A Data-Aware Resource Broker for Data Grids

Huy Le, Paul Coddington, and Andrew L. Wendelborn

School of Computer Science, University of Adelaide
Adelaide, SA 5005, Australia
{paulc,andrew}@cs.adelaide.edu.au

Abstract. The success of grid computing depends on the existence of grid middleware that provides core services such as security, data management, resource information, and resource brokering and scheduling. Current general-purpose grid resource brokers deal only with computation requirements of applications, which is a limitation for data grids that enable processing of large scientific data sets. In this paper, a new data-aware resource brokering scheme, which factors both computational and data transfer requirements into its cost models, has been implemented and tested. The experiments reported in this paper clearly demonstrate that both factors should be considered in order to efficiently schedule data intensive tasks.

1 Introduction

The term grid computing is typically used to refer to a networked virtual computing environment which extends across geographical and organisational boundaries [1, 2]. It allows a group of individuals and/or institutions, a "virtual organisation", to share resources. Resources that can be shared include computing cycles, data storage, software, and special scientific equipment. The heterogenous and dynamic nature of the grid environment makes the task of developing grid applications extremely challenging. Thus, the success of grid computing will depend on the existence of high-level frameworks or middleware to abstract over the complexity of the underlying environment, and facilitate the design, development and efficient execution of grid applications.

Broadly speaking, such a framework will have components for: high-level description of the stages of processing needed to produce desired results from available data; decomposition of that processing into tasks, or jobs, that can be deployed on grid resources; a *resource broker* mechanism for matching the requirements of these jobs with the resources available and scheduling their execution; and mechanisms for deploying jobs onto chosen resources.

In this paper, we investigate resource management and scheduling issues associated with the resource broker component, which is responsible for matching resource requirements of jobs with actual available and suitable grid resources.

Clearly, the resource broker must have ready access to up-to-date information about the grid in which to initiate a process of *resource discovery* to firstly find a set of candidate resources, and a mechanism for determining the most suitable of these candidates. A resource broker takes into account relevant properties of a given resource, typically computational properties such as processor speed, amount of memory, and processor architecture. Other properties that may be considered by a resource broker for data grids are communication bandwidth and data storage. Information is provided by information services such as the Metadata Directory Service (MDS) and Replica Catalog of the Globus Toolkit [3] and the Network Weather Service (NWS) [4].

A resource broker that uses primarily computational properties in its decision making we refer to as *compute-centric*, and one using principally data-oriented properties as *data-centric*. A compute-centric strategy is appropriate when jobs are primarily computationally intensive. A data-centric strategy may be appropriate when using a massive data set, with better performance achieved by moving code to data.

Our interest here is in general data grids, in which both computation and data are important. Many interesting applications involve geographically dispersed analysis of large repositories of measured or computed data. For example, the Belle experiment [5], at the KEK particle accelerator in Japan, is a large world-wide collaboration. It generates terabytes of experimental and simulation data, with significant data transfer costs. Computation is also important, as researchers conduct simulations and analysis. Here, the broker should make better choices by using a strategy accounting for costs of both data transfer and computation. Studies by Ranganathan and Foster [6] support this view.

However, current general-purpose grid resource brokers are compute-centric. Batch queuing systems such as PBS, LSF, Sun Grid Engine and Condor were originally developed to manage job scheduling on parallel computers or local-area networks of workstations, and therefore ignored the cost of data movement. These systems have been extended to manage job scheduling in wide-area computational grids, but still do not consider data transfer costs in their scheduling models. The Nimrod project [7] offers a novel approach to scheduling, based on an economic model that assigns costs to resource usage [8], allowing users to trade-off execution time against resource costs, but data transfer is not considered by the scheduler.

The AppLeS (Application Level Scheduling) project [9] approaches the scheduling problem at the application level, dynamically generating a schedule based on application-specific performance criteria. Data properties can be used, but scheduling logic is embedded within the application itself, an approach not easily re-targeted for different applications or execution environments [10]. Our work aims to create a generic resource broker for any grid application.

