In *Applied Parallel Computing. State-of-the-art in Scientific Computing.*
Springer-Verlag, Lecture Notes in Computer Science, (to appear).

A Grid Resource Broker Supporting Advance Reservations and Benchmark-Based Resource Selection

Erik Elmroth and Johan Tordsson

Dept. of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden.
`{elmroth, tordsson}@cs.umu.se`

Abstract. This contribution presents algorithms, methods, and software for a Grid resource manager, responsible for resource brokering and scheduling in early production Grids. The broker selects computing resources based on actual job requirements and a number of criteria identifying the available resources, with the aim to minimize the total time to delivery for the individual application. The total time to delivery includes the time for program execution, batch queue waiting, input/output data transfer, and executable staging. Main features of the resource manager include advance reservations, resource selection based on computer benchmark results and network performance predictions, and a basic adaptation facility.
Keywords: Resource broker, scheduling, production Grid, benchmark-based resource selection, advance reservations, adaptation, Globus toolkit.

1 Introduction

The task of a Grid resource broker and scheduler is to dynamically identify and characterize the available resources, and to select and allocate the most appropriate resources for a given job. The resources are typically heterogeneous, locally administered, and accessible under different local access policies. The broker operates without global control, and its decisions are entirely based on the information provided by individual resources and information services with aggregated resource information. In addition to information about what resources are available, each resource may provide static information about architecture type, memory configuration, CPU clock frequency, operating system, local scheduling system, various policy issues, etc, and dynamic information such as current load and batch queue status. For more information on typical requirements for a good resource broker, see [3].

A reservation capability is vital for enabling co-allocation of resources in highly utilized Grids. This feature also provides a guaranteed alternative to predicting batch queue waiting times. For general discussions about resource

This work was funded by The Swedish Research Council (VR) under contract 343-2003-953 and The Faculty of Science and Engineering, Umeå University.

reservations (and co-allocations), see, e.g., [5, 8]. A reservation feature naturally depends on the reservation support provided by local schedulers [12] and the use of advance reservations also have implications on the utilization of each local resource [21].

The performance differences between Grid resources and the fact that their relative performance characteristics may vary for different types of applications makes resource selection difficult, see, e.g., [9, 16, 19, 20]. Our approach to handle this is to use a benchmark-based procedure for resource selection. Based on the user's identification of relevant benchmarks and an estimated execution time on some specified resource, the broker estimates the execution time for all resources of interest. This requires that a relevant set of benchmark results are available from the resources' information systems.

As most Grid resources still are available without performance and queue time guarantees, the adaptation feature provided by our resource broker is useful. It allows a user to request that the broker after an initial job submission strives to re-direct the job to another resource, likely to give a shorter total time to delivery.

A resource broker and scheduler is fundamental in any large-scale Grid-environment for scientific applications. Our software is mainly targeted for the NorduGrid and SweGrid infrastructures. These are both Globus-based production environments for 24 hour per day Grid usage.

The outline of the paper is as follows. Section 2 gives a brief introduction to the NorduGrid software and the general resource brokering problem. The main algorithms and techniques of our resource broker are presented in Section 3, including, e.g., support for making advance resource reservations and to select resources based on estimates of the total time to delivery. The latter feature builds on benchmark-based execution time estimates and network performance predictions. Minor extensions to the NorduGrid user interface are presented in Section 4. Sections 5 and 6 present some concluding remarks and acknowledgments, respectively, followed by a list of references.

2 Background and Motivation

Our development of resource brokering algorithms and prototype implementations are mainly focused on the infrastructure and usage scenarios typical for NorduGrid [13] and SweGrid [23]. The main Grid middleware used is the NorduGrid software [6]. The Grid resources are typically Linux-based clusters (so, in the rest of this paper, the word cluster is often used instead of the more general term Grid resource). NorduGrid includes over 40 clusters of varying size, most of them located in the Nordic countries. SweGrid currently includes six clusters, each with 100 Pentium 4 processors.

2.1 The NorduGrid software

The NorduGrid middleware is based on standard protocols and software such as OpenLdap [14], OpenSSL [15] and Globus Toolkit version 2 [7, 10]. The latter is not used in full, as some Globus components such as the GRAM (with the gatekeeper and jobmanager components) are replaced by custom made components [6].

NorduGrid makes use of the Grid Security Infrastructure (GSI) in Globus Toolkit 2. GSI specifies a public key infrastructure, and SSL (TLS) for authenticated and private communication. GSI also specifies short-term user-proxy certificates, signed by either the user or by another proxy. The proxy allows the user to access various remote resources without (re)typing passwords and to delegate authority. An extension of a subset of the Globus resource specification language (RSL) [17] is used to specify resource requirements and necessary job execution information.

