Accepted for publication in "Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications". Elsevier.

Grid Resource Brokering Algorithms Enabling Advance Reservations and Resource Selection Based on Performance Predictions

Erik Elmroth and Johan Tordsson

Dept. of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden.

Abstract

We present algorithms, methods, and software for a Grid resource manager, that performs resource brokering and job scheduling in production Grids. This decentralized broker selects computational resources based on actual job requirements, job characteristics, and information provided by the 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, and transfer of executable and input/output data to and from the resource. The main features of the resource broker include two alternative approaches to advance reservations, resource selection algorithms based on computer benchmark results and network performance predictions, and a basic adaptation facility. The broker is implemented as a built-in component of a job submission client for the NorduGrid/ARC middleware.

Key words: Resource broker, Grid scheduling, Runtime predictions, Performance-based resource selection, Advance reservations.

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 policies. A decentralized broker, as the one considered here, operates without global control, and its decisions are entirely based on the information made available by individual resources and index servers providing lists of available resources and aggregated resource information. For an introduction to typical resource brokering requirements and solutions, see [2–4].

The overall objective of the broker presented here is to select the resource that gives the shortest time for job completion for each job, including the time for file staging, batch queue waiting, and job execution. In order to perform this selection, the broker needs to predict the times required to perform each of these tasks for all resources considered, predictions that can be rather difficult to make due to the heterogeneity and the dynamic nature of the Grid.

The performance differences between Grid resources and the fact that their relative performance characteristics may vary for different applications makes predictions of job execution time difficult, see, e.g., [5–7]. We address this problem with a benchmark-based procedure for execution time prediction. 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. Notably, the results do not necessarily have to be for standard computer benchmarks only. On the contrary, performance results for real application codes for some test problem is often to recommend. The time prediction for staging of executable, input files, and output files are based on network performance predictions and file size information.

An advance reservation capability in the resource broker is vital for meeting deadlines for time critical applications and for enabling co-allocation of resources in highly utilized Grids. A reservation feature also provides a guaranteed alternative to predicting batch queue waiting times. Such a feature naturally depends on the reservation support provided by local schedulers [8] and the use of advance reservations also has implications on the utilization of each local

⋆ This work is a revised and extended version of [1]. It has been funded by The Swedish Research Council (VR) under contracts 343-2003-953 and 621-2005-3667, and it has been conducted using the resources of High Performance Computing Center North (HPC2N).

Email address: {elmroth,tordsson}@cs.umu.se (Erik Elmroth and Johan Tordsson).

resource [9]. For general discussions about resource reservations (and co-allocation), see, e.g., [10,11].

Our resource brokering software is mainly intended for the NorduGrid [12] and SweGrid [13] infrastructures. These are both production environments for 24 hour per day Grid usage, built on the Globus Toolkit 2 based NorduGrid/ARC (in the following denoted ARC, which is an abbreviation for Advanced Resource Connector) middleware [16]. The broker is implemented as a built-in component of an ARC job submission client. For an extension of this work towards a service-oriented framework, see [14,15].

The outline of the paper is as follows. Section 2 gives a brief introduction to the ARC software and the general resource brokering problem. The main algorithms and techniques of our resource broker are presented in Section 3. Minor extensions to the ARC user interface are presented in Section 4. Sections 5, 6, and 7 present future work, some concluding remarks and acknowledgements, respectively, followed by a list of references.

2. Background and Motivation

Our development of resource brokering algorithms and prototype implementations is mainly focused on the infrastructure and usage scenarios typical for NorduGrid and SweGrid. The main Grid middleware used is the ARC. The Grid resources are typically Linux-based clusters (in the rest of this paper, the word cluster is often used instead of the more general term Grid resource). NorduGrid includes over 50 clusters, totally comprising over 5000 CPUs, distributed over 13 countries with most of the clusters located in the Nordic countries. SweGrid currently consists of six Swedish clusters, each with 100 CPUs.

2.1. *The ARC software*

The ARC middleware is based on (de facto) standard protocols and software such as OpenLDAP [17], OpenSSL [18] and Globus Toolkit version 2 [19,20]. The latter is not used in full, as some Globus components such as the GRAM (with the gatekeeper and jobmanager) are replaced by custom components [16]. Moreover, the Globus GSI (Grid Security Infrastructure) is used, e.g., for certificate and proxy management.