After the work reported in this paper had been completed, similar work on data-aware resource brokers was published. The approach of the Gridbus Broker [11] is that when a job is being scheduled, if there is a choice of compute nodes of similar capability, the scheduler will choose the one with the highest bandwidth connection to a copy of the input data. Version 1.1 of the JOSH system [12] for Sun Grid Engine also takes into account data transfer in its scheduling decisions.

The primary objective of this investigation is to show that application performance can be improved by making scheduling decisions that take into account not just the computational performance of resources, but also data transfer factors. To this end, we discuss the design, implementation, measurement and evaluation of a new, data-aware resource broker.

2 The Data-Aware Resource Broker

The Data-Aware Resource Broker (DARB) is a generic resource broker, designed specifically with the needs of data-intensive applications in mind. DARB undertakes resource discovery, that is, finding resources in the grid environment that match the needs of the application; and resource selection, that is, choosing between alternative data, software and hardware resources.

For this investigation, DARB was implemented in Java within the Gridbus framework [13]. Gridbus is an extension of Nimrod/G [7], which supports execution of parameter sweep experiments over global grids. Such studies are ideal for the grid, since they consist of a large number of independent jobs operating on different input data. Nimrod/G and Gridbus provide a simple declarative parametric modelling language and GUI tools for specifying simulation parameters. A job is generated for each member of the cross product of these parameters. The system manages the distribution of the various jobs to resources and organises the aggregation of results.

2.1 Architecture and Implementation

Our experimental framework comprises three key components, shown in Figure 1.

1. The Job Submission Component provides a mechanism for users to submit applications for execution.
2. The Scheduling Framework is the resource broker component, responsible for allocating the user-submitted jobs to available resources in an efficient manner.
3. The Job Dispatching Component interfaces with the underlying grid middleware, dispatching and monitoring jobs.

Here we focus on the scheduling framework, which we have implemented using DARB. For these experiments, we use the job submission and dispatching modules provided by Gridbus. DARB interfaces with the Gridbus system, however it is important to note that the DARB resource broker is generic, and can be "plugged in" to other grid environments.

DARB consists of three modules:

1. The Grid Information Module regularly queries the MDS and NWS for state information of the grid nodes and network links. This information is used to build a model of the current grid environment, representing the grid as a graph structure in which grid nodes are represented by vertices and communication links by edges.

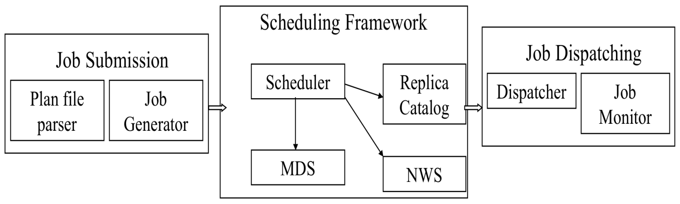

Fig. 1. System Architecture

2. The Replica Catalog Module provides a convenient interface for accessing information on file locations using the Globus Replica Catalog. It is used to support data-aware scheduling.
3. The Scheduler Module uses information from the Grid Information and Replica Catalog modules to map jobs (generated by the Initializer Module) to appropriate grid nodes (using the Dispatcher Module).

To form a complete system, these modules are used in conjunction with Initializer, Dispatcher and Job Monitor modules provided by the Gridbus system.

2.2 The Scheduler Module

The data grid environment which the DARB targets consists of distributed sites that have both data stores and computational capabilities. Input data sets for applications are replicated among these sites. The Globus Replica Catalog provides a mapping between the logical name of a data set and its physical locations. Each site in the environment may have different computational capabilities and data stores, which will change over time. In such an environment, it is vital for a resource broker to be provided with information about these characteristics, so it can and factor them into an appropriate performance model.