The NorduGrid user interface consists of command line tools for managing jobs. Users can submit jobs, monitor the execution of their jobs and cancel jobs. The resource broker is part of the the job submission tool, `ngsub`. Other tools allows, e.g., the user to retrieve output from jobs, get a peak preview of job output and remove job output files from a remote resource. Communication with remote resources is handled by a simple GridFTP client module.

Each Grid resource runs one instance of the NorduGrid *GridFTP server*. When submitting a job, the user invokes the broker which uploads an RSL job submission request to the GridFTP server. The GridFTP server specifies plugins for custom handling of FTP protocol messages. NorduGrid makes use of these for Grid access to the local file systems of the resources, to manage Grid access control lists, and most important, for Grid job management.

Each resource also runs a *Grid manager* that manages the Grid jobs through the various phases of their execution. The Grid manager periodically searches for recently submitted jobs. For each new job, the RSL job description is analyzed, and any required input files are staged to the resource using GridFTP. The job description is translated into the language of the local scheduler, and the job is submitted to the batch system. Upon job completion, the Grid manager stages the output files to the locations specified in the job description.

2.2 The resource brokering problem

In general, we can identify two major classes of Grid resource brokers, namely centralized and distributed brokers. A centralized broker manages and schedules all jobs submitted to the Grid, while a distributed broker typically handles jobs submitted by a single user only. Centralized brokers are, at least in theory, able to produce optimal schedules as they have full knowledge of the jobs and resources, but such a broker can easily become a performance bottleneck and a single point of failure. Moreover, they turn a Grid environment more into a single virtual computer than into a dynamic infrastructure for flexible resource sharing. A distributed broker architecture, on the other hand, scales well and makes the Grid more fault-tolerant, but the partial information available to each instance of the broker complicates the scheduling decisions. A hybrid type is the hierarchical broker architecture, in which distributed brokers are scheduled by centralized brokers, attempting to combine the best of two worlds. Grid systems utilizing a centralized broker include, e.g., EDG [18] and Condor [11]. Distributed brokers are implemented by, e.g., AppLeS [2] and NetSolve [4]. For further discussions on a similar classification of brokers, see, e.g., [1].

In the following we focus on algorithms and software for a distributed resource broker. Such a broker typically seeks to fulfill the user's resource requests by selecting the resources that best suit the user's application. Selecting the most suitable resources often means identifying the resources that provide the shortest Total Time to Delivery (TTD) for the job. TTD is the total time from the user's job submission to the time when the output files are stored where requested. This includes the time required for transferring input files and executable to the resource, the waiting time, e.g., in a batch queue, the actual execution time, and the time to transfer the output files to the requested location(s).

In order to manage this task, the broker needs to identify what resources are available to the user, to characterize the resources, to estimate all parts of the TTD, etc. Other requests the user may put on the broker is to make advance resource reservations, e.g., at a specific time, or to not only submit the job but also to adapt to possible changes in resource load and possibly perform job migration.

3 Resource Brokering Algorithms

Our main brokering algorithm performs a series of tasks, e.g., it processes the RSL specifications of job requests, identifies and characterizes the resources available, estimates the total time to delivery for each resource of interest, makes advance reservation of resources, performs the actual job submission. Figure 1 presents a high-level outline of the algorithm.

```
Input: RSL-specification(s) of one or more job requests.
Action: Select and submit jobs to the most appropriate resources.
Output: none.

1. Process RSL specification and create a list of all individual job requests.
2. Contact GIIS server(s) to obtain a list of available clusters.
3. Contact each resource's GRIS for static and dynamic resource information (hard-
   ware and software characteristics, current queue and load, etc).
4. For each job:
   (a) Select the cluster to which the job will be submitted:
         i. Filter out clusters that do not fulfill the requirements on memory, disk
            space, architecture etc, and clusters that the user is not authorized to use.
        ii. Estimate TTD for each remaining resource (see Section 3.2).
            If requested, resource reservation is performed during this process.
       iii. Select the cluster with the shortest predicted TTD.
   (b) Submit the job to the selected resource.
   (c) Release any reservations made to non-selected clusters.
```

Fig. 1. Brokering algorithm overview.

The input RSL specification(s) contains one or more job requests including information about the job to run (e.g., executable, input/output files, arguments), actual job requirements (e.g., amount of memory needed, architecture requirement, computer time required, requests for advance reservations), and optionally, job characteristics that can be used to make improved resource selection (e.g., listing of benchmarks with relevant performance characteristics).