The ARC user interface consists of command line tools for job management. Users can submit jobs, monitor the execution of their jobs and cancel jobs. The resource broker is an integrated part of the job submission tool, `ngsub`. Other tools allow, e.g., the user to retrieve results from jobs, get a preview of job output and remove all files generated by the job from the remote resource. Communication with remote resources is handled by a GridFTP client module.

Each Grid resource runs an ARC *GridFTP server*. When submitting a job, the user invokes the broker which uploads an xRSL job request to the GridFTP server on the selected resource. xRSL is an extended subset of the Resource Specification Language, originally proposed by the Globus project. The ARC GridFTP server specifies plugins for custom handling of FTP protocol messages. In ARC, these are used for Grid access to the local file systems of the resources, to handle Grid access control lists, and most important, for management of Grid jobs.

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 new jobs accepted by the GridFTP server. For each new job, the xRSL job description is analyzed, and any required input files are staged to the resource. Then, 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 location(s) specified in the job description. The Grid manager can be configured to, at each job state change, execute a script that intercepts (and possibly cancels) the job. These scripts form natural plugin points for e.g., accounting [21], and, as demonstrated in Section 3.2.1, for authorization of jobs that request advance reservations.

2.2. *The resource brokering problem*

One way to classify a Grid resource broker is by the scope of its operations. A centralized broker manages and schedules all jobs submitted to the Grid, whereas a decentralized broker typically handles jobs submitted by a single user only. Centralized brokers have good knowledge and control of the jobs and resources and can hence produce good schedules, but such a broker can easily become a performance bottleneck and a single point of failure. A decentralized brokering architecture, on the other hand, scales well and makes the Grid more fault-tolerant, but the incomplete information available to each instance of the broker makes the scheduling problem more difficult.

We can distinguish between two major categories of the scheduling policies used by Grid resource brokers. System-oriented brokers often strive to maximize overall system throughput, average response times, fairness, or a combination of these, whereas user-oriented scheduling systems try to optimize the performance for an individual user, typically by minimizing the response time for each job submitted by that user. This is done regardless of the impact on the overall scheduling performance and at the expense of competing schedulers. A centralized broker typically performs system-oriented scheduling whereas a decentralized broker often uses a user-oriented policy. Grid systems utilizing a centralized broker include, e.g., EMPEROR (user-oriented) [22] and Condor (system-oriented) [23]. Distributed brokers are implemented by, e.g., Nimrod-G (user-oriented) [24] and GridWay (user-oriented) [25], the latter can also be deployed as a centralized broker. For further discussions on classifications of brokers, see, e.g., [26,27].

Figure 1 illustrates a Grid with decentralized resource brokers. Each Grid resource registers itself to one or more

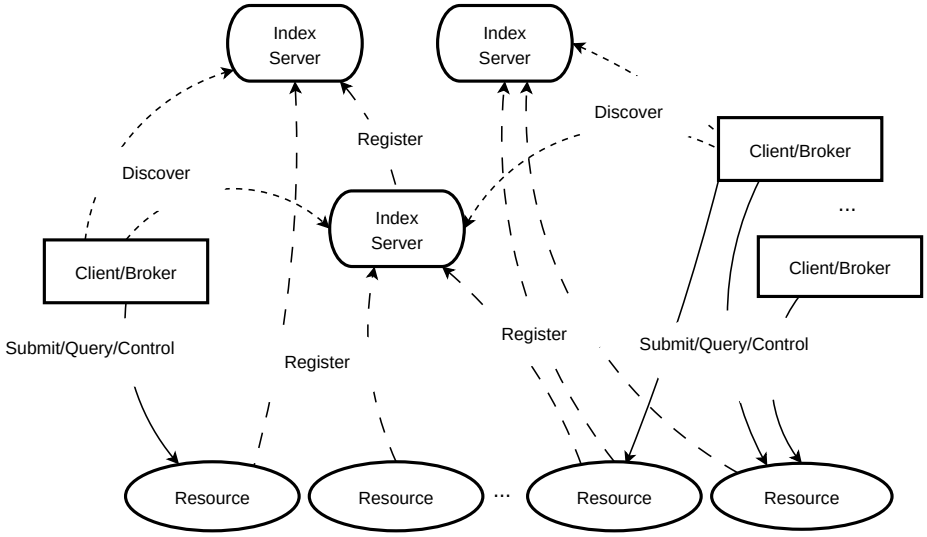

Fig. 1. Interactions between resources, index servers and brokers.