In the Scheduler module, MDS and NWS information from the Grid Information module is used to determine the current state of the grid environment. The Replica Catalog is used to determine the locations of a data file requested by an application. In order to arrive at a schedule for an application, DARB takes into account the machine architecture, clock speed, and number of CPUs; the CPU load; the network bandwidth; the size of the input data (obtained from the Replica Catalog); and the amount of computation involved.

The total time to execute a job on a particular compute node comprises of two components: data transfer time and computation time. If the compute node already has the input file, data transfer time will be zero. Otherwise, the time to transfer the input file is calculated using the bandwidth forecast obtained from the NWS. Experiments have shown that latency is negligible for the transfer of large data files, and as such, the latency forecast from NWS is not factored into the calculation. We use an estimate, based on input data size and bandwidth information (from the Grid Information Module), of expected transfer time.

A correction is applied in the calculation of the transfer time, since our experiments showed that the NWS tends to return bandwidth forecasts that are less than the actual value. As noted by Primet et al. [14], NWS transfers small amounts of data (default 64KB) which leads to very short transfer time on fast networks. Thus, any error in the time determination, due to the TCP slow start mechanism or other factors, will strongly influence measured bandwidth values. Primet et al. suggested the complementary use of additional tools to conduct less frequent probe experiments using larger amounts of data. An alternative approach is taken in DARB, which relies on measurements of data transfer times from the execution of previously submitted jobs.

The expected computation time is taken from a user-supplied estimate of job execution time on a given architecture, adjusted by the current load on a candidate node, and the number of CPUs at that node. In parameter sweep applications, the user estimate could be refined based on measured times for previous executions of the same program, however this has not yet been implemented.

2.3 The Scheduling Algorithm

Parameter sweep applications involve N independent jobs (each with the same task specification, but a different input file) on M distributed nodes, where N is typically much larger than M. In order to efficiently map these jobs to the nodes, it is important to consider both availability of computational resources and location of required data. A balance needs to be reached between minimizing computation and data transfer times. In general, if computational resources are available, it is preferable to execute the jobs where data already exists in order to minimize network traffic. However, when nodes containing data are busy and there are idle nodes elsewhere, the cost of data transfer may be outweighed by the increase in job throughput if the jobs are rescheduled to the free nodes.

Initially, the Data-Aware Resource Broker attempts to allocate the jobs to the nodes that already contain the input data. The available CPU value of each node is monitored to ensure that they are not overloaded with jobs. This process continues until it is no longer optimal in terms of overall throughput for the nodes containing the data to execute the jobs. This occurs when there are idle nodes and the expected execution time on one of those nodes is less than the expected execution time on the node containing the data. If this is the case, the resource broker retrieves the list of nodes that are available and calculates the expected completion time for the particular job on each node. The expected execution time of a job on a particular node is calculated via the following steps:

1. retrieve the list of nodes containing the input file required by the job;
2. select, from the above list, the best node in terms of bandwidth to the node that the job is to be executed on;
3. calculate the expected execution time, by adding expected execution and transfer times.

The job is rescheduled to the node that gives the best expected execution time.

3 Experimental Evaluation

This section presents the results of some initial experiments to evaluate the performance of the Data-Aware Resource Broker, and compare it with the standard compute-centric and data-centric resource broker strategies. The experiments are based on the requirements of the Belle data grid, but would be common to many scientific applications.

It is assumed that there are a large number of data files distributed across the nodes of the grid, and a researcher wants to run a data processing program on each of a specified subset of these data files. The researcher may even want to run many instances of the program for each data file, using different input parameters. It is assumed that all required jobs can be run independently, and the researcher's goal is to minimize the total turnaround time in executing all of these jobs. The resource broker should therefore schedule the jobs to the available nodes so that the job throughput is maximized. Hence, performance is measured in terms of total elapsed (wall clock) time to execute a specified set of jobs.