In Step 1, the user's request is processed and separated into individual jobs. In Step 2, the broker identifies what resources that are available by contacting one or more Grid Index Information Services. The specific characteristics of the resources found are identified in Step 3, by contacting the Grid Resource Information Service on each individual resource. The actual brokering process is mainly performed in Step 4, where resources are evaluated, selected, and optionally reserved in advance for each job. Finally, the jobs are actually submitted and any non-utilized reservations are released. In the following presentation, we focus on the more intricate issues of performing advance reservations and on how to determine an estimate (prediction) of the total time to delivery.

Notably, this algorithm does not reorder the individual job requests (when multiple jobs are submitted in a single invocation). This can possibly be done in order to reduce the average batch queue waiting time, at least by submitting shorter jobs before longer ones given that they require the same number of CPUs. In the general case, factors such as local scheduling algorithms and competing Grid brokers make the advantage of job reordering less obvious.

3.1 Advance resource reservations

The advanced reservation feature makes it possible to obtain a guaranteed start time in advance. A guaranteed start time brings two advantages. It makes it possible to coordinate the job with other activities, and resource selection can be improved as the resource comparison is based on a guaranteed start time rather than on an estimate.

The reservation protocol developed supports two operations: requesting a reservation and releasing a reservation. A reservation request contains the start time and the requested length of the reservation, the requested number of CPUs, and optionally, an account to be charged for the job. Upon receiving a reservation request from the broker, the GridFTP server adds the user's local identity to the request, information unknown to the broker but required by the local scheduler. Then, the GridFTP server invokes a script to request a reservation from the local scheduler. If the scheduler accepts the request and creates the reservation, the GridFTP server sends a unique identifier and the start time of the reservation to the broker. If no reservation could be created, a message indicating failure is returned to the broker. The GridFTP server saves the reservation identifier and a copy of the user's proxy for every successful reservation, enabling subsequent identification of the user who made the reservation.

For releasing a reservation, the broker uploads a release message containing the reservation identifier and the GridFTP server confirms that the reservation is released.

Job submission with a reservation. Upon a successful reservation, the broker adds the received reservation identifier to the RSL job description before submitting the job to the resource.

Before the job is submitted to the local scheduler, the Grid manager analyzes the job description and detects the reservation identifier. The Grid manager inspects the saved proxy files and their associated reservation identifiers to ensure that the reservation exists. Furthermore, the Grid manager compares the proxy used to submit the job with the one used to make the reservation, ensuring that no user can submit jobs to another user's reservation by spoofing the reservation id. The job is not submitted to the local scheduler unless the specified reservation exists and was created by the user submitting the Grid job.

When the job finishes executing on the resource, the Grid manager may remove the reservation, allowing the user to run only the requested job. Alternatively, resources may allow the user to submit more jobs to the reservation once the first has finished. The configuration of the Grid manager determines the policy to be used.

The advance resource reservation feature requires a reservation capability in the local scheduler. The current implementation supports the Maui scheduler [22], although any local scheduler may be used. Support for other schedulers can easily be added by adapting the scripts interacting with the local scheduler (see, e.g., [12]).

3.2 Estimating the Total Time to Delivery

The estimation of the total time to delivery (TTD), from the user's job submission to the final delivery of output files to requested storage requires that the time to perform the following operations is estimated:

1. Stage in: transfer of input files and executable to the resource,
2. Waiting, e.g., waiting in batch queue and for operation 1 to complete,
3. Execution, and,
4. Stage out: transfer of output files to requested location.

Notably, the waiting time is here defined as the maximum of the time for stage in and all other waiting times before the job actually can start to execute. The estimated TTD is given as the sum of estimated times for operations 2, 3, and 4. Sometimes, the time for stage out cannot be estimated, due to lack of information about output file sizes. In these cases that part is simply omitted from the TTD used for comparisons. Below, we summarize how we make these estimates.

Benchmark-based time predictions. The execution time estimate needs to be based both on the performance of the resource and the characteristics of the applications. This issue is made slightly more intricate by the fact that the relative performance difference between different computing resources typically varies with the character of the application. In order to circumvent this problem, we give the user the opportunity to specify one or more benchmarks with performance characteristics similar to those of the application. This information is given together with an execution time estimate on a resource with a specified benchmark result. Based on this information and benchmark results for each individual resource, we make execution time estimates for all resources of interest. In doing this, we assume linear scaling of the application in relation to the benchmark, i.e., a resource with a benchmark result a factor k better is assumed to execute the application a factor k faster.