index servers, which in turn can register to higher level index servers, thus forming an index server hierarchy. All clients accessing the Grid resources use their own brokers. Each broker, optionally, contacts one or more index servers to discover what Grid resources are available. The brokers query individual clusters for detailed resource information and perform job submission and job control by communicating directly with the resources.

In the following, we focus on algorithms and software for a decentralized user-oriented resource broker. Our broker seeks to fulfill the user's resource requests by selecting the resources that best suit the user's application. In this context, selecting the most suitable resources means identifying the resources that provide the shortest Total Time to Delivery (TTD) for the job. The TTD is the total time elapsed from the user's job submission until 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).

3. Resource Brokering Algorithms

Our main brokering algorithm performs a series of tasks, e.g., it processes the xRSL specifications in the job requests, discovers and characterizes the resources available, estimates the TTD for each resource of interest, makes advance reservation of resources, and performs the actual job submission. Algorithm 1 presents a high-level outline of the tasks performed.

The input xRSL specification(s) contain one or more job requests including information about the application to run (e.g., executable, arguments, input/output files), actual job requirements (e.g., amount of memory needed, architecture requirements, execution time required), and optionally, job characteristics that can be used to improve the resource selection (e.g., listing of benchmarks with performance characteristics relevant for the application). The broker input can also include a request for advance reservations.

In Step 1 of Algorithm 1, the user's request is processed and split into individual job requests. In Step 2, the broker discovers what resources are available by contacting one or more index servers. The specific characteristics of the resources found are identified in Step 3, by querying each individual resource. Each resource may provide static information about architecture type, memory configuration, CPU clock frequency, operating system, local scheduling system, etc, and dynamic information about current load, batch queue status and various usage policies. Steps 2 and 3 are both performed by LDAP queries sent from the broker to the index servers and the resources, respectively. The actual brokering process is mainly performed in Step 4, which is repeated for each job request. In Step 5, resources are evaluated according to the requirements in the job request and only the appropiate resources are kept for further investigation. Step 6 predicts the performance of each resource by estimating the TTD, a step that may include creation of advance reservations. Then, the currently considered job is submitted in the loop started at Step 7. The loop is repeated until either the job is successfully submitted or all submission attempts fail, the latter causing the job to fail. In Step 8, the best of the (remaining) clusters is selected for submission. In Step 9, the actual job submission is performed. Steps 10 and 11 test if the submission fails, and if so, the cluster is discarded and the algorithm retries from Step 7. In Step 14, after the submission of one job is completed, any non-utilized reservations are released. Finally, in Step 16, job identifiers for all successfully submitted jobs are returned. These job identifiers are returned from successful executions of Step 9.

Notably, Algorithm 1 does not reorder the individual job requests internally (when multiple jobs are submitted in a single invocation). This can possibly be done in order

Algorithm 1 Job submission and resource brokering.

Require: xRSL-specification(s) of one or more job requests.
Ensure: Returns job identifier(s) for the submitted job(s).

```
Validate the xRSL specification(s) and create a list of all individual job requests.
Contact one or more index servers to obtain a list of available clusters.
Query each resource for static and dynamic resource information (hardware and software characteristics, current status and load, etc).
for each job do
    Filter out clusters that do not fulfill the job requirements on memory, disk space, architecture etc, and clusters that the user is not authorized to use.
    Estimate TTD for each remaining resource (see Section 3.1). If requested, advance reservations are created during this process.
    repeat
        Select the (remaining) cluster with the shortest predicted TTD.
        Submit the job to the selected resource.
        if submission fails then
            Discard the cluster.
        end if
    until the job is submitted or no appropriate clusters are left to try.
    Release any reservations made to non-selected clusters.
end for
Return the job identifier(s).
```

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. However, in the general case, factors such as local scheduling algorithms (backfilling) and competing users make the advantage of job reordering less obvious.

