated infrastructures, with many interdependencies, thus making them unsuitable for Campus Grid systems. In this paper we describe the design and implementation of a more lightweight resource broker based on Condor-G [1]. A daemon service has been developed which can read LDAP based information and translate the published information into a Condor ClassAd [2]. This additionally draws from a connected organization management system registered system capability information e.g., SMP, MPI and installed software.

In addition, a command line submission tool has been developed, which makes user authorization decisions, to minimize submission errors as well as additional individual tools for viewing the system state including resources utilized and jobs queued and removal of tasks that have been submitted erroneously.

This broker service is running on the University of Oxford Campus Grid in a production environment and has efficiently distributed over 63k individual tasks to university and UK National Grid Service resources.

1. Introduction

When first suggested computational Grids [3] were designed so that a user would submit work within a Grid environment after which the Grid itself would decide the location best suited to execute the task. Even with the rise of several toolkits for the construction of Grids [4], automated resource discovery and job distribution though has been lacking development.

Within a university Campus Grid environment though it is essential that a method of fair and equitable distribution of tasks across connected resources should be available. This is especially true where the system is intending to peer national infrastructures with departmental resources.

Most current solutions to this problem though are generally part of significant infrastructures [5] with software interdependencies that make use of individual components very difficult. This in turn makes then unsuitable for deployment in a lightweight software environment that must not intrude into the workings of already installed systems including those that may be heavily used by their current owners. In this paper, we will introduce a more lightweight resource broker toolkit, which brings more flexibility to a campus level Grid.

2. Using Grid Wide Information to Make Condor ClassAds

In order to make efficient decisions for resource provisioning, within a Campus Grid system there is a specific set of information to which it is necessary to have access. These are split into two classes as follows:

a. Static information: e.g. processor architecture, physical memory, operating system, scheduling system, no. of nodes. These properties are fixed unless the resource is upgraded or reinstalled. Therefore these can be retrieved infrequently from a single information source.
b. Transient information: e.g., system availability, scheduler load, queue length, used disk or memory. These properties are

only valid for a fairly short period and must be kept constantly up to date.

These two classes of information can also be separated as to their physical data sources, a system relational database for static information and a dynamic information source (LDAP based) for transient information. There is a choice for transient information, which can either be read directly from the individual connected resources or from a single Globus Grid Information Index Server [7] or a Berkley Database Information Index system [7]. This aggregates the individual resource information to create a single point of retrieval for system wide information containing a description of the resource type, capability and current capacity for each resource. In order to ensure uniformity of information gathered, a specific schema must be chosen. For this lightweight toolkit, the raw Globus Monitoring and Discovery Service version 2 (MDS2) [6] was used though this can be extended through the use of the GLUE information schema [9]. This would particularly allow the integration of EGEE resources in the future.

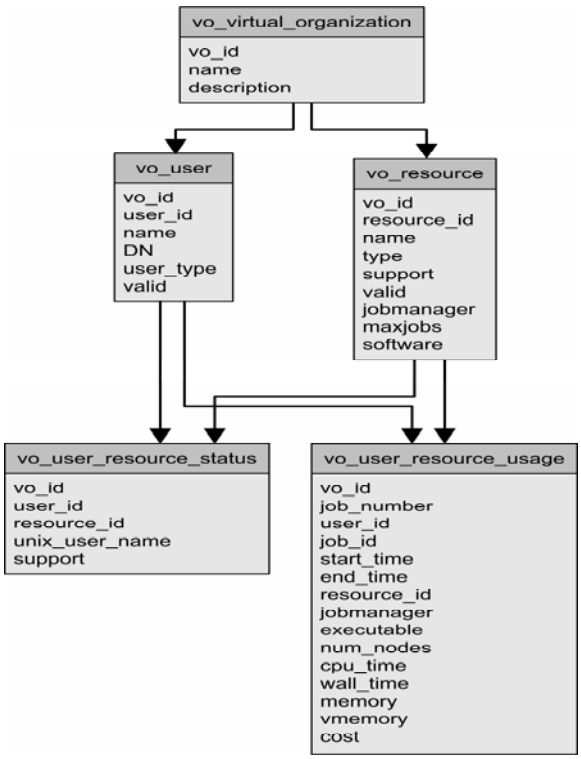


Figure 1. The Schema for the Underlying System Registration and Virtual Organisation Management System

To ensure that correctly registered resource are available to users both from the point of view of security and correct operation a relational database is attached to the system which contains static properties about connected resources. This gives additional security by cross checking the information received from the information server with what is recovered from the database. Figure 1 shows the database schema for the lightweight resource broker, which can be regarded as the data structure of a virtual organization. This has been implemented using a Postgres relational database, which can quite easily be accommodated on the same server as the Condor master daemons. Additional functionality has been built into this database to register the users and which system each is allowed to use as well as provide a central repository for accounting information across the whole Grid

2.1. Translating the Information into a Class-Ad

The format of the Condor ClassAd is well documented [6] and so will not be dwelt on except to give the example below so the reader can appreciate the extra text that must be added.