In some cases the data files to be processed may be distributed fairly regularly across nodes in the grid, for example in the Belle data grid these could be the output of Monte Carlo simulations of the Belle experiment, which are generated on each of the nodes and might be stored on the node on which they were generated. In other situations the distribution of the data among grid nodes may be highly irregular, for example data from a physics experiment such as Belle may be stored on a single server at the experiment's location, with replicas of the data located at a relatively small number of other nodes.

The experiments were performed on the Australian Belle Data Grid testbed composed of five compute server nodes located at the the University of Melbourne, the University of Sydney, the University of Adelaide, and the Australia National University in Canberra. Two machines were located in Melbourne, the rest were hundreds of kilometers apart. All of the servers used Intel processors running Linux, with a single 2.8 GHz Xeon processor at Adelaide and Sydney, dual 2.8 GHz Xeons at Melbourne and Canberra, and a single 2 GHz Pentium 4 processor at Melbourne. The average network bandwidth between the different machines varied considerably, from about 2 Mbits/sec (Adelaide to Canberra) to 15 Mbits/sec (Sydney to Canberra) for machines in different cities, and around 60 Mbits/sec between the two Melbourne nodes. We implemented a single clique containing all the nodes of our testbed for the purpose of running NWS sensors.

A simulated application, called DataSim, was developed for initial testing of DARB. It is a simple program which reads from an input file, performs some computation, and writes to an output file. The program does not require all the input to be available when it starts. Computation can proceed as soon as there is a specified amount of input to be processed, i.e. it can support data streaming. Several factors are customisable via its arguments, including the compute/input ratio (how much computation is performed per input element); the output/input ratio (how much output is written per input element); and whether input and output are streamed.