Figure 2 presents a sequence diagram for the tasks performed in Algorithm 1. The interactions are between the broker on the client machine, the Grid resources, and an index server, each typically running on a separate host. The broker's different interactions with a resource are performed with three different components, for requesting resource information (the index server), creating an advance reservation (any of the reservation components presented in Section 3.2), and submitting a job (the GridFTP server). It should be noted that the steps for requesting resource information, and possibly also for requesting an advance reservation, are normally performed for a number of resources before one is selected for the final job submission. In the sequence diagram in Figure 2, the GetInformation and CreateReservation operations are invoked for all resources of interest, whereas SubmitJob is called on the selected resource only.

In the following presentation, we focus on the more intricate details of performing advance reservations and the algorithms used to predict the TTD.

3.1. *Estimating the Total Time to Delivery*

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

(i) Stage in: transfer of input files and executable to the resource,
(ii) Waiting, e.g., in a batch queue and for operation (i) to complete,
(iii) Execution, and,
(iv) Stage out: transfer of output files to the requested location(s).

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 the estimated times for operations (ii), (iii), and (iv). If the time for stage out cannot be estimated due to lack of information about output file sizes, that part is simply omitted from the TTD. Below, we summarize how we make these estimates.

3.1.1. *Benchmark-based execution time predictions*

The execution time estimate needs to be based both on the performance of the resource and the characteristics of the application, as the relative performance difference between different computing resources typically varies with the character of the application. In order to do this, 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.

We remark that what is here referred to as benchmarks do not exclusively have to be standard computer benchmarks. On the contrary, for applications commonly used on the resources it is to recommend to use real application codes for this benchmarking. For example, in Swegrid, benchmarking with the ATLAS software [28] would be relevant for a

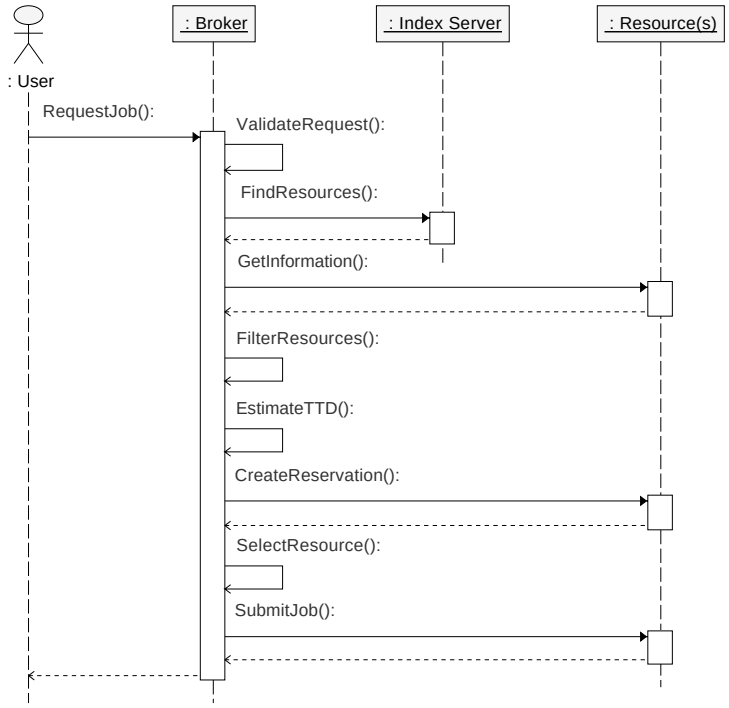

Fig. 2. Sequence diagram for job submission.

large group of high-energy physics users. An alternative to running the full application codes for benchmarking is to run small test programs representative for the application [29].

To run the benchmarking code at the time of each job submission would impose a significant overhead as one additional job (the test code) would have to be submitted, and wait in the batch queue on each resource of interest, before submission of the real job. Therefore, our approach is to run the benchmarks (e.g. test codes) once for each resource and then have the results published in the information system. As the current usage of Grid resources typically involves very large number of jobs requiring a small number of applications, this minor extra work, e.g., at time of software installation, is well motivated. In addition, a few standard benchmarks can be included for general usage, as is currently done in NorduGrid.