MyType = "Machine"
TargetType = "Job"
Name = ”bedrock.oucs.ox.ac.uk-condor“
gatekeeper_url=”bedrock.oucs.ox.ac.uk/jobmanager-condor"
*Requirements=(**CurMatches<20**)&&*
(TARGET.JobUniverse == 9)
WantAdRevaluate = True
UpdateSequenceNumber = 1097580300
CurMatches = 0
OpSys = "LINUX“
Arch = "INTEL"
Memory = 501
MPI = False
INTEL_COMPILER=True
GCC3=True

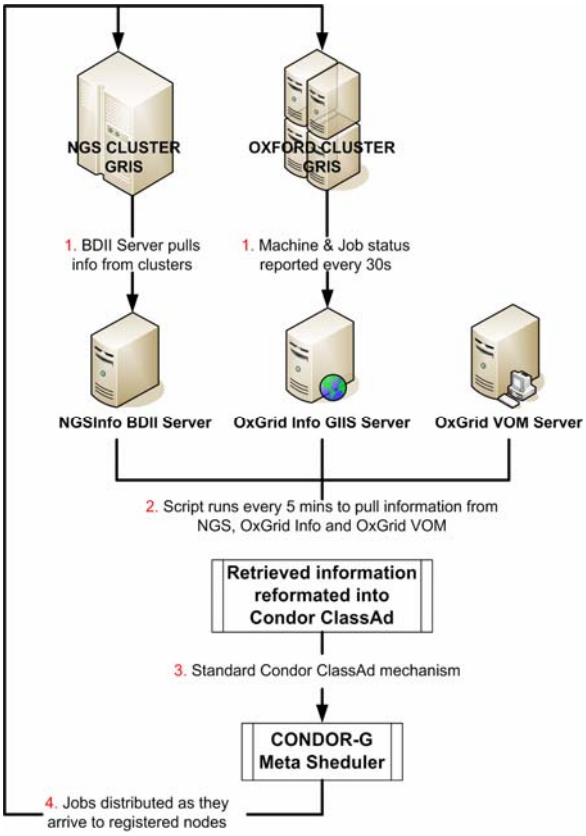
The difference of the above ClassAd differs from a standard one is highlighted in **bold**. These include a specified maximum number of matched individual jobs that can be sent to this Grid

resource, CurMatches, which is set by the resource owner and can be updated by them as necessary dependent on local load. There is also a set of additional variables describing specific resource capabilities which are tested against when a job is submitted by the Condor matchmaker. In this example there are MPI, INTEL_COMPLIER and GCC3, though these are free variables and so any capability can be described in this manner and hence tested by a submitted job.

In order to reformat the resource information for the Condor system, we have developed a translator which gathers the resource information within the Campus Grid system and reorganizes this as a Condor ClassAd ready for reading into Condor. The detailed steps in this operation are as follows and are shown in Figure 2.

1. Read from information server file names and LDAP roots of the specified information servers.
2. Check that a specific resource has not been excluded for operational reasons, if this is so bypass any reference to it in the information server LDAP retrieved.
3. Perform a complete retrieval on the full LDAP tree and search for the following:
    a. Hostname,
    b. Job Scheduler,
    c. Operating System,
    d. Number of processors and their architecture,
    e. Physical Memory,
    f. Volume of Local Storage.
4. From the system relational database retrieve the following information:
    a. Job Scheduler (to check against the one being published within the information schema),
    b. Maximum number of jobs that can be matched with a resource,
    c. Names of installed and registered software packages,
    d. By querying the Condor queue to find out how many tasks are currently matched against the resource and generate the CurMatches variable value. According to the above steps, ClassAd will not be made, unless all of these factors are present the resource. There is also a specific exclusion method should it be found that a systems retrieved LDAP information is corrupted (most likely through the published Job Scheduler not matching the registered systems) will also cause the ClassAd not to be written.
5. In parallel as each resource is discovered in the information server, a request is made to the system database to verify the information retrieved from the LDAP system. This ensures that no rogue systems are registering with the information server and possibly corrupting the Grid.
6. The ClassAd itself is now written using a custom library that was designed to separate the writing from the collecting the information. This modularity will allow alterations to both methods without alteration to the interface between them.
7. Finally, using a specific Condor command for each system and ClassAd is then added to the Condor collector and then removed.

Figure 2. Overall Operation of the Data



Harvester Resource Broker

Following the above steps, it results in a list of remote systems being visible to the Condor system as if they were local Condor workers and can therefore take part in matchmaking. Figure 3 shows a sample matchmaking query results for the Oxford Campus Grid system.