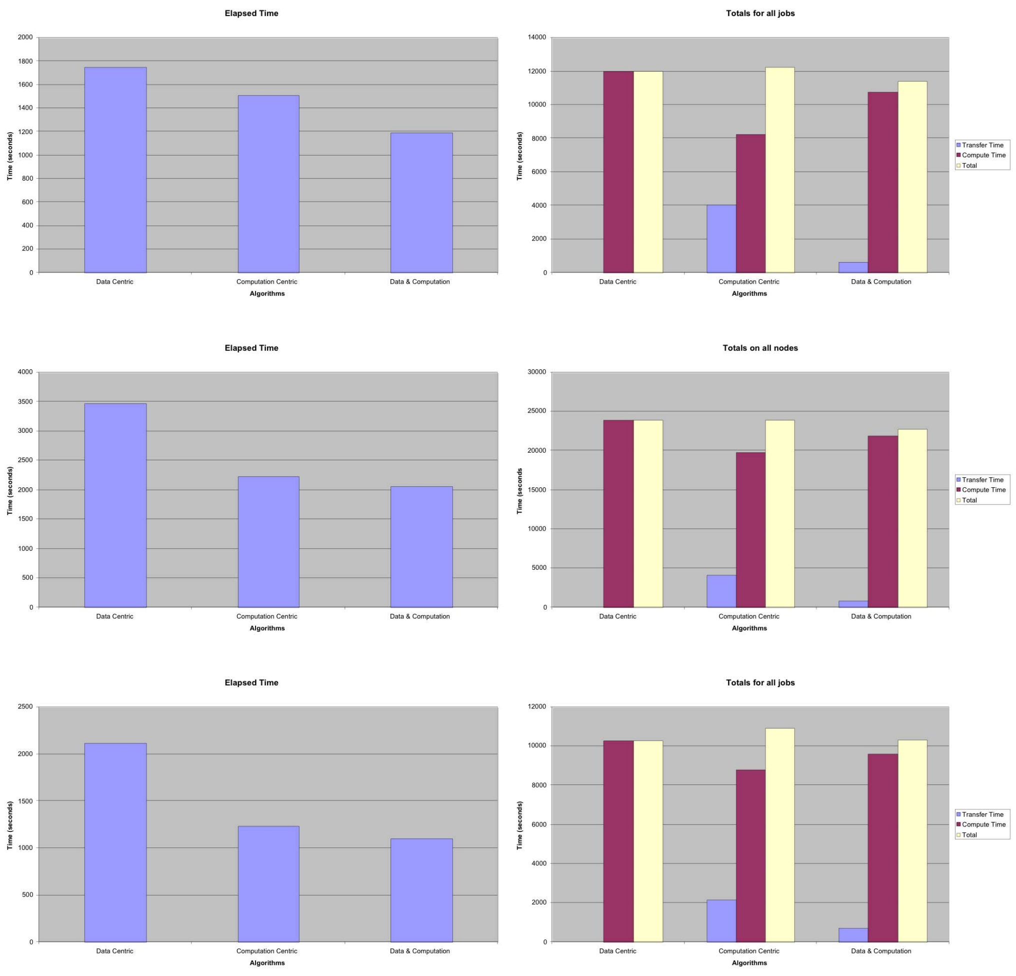

Fig. 2. Total execution times for all 40 runs of the DataSim application with 80 MB input files using compute-centric, data-centric or data-aware resource brokers. The first column gives the total elapsed wall clock time, while the second column is the aggregated time for computation and communication summed over all nodes. Results for even distribution of files are shown in the first row (compute/input ratio of 5) and second row (compute/input ratio of 10). The third row gives results for uneven distribution with compute/input ratio of 10.

Each experiment consisted of 40 runs of the test application for a particular set of parameters, with four runs for each of 10 different input files that were distributed across all five nodes in the grid testbed. This was repeated 10 times for each experimental scenario, to get an average result. The following parameters were varied in the different scenarios:

– **Size of input files:** 40Mb and 80Mb input files were used in the tests. Obviously data file size varies greatly between different applications, however varying the value gives an indication of sensitivity to input file sizes.
– **Distribution of input files:** Both even and uneven distributions of input files were tested. For the uneven distribution, the Adelaide and Canberra nodes contained 60% percent of the input files while the other 40% percent were distributed among the three nodes in Sydney and Melbourne.
– **Computation/Input ratio:** This controls how many times a certain computation is performed for each element of input. Values of 5 and 10 were used in the experiments. The difference between the DARB and compute-centric algorithms will not be apparent for higher values of computation/input ratio since data transfer time will be insignificant. If the ratio is set too low, communication time will dominate and results for DARB will approach those of the data-centric algorithm.

In calculating the network transfer times, it was assumed that the output files do not have to be transferred back to the client or a specific data storage node. These possibilities will be explored in future work.

Figure 2 shows results for the scenarios listed above, for the 80 MB input data files. The results for the 40 MB input files are very similar, although as would be expected, the results for the data-centric approach are worse while the results for the compute-centric approach are closer to those produced by DARB. More detailed results for all of the experiments are available in Le's thesis [15]. The most important measure is the elapsed wall clock time for the completion of all the jobs, which is shown in the first column of Figure 2. The second column shows the aggregated sum of compute times, data transfer times, and total times for job execution on each node, which gives an indication of the tradeoffs in data transfer and compute times for each algorithm. By considering both computational capability and cost of data transfers, the DARB approach gave the lowest total elapsed time in all the experiments.

As expected, the data-centric approach did better when the input file sizes were larger and/or the computation/input ratio was lower. By always scheduling jobs to nodes which already contain the data, the data-centric algorithm left many of the more computationally powerful nodes idle while the slowest node was completing its allocation of jobs. This led to significantly poorer job throughput.

The compute-centric approach did better when the input file sizes were smaller and/or the computation/input ratio was higher. Even in the cases when the compute-centric approach did almost as well as DARB for total execution time, the total data transfer time for the compute-centric algorithm was several times higher than that for the DARB algorithm. In situations where network costs are based on traffic volume, this could add significantly to the monetary cost of the computations. This can also lead to network congestion, and it was noted in some experiments that the large amount of network traffic caused certain connections to become bottlenecks, leading to significant variation in performance in the compute-centric algorithm.