Based on benchmark results published by each individual resource and the user's specification of relevant benchmarks for the application, the broker makes 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.

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

If the cluster provides results for some, but not for all requested benchmarks, the broker compensates for the uncertainty by taking the corresponding time estimates for the missing benchmark(s) to be a penalty factor c times the longest execution time estimated from other benchmarks for that cluster. The default penalty factor is $c = 1.25$, but this can be reconfigured by the user.

The predicted execution time is used twice: as part of the TTD comparison during resource selection and as the requested execution time at job submission. For the TTD comparison, the average of the n execution time estimates is used as it reflects the overall resource performance with respect to the user-specified benchmarks. The maximum of the n predicted values is used as the requested execution time. This gives an accurate, yet conservative estimate of the execution time. We remark that a sufficient but short execution time estimate may lead to an earlier job start due to standard batch system scheduling algorithms. A good estimate is also more likely not the be too short, and hence reduces the risk of job cancellation by the local scheduler.

3.1.2. *File transfer time 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. If any of the input files are replicated, time estimations are

made for each copy of the file. The current version of the ARC Grid manager does however not use these estimates when selecting which copy of a replicated file to stage to the resource prior to execution, but rather choses a random replica.

The network bandwidth predictions are performed using the Network Weather Service (NWS) [30]. NWS combines periodic bandwidth measurements with statistical methods to make short-term predictions for the available bandwidth.

3.1.3. *Waiting time estimations*

The most accurate method to predict the waiting time is to create an advance reservation for the job, which gives a guaranteed start time rather than an estimate. However, a reservation based approach to predicting the waiting time cannot be used if the resource lacks support for advance reservations (via one of the mechanisms described in Section 3.2), or if the user chooses not to activate the reservation feature. In this case, the broker resorts to predicting the waiting time from the current load of the resource. This alternative prediction tends to be very coarse due to the complex nature of batch system scheduling algorithms and the limited information available to the broker about other queing jobs.

3.2. *Advance resource reservations*

The advance reservation feature makes it possible to obtain a guaranteed start time for a job, giving several advantages. It makes it possible to meet deadlines for time-critical jobs and to coordinate the job with other activities. In this section, we present two alternative approaches for supporting advance reservations, one implemented as an extension to the ARC GridFTP server and the other a service-based reservation framework.

The reservation mechanism based on GridFTP is implemented as an extension to the job management plugin in the ARC GridFTP server. The reservation protocol supports two operations: requesting a reservation and releasing a reservation. The reservation request contains the start time and requested duration of the reservation and the required number of CPUs. Upon receiving a reservation request from the broker, the GridFTP server on the resource authorizes the requestor. After authorizing the user, the job management plugin of 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 returns a unique identifier and the start time of the reservation to the broker. If no reservation can be created, a message indicating failure is returned. The GridFTP server saves the reservation identifier and a copy of the user's proxy for every successful reservation, enabling subsequent authorization of the user who made the reservation. To release a reservation, the broker uploads a release message containing the reservation identifier and the GridFTP server confirms that the reservation is released.

The service-based framework for creating and managing reservations builds on OGSI-compliant Grid services [31,32] and is implemented using the Globus Toolkit version 3 [33]. The framework consists of two services: the *ReservationFactory*, for creating reservations and the *Reservation*, which is used for controlling and monitoring created reservations. These services implement the following subset of the functionality described in [34]; two-phase reservations, i.e., reservations that have soft-state and are released shortly after their creation unless they are confirmed, and a reservation architecture not locked to a specific resource but supporting reservations of multiple resource types (computers, networks, disks, etc.). The implementation provides the resource type independence, but reservation plugin components are currently only supported for computers.

The ReservationFactory is independent of the local reservation system and uses a set of *reservation managers* to handle interactions with the reservation management system for a specific resource type (computer, network, etc.). The operation exposed by the ReservationFactory is createReservation, which takes a set of general and a set of resource specific parameters as its arguments. The general parameters include the resource type requested, a start time window specifying the range of acceptable start times for the reservation, the time when the reservation should be released unless it is confirmed, and a flag indicating whether the reservation is malleable. The start time for a malleable reservation may be altered by the local scheduler as long as it is kept within the specified start time window. Resource utilization typically decreases if advance reservations are used [9], but this impact can be reduced if the reservations are malleable [35].