We remark that a good execution time estimate serves two purposes. First, a sufficient but short estimated execution time may lead to an earlier job start, due to standard batch system scheduling algorithms. Second, a good estimate is more likely not to be too short, and hence reduces the risk for job preemption by the local scheduler.

The user can specify k benchmarks as triples $\{b_i, r_i, t_i\}, i = 1, \ldots, k$, where b_i is the benchmark name, r_i is the benchmark result on a system where the application requires the time t_i. The broker matches these specifications with the benchmark results provided by each cluster. For each requested benchmark that is available at the resource, an execution time for the application is predicted.

If the cluster does not provide a result for a requested benchmark, the corresponding time estimate is taken to be a penalty factor c times the longest execution time estimated from other benchmarks for that cluster. The penalty factor can be configured by the user. As a default value, $c = 1.25$ is used.

When comparing the performance of different clusters during the resource selection procedure, one part of the TTD estimate for each resource is given from this procedure. The value used is the average execution time estimate of the k estimates obtained from different benchmarks. At job submission, after the selection procedure, an execution time must be included in the request sent to the resource. In order to reduce the risk for scheduler preemption, this execution time is chosen as the maximum of the k estimates calculated.

Network performance predictions. The time estimation for the stage in and stage out procedures are based on the actual (known) sizes of input files and the executable file, user-provided estimates for the sizes of the output files, and network bandwidth predictions.

The network bandwidth predictions are performed using the Network Weather Service (NWS) [24]. NWS combines periodic bandwidth measurements with statistical methods to make short-term predictions about the available bandwidth.

3.3 Job queue adaptation

Information gathered about the state of a Grid is necessarily old. Network load and batch queue sizes may change rapidly, new resources may appear and others become unavailable. The load predictions used by the broker as a basis for resource selection will become out-of-date. Nevertheless, more recent information will always be available as Grid resources periodically advertise their state. With this in mind, the broker can keep searching for better resources once the initial job submission is done. If a new resource that is likely to result in a earlier job completion time is found (taking into account all components of TTD, including file restaging), the broker migrates the job to the new resource. This procedure is repeated until the job starts to execute on the currently selected resource.

4 User Interface Extensions

We have extended the standard NorduGrid user interface with some new options and added some new attributes to the NorduGrid RSL, making the new features available to users.

Benchmarks and advance resource reservations. In order to make use of the feature of benchmark-based execution time prediction, the user must provide relevant benchmark information as described by the following example. Assume that the user knows that the performance of the application my_app is well characterized by the NAS benchmarks LU, BT and CG. For each of these benchmarks, the user needs to specify a benchmark result and an expected execution time on a system corresponding to that benchmark result. Notably, the expected execution time must be given for each benchmark, as the benchmark results may be from different reference computers. This is specified using the new RSL attribute `benchmarks`.

Figure 2 illustrates how the user specifies that the application requires 65 minutes on a cluster where the results for the NAS LU and BT benchmarks class C are 250 and 200, respectively. The estimated execution time is 50 minutes on an (apparently different) cluster where the CG benchmark result is 90.

```
&(executable = my_app)(stdin = my_app.in)(stdout = my_app.out)
 (benchmarks = (nas-lu-c 250:65)(nas-bt-c 200:65)(nas-cg-c 90:50))
```

Fig. 2. Sample RSL request including benchmark-based execution time predictions.

Network transfers. In the example in Figure 3, the job involves transfer of large input and output files. The broker will determine the actual sizes of the input files when estimating the transfer time for these. The new, optional, RSL attribute `outputfilessizes` allows the user to provide an estimate of the size of the job output. As shown in the figure, the user need not include all output files in the `outputfilessizes` relation. Estimated file sizes can specified in bytes or with any of the suffixes kB, MB or GB. The NorduGrid middleware uses the `inputfiles` and `outputfiles` attributes as a replacement for `file_stage_in` and `file_stage_out` from the RSL specification.

Command line options. In addition to the RSL extension, the broker supports some new command line options. The option `-A` is used to request the broker to perform queue adaptation. The reservation feature is invoked using the option `-R`. The option `-S` is used to build a pipeline between jobs, so that output from one job is used as input to the next.

```
&(executable = my_program)(arguments = params input)(stdout = logfile)
 (inputfiles = (params gsiftp://host1/file1)
               (input http://host2/file2))
 (outputfiles = (results gsiftp://host3/my_program.results)
                (data gsiftp://host3/my_program.data)
                (logfile gsiftp://host4/my_program.log))
 (outputfilessizes = (results 230MB)
                     (data 5GB))
```

Fig. 3. Sample RSL request with specifications required to estimate file transfer times.