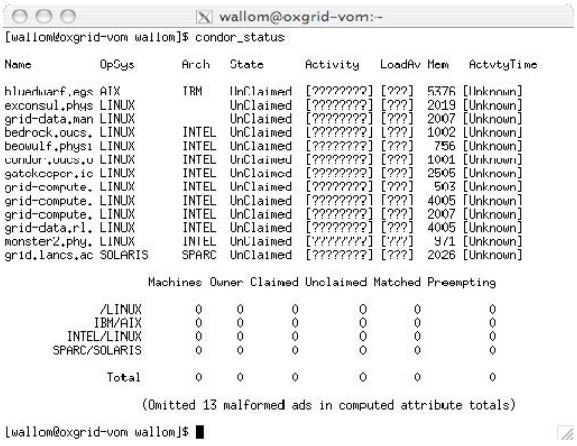


Figure 3. Resources available to the Condor-G system within the Oxford Campus Grid

2.2. Tuning Condor for Operation as a Meta-scheduler

In order to make more precise job allocation within the whole Campus Grid system, we described a way to tuning the Condor as a meta-scheduler for jobs delegation. For such a purpose, there are several steps that must be taken with Condor to ensure its operation as an efficient meta-scheduler. It must have all of the operational changes to make it as efficient as possible in its use of remote resources. These include:

- *NEGOTIATE_ALL_JOBS_IN_CLUSTER = True*. This ensures that for each pass of the Matchmaker all submitted tasks are attempted to be matched to available resources rather than only on the first task in the queue.
- *RunBenchMark=False*. Since the resource broker is not an execution node then this removes additional possible operations that waste time in the matchmaking cycle.
- *JOB_START_DELAY = 10*. As each submitted task is begun the Condor system will wait for 10 seconds before trying to start the next.
- *ENABLE_GRID_MANAGER = True*. The Grid manager is used to control the actual submission of Grid tasks from the Condor system and as such must be enabled.
- *GRIDMANAGER_MAX_SUBMITTED_JOBS_PER_RESOURCE = 20*. Since a user is able to manually submit directly to a connected grid resource there needs to be a mechanism by which an already heavily loaded resource will not get swamped by jobs that the Matchmaker submits on top.
- *GRIDMANAGER_MAX_PENDING_SUBMITS_PER_RESOURCE = 10*. To ensure that once a machine has the maximum number of running jobs the Matchmaker does not just continue to submit jobs to that resource even though they may be very long tasks and resource are available elsewhere we ensure that only 10 jobs at a time can be queued on the node waiting for execution.
- *GRIDMANAGER_JOB_PROBE_INTERVAL=60*. To ensure that jobs are cleared from the execute host but regularly but do not have their status overly check and hence overload the execution node system head node the interval is raised to every 10 seconds.
- DEFAULT_UNIVERSE = GLOBUS. The Condor system can handle many different execution environments. Since this is operating as a metascheduler then the default if none have been defined in the job submission script is to consider the job a Globus/Grid job.
- *CLASSAD_LIFETIME = 900*. As each machine Class-Ad has time specific information in it then there must be a period after which the resource broker considers it invalid. We have set this at 15 minutes to allow for any transient communication problems between the RB service and a particular remote resource.
- *NEGOTIATOR_INTERVAL = 30*. This is the period between each Matchmaking cycle within the Condor Negotiator Daemon.

The above steps will guarantee having the Grid-manager setup correctly. The next ting is to ensure that the system performs matchmaking as often as possible and tries to match all jobs in the queue, not just the first. This ensures that all chances to run jobs within the queue are taken.

3. Simplifying Job Submission Procedure into the Resource Broker

The Condor system uses a bespoke job submission language [2] that the Campus Grid users would have to learn the basics before they can submit tasks. This is not convenient since one of our primary aims is to not alter how our users interact with the applications they have already written to perform their research. We have developed a separate tool to simplify the job submission procedure and hence improve the usability of the Campus Grid.

In addition, it has long been recognized that the most useful interface for most experienced computational scientists is through the command line. This generally is through an application which operates by taking the task input parameters as options. Once executed it automatically only submits child tasks to resources that they have permission to use and packages their application with the dependent libraries or software that it needs to complete successfully. Therefore we developed an interactive command line application, job-submission-script, which has been implemented in Perl. The following are the operations of the job-submission-script:

- The command line option are passed and contains the parameters as follows:
  - Name of the executable to be run,
  - Boolean variable on whether to transfer to executable to the remote execution host,
  - Arguments necessary to be passed to the executable,
  - Names of extra input files to be transferred to the execution host,
  - Name of the output files to be transferred back from the execution host,
  - Licensed software required to run the task on the execution host,
  - The user may also specify a specific system within the Grid though this is not the default method of operation.
- Interrogate the system database to produce a list of registered system that the submitting user is allowed to use. This produces a text string of hostnames with jobamangers.
- Start writing the Condor job submit file, an example of which follows:

executable = update_file
Transfer_Executable = True
globusscheduler = $$(gatekeeper_url)
Requirements =
(TARGET.gatekeeper_url == "t2ce02.physics.ox.ac.uk/jobmanager-lcgpbs" || TARGET.gatekeeper_url == "condor.oucs.ox.ac.uk/jobmanager-condor" || TARGET.gatekeeper_url == "grid-compute.oesc.ox.ac.uk/jobmanager-pbsox" ||
TARGET.gatekeeper_url == "bedrock.oucs.ox.ac.uk/jobmanager-sge")
&& TARGET.gatekeeper_url =!=
UNDEFINED && **TARGET.OpSys == "LINUX"**
match_list_length = 1
arguments = TEST_3_2.in TEST_3_2.out
transfer_input_files = TEST_3_2.in
transfer_output_files = TEST_3_2.out
WhenToTransferOutput = ON_EXIT
universe = grid
grid_type = gt2
notification = ERROR
output = temp-1168783341-2.out
error = temp-1168783341-2.err
log = temp-1168783341-2.log
queue

Once the submission file has been written, the job is then submitted into the Condor system. The items in **bold** are those that are directly specified by the submitting user whilst the values in ***bold italics*** are retrieved from the system database.

Important parts of the submission file are that the name for the 'globusscheduler' variable, which is used to define which remote resource the job should be matched with, in this case defined as $$(gatekeeper_url) which indicates that the

Matchmaker should make the decision, is exactly the same within both submit file and the machine ClassAd constructed using the translator service.

4. Conclusion

In this paper we have described the framework of a lightweight resource broker and its support toolkits for a Campus level Grid service, which adopted the Condor-G system as the underlying backbone of the Oxford Campus Grid infrastructure. This has been shown to be operationally easy to use and maintain. An important feature of this Grid system is that the automatic generation of registered resource Class-Ads and writing of job submission scripts are added to make the system as a whole a fully functional meta-scheduler for a Campus Grid, which has improved efficiency and usability. Such features are important for interdisciplinary resource sharing and accelerate the university wide resource integration. This has all been done by providing a lightweight Perl based toolkit added to a standard Condor-G instance in comparison to other available brokers which use many thousands of lines of code, require specific configuration of remote resources etc.

Within the last year the Oxford resource broker has launched ~63000 individual tasks onto resources located within the departments of University of Oxford and the resources of the NGS.

5. Future works

In the future, we will integrate more Grid resources into our Campus Grid system and increase diversity of available resource brokers within the Campus Grid, such as adopting Nimrod-G [8], which is pull-style job schedulers, improving the interactivity between job schedulers and building a more intelligent meta-schedulers. All of these aspects will be involved for our next step Campus Grid design and development.

6. Acknowledgments

We would like to acknowledge the help and assistance of the Condor team at Wisconsin as well as Dr. Jon Wakelin for assistance with the Perl coding to ensure efficient use of resources.

7. References

[1] Frey, J. and Tannenbaum, T. and Livny, M. and Foster, I. and Tuecke, S., Condor-G: A Computation Management Agent for Multi-Institutional Grids, *Cluster Computing 2002*, Vol 5, Page 237-246, Springer, 2002.

[2] Solomon, M, The ClassAd Language Reference Manual, Version 2.1, Computer Sciences Department, University of Wisconsin, Madison, WI, USA http://www.cs.wisc.edu/condor/classad/, Oct, 2003.

[3] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[4] I. Foster, C. Kesselman, Globus: a Metacomputing Infrastructure Toolkit, *International Journal of High Performance Computing Applications*, Vol. 11, No. 2, 115-128 (1997).

[5] Gagliardi, F. and Jones, B. and Grey, F. and Begin, M.E. and Heikkurinen, M., Building an infrastructure for scientific Grid computing: status and goals of the EGEE project, *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, Vol 363, Page 1729-1742, The Royal Society, 2005.

[6] Czajkowski, K. and Fitzgerald, S. and Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing, *10th IEEE International Symposium on High Performance Distributed Computing*, Page 181—184, 2001.

[7] http://agrid.uibk.ac.at/wpa2/bdii.html

[8] Buyya, R. and Abramson, D. and Giddy, J., Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, *HPC Asia*, 2000.

[9] Andreozzi, S. and Burke, S. and Field, L. and Fisher, S. and Konya, B. and Mambelli, M. and Schopf, JM and Viljoen, M. and Wilson, A., GLUE Schema Specification-Version1.2, http://glueschema.forge.cnaf.infn.it/, Dec, 2005.

[10] N. Coleman, R. Raman, M. Livny and M. Solomon, Distributed Policy Management and Comprehension with Classified Advertisements, UW-CS-TR-1481, April, 2003.