The resource specific parameters have no fixed type and can be used to describe any type of requirement, e.g., the number of CPUs to be reserved on a cluster. The createReservation operation returns the exact start time of the reservation and a local reservation identifier. When createReservation is invoked, the ReservationFactory forwards the incoming request to the reservation manager of the type specified in the request. For computer reservations, the actual interaction with the local reservation manager is performed similarly as in the GridFTP based reservation system. Moreover, a copy of the user's proxy certificate is stored for later authorization.

One instance of the Reservation service is created for each reservation. The Reservation service exposes operations for querying the status of the reservation, confirming a reservation and cancelling a reservation (destroying the service). The last operation is reused from its implementation in [33].

Below, we outline a short summary of advantages and disadvantages of the two approaches to implement advance reservations.

The GridFTP-based reservation framework is light-

weight and has good performance. It is also easy to deploy, as the ARC GridFTP server already is installed on the resources and used by the broker as described in Section 2.1. On the other hand, this solution is non-standard and can hence only be used in the ARC middleware and only to reserve computational resources. Furthermore, there is no support for two-phase reservations, which causes a waste of resources if the broker for some reason is unable to explicitly release a created reservation that never will be used.

The service-based reservation framework is a more general and flexible solution that is not limited to computational resources and deployment in ARC. It is also based on standard Web services technology instead of a custom plugin to the ARC GridFTP server, and it supports two-phase reservations. The disadvantages with the service-based version include the overhead associated with service invocation and that the service-based framework requires installation of several additional software components, on both the Grid resources and in the client (broker).

3.2.1. *Job submission with a reservation*

If a reservation is successfully created on the selected resource, the broker adds the received reservation identifier to the xRSL job description before submitting the job request to the resource.

Before the Grid manager submits the job to the local scheduler, a reservation authorization plugin script analyzes the job description and detects the reservation identifier. The script inspects the saved proxy files and their associated reservation identifiers to ensure that the specified reservation exists. Furthermore, the script compares the proxy used to submit the job with the one used to create the reservation. The job request is denied unless the specified reservation exists and is created by the job submitter.

After job completion, the Grid manager may remove the reservation, allowing the user to run only the requested job. Alternatively, resources may permit the user to submit more jobs within the same reservation once the first job is completed. The configuration of the Grid manager plugin script and the local batch system determines the policy to be used.

The advance reservation feature requires that a reservation capability is provided by the local scheduler. The current implementation supports the Maui scheduler [36], although any local scheduler may be used (see, e.g., [8]). Support for other schedulers than Maui can easily be added by adapting the scripts that create and release reservations from the local batch system.