5 Concluding Remarks

The resource broker presented is developed with focus on the NorduGrid software and the NorduGrid and SweGrid production environments. It includes support for making advance resource reservations and selects resources based on benchmark-based execution time estimates and network performance predictions. The broker is a built-in component of the user's submission software, and is hence a user-owned broker acting with no need for global control, entirely basing its decisions on the dynamic information provided by the resources.

6 Acknowledgments

We acknowledge Åke Sandgren, High Performance Computing Center North (HPC2N), Umeå, Sweden, for technical support. We are also grateful for the constructive comments in the feedback from the refereeing process.

References

1. C. Anglano, T. Ferrari, F. Giacomini, F. Prelz, and M. Sgaravatto. WP01 report on current technology. http://server11.infn.it/workload-grid/docs/DataGrid-01-TED-0102-1_0.pdf.
2. F. Berman and R. Wolski. The AppLeS project: A status report. In N. Koike, editor, *Proceedings of the 8th NEC Research Symposium*, 1997.
3. J. Brooke and D. Fellows. Draft discussion document for GPA-WG – Abstraction of functions for resource brokers. http://grid.lbl.gov/GPA/GGF7_rbdraft.pdf.
4. H. Casanova and J. Dongarra. Netsolve: A network server for solving computational science problems. *Int. J. Supercomput. Appl.*, 11(3):212–223, 1997.
5. K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In Feitelson D.G. et al., editors, *Job Scheduling Strategies for Parallel Processing*, volume 2537 of *Lecture Notes in Computer Science*, pages 153–183, Berlin, 2002. Springer-Verlag.
6. P. Eerola, B. Kónya, O. Smirnova, T. Ekelöf, M. Ellert, J.R. Hansen, J.L. Nielsen, A. Wäänänen, A. Konstantinov, J. Herrala, M. Tuisku, T. Myklebust, F. Ould-Saada, and B. Vinter. The NorduGrid production Grid infrastructure, status and plans. In *Proc. 4th International Workshop on Grid Computing*, pages 158–165. IEEE CS Press, 2003.

7. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int. J. Supercomput. Appl.*, 11(2):115–128, 1997.
8. I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *7th International Workshop on Quality of Service*, pages 27–36, Washington - Brussels - Tokyo, 1999. IEEE.
9. I. Foster, J.M. Schopf, and L. Yang. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Proceedings of the ACM/IEEE SC2003: International Conference for High Performance Computing and Communications*, 2003.
10. Globus. http://www.globus.org.
11. M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, June 1988.
12. J. MacLaren. Advance reservations state of the art. http://www.fz-juelich.de/zam/RD/coop/ggf/graap/sched-graap-2.0.html.
13. NorduGrid. http://www.nordugrid.org.
14. OpenLDAP. http://www.openldap.org.
15. OpenSSL. http://www.openssl.org.
16. K. Ranganathan and I. Foster. Simulation studies of computation and data scheduling algorithms for data Grids. *Journal of Grid Computing*, 1(1):53–62, 2003.
17. The Globus Resource Specification Language RSL v1.0. http://www-fp.globus.org/gram/rsl_spec1.html.
18. M. Ruda, C. Anglano, S. Barale, L. Gaido, A.Guarise, S. Lusso, A. Werbrouck, S. Beco, F. Pacini, A. Terracina, A. Maraschini, S. Cavalieri, S. Monforte, F. Donno, A, Ghiselli, F. Giacomini, E. Ronchieri, D. Kouril, A. Krenek, L. Matyska, M. Mulac, J. Popisil, Z. Salvet, J. Sitera, J.Visek, M. Vocu, M. Mezzadri, F. Prelz, M. Sgaravatto, and M. Verlato. Integrating Grid tools to build a computing resource broker: activities of datagrid WP1. In *CHEP'01 computing in high energy and nuclear physics.*
19. W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, Berlin, 1999. Springer-Verlag.
20. W. Smith, I. Foster, and V. Taylor. Using run-time predictions to estimate queue wait times and improve scheduler performance. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 202–219, Berlin, 1999. Springer-Verlag.
21. W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *14th International Parallel and Distributed Processing Symposium*, pages 127–132, Washington - Brussels - Tokyo, 2000. IEEE.
22. Supercluster.org. Center for HPC Cluster Resource Management. http://www.supercluster.org.
23. Swegrid. http://www.swegrid.se.
24. Rich Wolski. Dynamically forecasting network performance using the network weather service. *Journal of Cluster computing*, 1(1):119–132, 1998.