3.3. *Job queue adaptation*

Network load and batch queue sizes may change rapidly in a Grid. New resources may appear and others become unavailable. The load predictions used by the broker as a basis for resource selection can quickly become out-dated. Nevertheless, more recent information will always be available as Grid resources periodically advertise their state. To compensate for this, the broker has functionality to keep searching for better resources once the initial job submission is done. If a new resource that is likely to result in an earlier job completion time is found (taking into account all components of the TTD, including file restaging and job restart), the broker cancels the job and resubmits it to the new resource. This procedure is repeated until the job starts to execute on the currently selected resource. The job queue adaptation procedure can be viewed as the simplest form of Grid job migration, studied by e.g., [23].

```
&(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. 3. Sample xRSL request including benchmark-based execution time predictions.

4. User Interface Extensions

We have extended the standard ARC user interface with some new options and added some new attributes to the ARC xRSL, in order to make the new features available to users.

4.1. *Benchmarks*

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 can specify a benchmark result and an expected execution time on a system corresponding to that benchmark result. Notably, the expected execution time must be specified for each benchmark, as the benchmark results may be from different reference computers. This is specified using the new xRSL attribute `benchmarks`.

Figure 3 illustrates how to specify that the application requires 65 minutes to execute 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.

In order to use benchmark-based execution time predictions, the information advertised by each cluster (about hardware, software, current state etc.) must be extended with benchmark results for that cluster. Users can, however, not know in advance what benchmarks are advertised by the clusters. To simplify the usage of benchmark-based execution time prediction, there is an additional client tool

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

Fig. 4. Sample xRSL request with information required to estimate file transfer times.

for discovering all benchmarks advertised by any of the clusters. This client performs resource discovery just as the broker, but instead of submitting a job, the client outputs a list of available benchmarks. For each benchmark, a list of clusters advertising this benchmark is printed with the benchmark result for each machine. From this list, a user can find a reference benchmark result for a machine where the user's job have executed previously.

4.2. *Network transfers*

In the example in Figure 4, the job involves transfer of large input and output files. The broker determines the actual sizes of the input files when estimating the transfer time for these. The new, optional, xRSL attribute `outputfilesizes` allows the user to provide an estimate of the size of the job output. A typical user runs the same application many times and will with time normally be able to provide very accurate estimations of job output size. As shown in Figure 4, the user does not have to include size estimates for all output files in the `outputfilesizes` relation. File size estimates can be specified in bytes or with any of the suffixes kB, MB or GB.

4.3. *Command line options*

In addition to the xRSL extensions, 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 activated 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.

5. Future Work

Current and future directions of this research include development of a service-oriented stand-alone resource broker and job submission service with the same basic functionality as the current tool [14,15]. This includes investigation of how various (emerging) Grid and Web services standards can be used to improve the portability and interoperability of the job submission service, e.g., in order to facilitate cross-middleware job submission. We also plan to complement the service with additional general components, e.g., for job monitoring and control. Additional topics currently being addressed include the design and analysis of efficient algorithms for resource co-allocation.

6. Concluding Remarks

The presented resource broker is developed with focus on the ARC middleware and the NorduGrid and SweGrid production environments. Some of its brokering algorithms are currently in production use in both environments. The broker includes support for making advance resource reservations and it selects resources based on benchmark-based execution time estimates and network performance predictions. The broker is a built-in component of the user's job submission software, and is hence a decentralized user-oriented broker acting with no need for global control, entirely basing its decisions on the dynamic information provided by the resources.

7. Acknowledgements

We thank Peter Gardfjäll and Åke Sandgren for fruitful discussions. We are also grateful to the anonymous referees, whose constructive comments have improved the presentation of this work.

References

[1] E. Elmroth, J. Tordsson, A Grid resource broker supporting advance reservations and benchmark-based resource selection, in: J. Dongarra, K. Madsen, J. Wasniewski (Eds.), State-of-the-art in Scientific Computing, LNCS 3732, Springer-Verlag, 2006, pp. 1061–1070.

[2] J. Brooke, D. Fellows, Draft discussion document for GPA-WG - abstraction of functions for resource brokers, http://www.ogf.org/Meetings/ggf7/drafts/GGF7_rbdraft.pdf, May 2006.

[3] J. Schopf, Ten actions when Grid scheduling, in: J. Nabrzyski, J. Schopf, J. Węglarz (Eds.), Grid Resource Management: State of the art and future trends, Kluwer Academic Publishers, 2004, Ch. 2, pp. 15–23.

[4] R. Yahyapour, Considerations for resource brokerage and scheduling in Grids, in: G. R. Joubert, W. E. Nagel, F. J. Peters, W. V. Walter (Eds.), Parallel Computing: Software Technology, Algorithms, Architectures and Applications, PARCO 2003, Dresden, Germany, 2004, pp. 627–634.

[5] I. Foster, J. Schopf, 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, pp. 31–46.

[6] K. Ranganathan, I. Foster, Simulation studies of computation and data scheduling algorithms for data Grids, Journal of Grid Computing 1 (1) (2003) 53–62.

[7] W. Smith, I. Foster, V. Taylor, Predicting application run times using historical information, in: D. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing, LNCS 1459, Springer-Verlag, Berlin, 1999, pp. 122–142.

[8] J. MacLaren, Advance reservations state of the art, http://www.fz-juelich.de/zam/RD/coop/ggf/graap/sched-graap-2.0.html, May 2006.

[9] W. Smith, I. Foster, V. Taylor, Scheduling with advance reservations, in: 14th International Parallel and Distributed Processing Symposium, IEEE, Washington - Brussels - Tokyo, 2000, pp. 127–132.

[10] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, S. Tuecke, SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems, in: D. Feitelson, L. Rudolph, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing, LNCS 2537, Springer-Verlag, Berlin, 2002, pp. 153–183.

[11] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, A. Roy, A distributed resource management architecture that supports advance reservations and co-allocation, in: 7th International Workshop on Quality of Service, IEEE, Washington - Brussels - Tokyo, 1999, pp. 27–36.

[12] NorduGrid, http://www.nordugrid.org, May 2006.

[13] SweGrid, http://www.swegrid.se, May 2006.

[14] E. Elmroth, J. Tordsson, An interoperable, standards-based Grid resource broker and job submission service, in: H. Stockinger, R. Buyya, R. Perrott (Eds.), First International Conference on e-Science and Grid Computing, IEEE CS Press, 2005, pp. 212–220.

[15] E. Elmroth, J. Tordsson, A standards-based Grid resource brokering service supporting advance reservations, coallocation and cross-Grid interoperability, *Submitted for Journal Publication*, December 2006.

[16] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, A. Wäänänen, Advanced Resource Connector middleware for lightweight computational Grids, Future Generation Computer Systems 27 (2007) 219–240.

[17] OpenLDAP, http://www.openldap.org, May 2006.

[18] OpenSSL, http://www.openssl.org, May 2006.

[19] I. Foster, C. Kesselman, Globus: A metacomputing infrastructure toolkit, Int. J. Supercomput. Appl. 11 (2) (1997) 115–128.

[20] Globus, http://www.globus.org, May 2006.

[21] P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, T. Sandholm, Scalable Grid-wide capacity allocation with the SweGrid Accounting System (SGAS), *Submitted for Journal Publication*, October 2006.

[22] L. Adzigogov, J. Soldatos, L. Polymenakos, EMPEROR: An OGSA Grid meta-scheduler based on dynamic resource predictions, Journal of Grid computing 3 (1–2) (2005) 19–37.

[23] M. Litzkow, M. Livny, M. Mutka, Condor - a hunter of idle workstations, in: Proceedings of the 8th International Conference of Distributed Computing Systems, 1988, pp. 104–111.

[24] D. Abramson, R. Buyya, J. Giddy, A computational economy for Grid computing and its implementation in the Nimrod-G resource broker, Future Generation Computer Systems 18 (8) (2002) 1061–1074.

[25] E. Huedo, R. Montero, I. Llorente, A framework for adaptive execution on Grids, Software - Practice and Experience 34 (7) (2004) 631–651.

[26] C. Anglano, T. Ferrari, F. Giacomini, F. Prelz, M. Sgaravatto, WP01 report on current technology, http://server11.infn.it/workload-grid/docs/DataGrid-01-TED-0102-1_0.pdf, May 2006.

[27] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of Grid resource management systems for distributed computing, Software - Practice and Experience 15 (32) (2002) 135–164.

[28] L. Perini. (Coordinator), Atlas Grid computing, http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/grid/, November 2006.

[29] A. Lastovetsky, Parallel Computing on Heterogeneous Networks, 1st Edition, Wiley-Interscience, 2003.

[30] R. Wolski, Dynamically forecasting network performance using the Network Weather Service, Journal of Cluster computing 1 (1) (1998) 119–132.

[31] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling, Open Grid Services Infrastructure (OGSI) version 1.0., http://www.globus.org/alliance/publications/papers/Final_OGSI_Specification_V1.0.pdf, May 2006.

[32] I. Foster, C. Kesselman, J. Nick, S. Tuecke, Grid services for distributed system integration, Computer 35 (6) (2002) 37–46.

[33] J. Gawor, T. Sandholm, Globus toolkit 3 core - a Grid service container framework, http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf, May 2006.

[34] A. Roy, V. Sander, Advance reservation API, http://www.ogf.org/ documents/GFD.5.pdf, May 2006.

[35] U. Farooq, S. Majumdar, E. W. Parsons, Impact of laxity on scheduling with advance reservations in Grids, in: MASCOTS '05: Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, IEEE Computer Society, Washington, DC, USA, 2005, pp. 319–324.

[36] Supercluster.org. Center for HPC Cluster Resource Management and Scheduling, http://www.supercluster.org, May 2006.