In the case where the data is irregularly distributed, the data-centric approach does relatively worse. This is a problem since the data-centric model is perhaps most likely be used in situations with a relatively small number of servers having copies of very large data sets with large data transfer costs.

4 Conclusion

We have described the design, implementation and evaluation of DARB, a data-aware resource broker designed to efficiently handle the allocation of jobs to available resources in a data grid environment. Information from NWS and the Globus MDS and Replica Catalog are used to predict likely computation and data transfer times for each job, which are then used to generate a schedule for the jobs that tries to maximize throughput.

The initial experiments reported here showed that DARB outperformed algorithms based only on computational requirements or data location, and hence it is necessary to take both into account for efficient job execution on a data grid. DARB also resulted in significantly reduced network traffic compared to a standard compute-centric approach.

This Gridbus Broker [11] has an advantage over DARB, since it does not require accurate estimates of the compute time or the data transfer time. However where such estimates are available, we would expect DARB to perform better, since the Gridbus Broker would not handle the situation where a slower node might be a better option due to much faster data communication times.

The performance of DARB has so far only been tested using a small number of experimental scenarios on a small grid testbed using a simulated program. In future work, further experimentation using DataSim will investigate a wider range of input file sizes and computation/input ratios, different numbers of data files, multiple input files, and a variety of distributions and replications of data files. DARB will also be evaluated on a larger and more heterogenous grid testbed, and using a variety of real data grid applications.

Currently DARB assumes that output files remain where they are computed. However, outputs are often needed elsewhere, such as the client, or a data repository. DARB can easily be extended to take into account transfer of output files, but only if the size of these files is known beforehand. This could be specified by the user, or estimated based on previous runs.

Currently the performance of the DARB is dependent on the accuracy of the user's estimate of computation time for different architectures and and clock speeds. A better approach would be to keep a profile of past execution times for each node, which could easily be done in an environment for parameter sweep applications, such as Nimrod/G or Gridbus.

Acknowledgements

Four of the machines used for the experimental evaluation of DARB were generously provided by IBM. We are very grateful to the Gridbus group, especially Srikumar Venugopal and Rajkumar Buyya, for help with Gridbus and discussions on scheduling, and also the Belle Data Grid group, particularly Lyle Winton, for help with Globus problems and general grid issues.

References

1. Foster, I., Kesselman, C., eds.: The Grid: Blueprint for a Future Computing Infrastructure. Morgan-Kaufmann (1999)
2. Foster, I.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications (2001)
3. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Intl. Journal of Supercomputer Applications **11** (1997)
4. Wolski, R., Spring, N., Hayes, J.: The Network Weather Service: A distributed resource performance forecasting service for metacomputing. Journal of Future Generation Computing Systems **15** (1999) 757–768
5. KEK: The Belle experiment. http://belle.kek.jp/ (2003)
6. Ranganathan, K., Foster, I.: Decoupling computation and data scheduling in distributed data intensive applications. In: Proc. of 11th IEEE Int. Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh (2002)
7. Abramson, D., Giddy, J., Kotler, L.: High-performance parametric modelling with Nimrod/G: Killer application for the Global Grid? In: Proc. of IPDPS 2000, Cancun, Mexico (2000)
8. Buyya, R., Abramson, D., Giddy, J.: An economy driven resource management architecture for global computational power grids. In: Proc. of Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas (2000)
9. Berman, F., et al.: Adaptive computing on the grid using AppLeS. IEEE Transactions on Parallel and Distributed Systems **14** (2003) 369–382
10. Su, A., Berman, F., Wolski, R., Strout, M.: Using AppLeS to schedule simple SARA on the computational grid. International Journal of High Performance Computing Applications **13** (1999) 253–262
11. Venugopal, S., Buyya, R., Winton, L.: A grid service broker for scheduling distributed data-oriented applications on global grids. Technical Report GRIDS-TR-2004-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne (2004) http://www.gridbus.org/techreports.html.
12. Sun Data and Compute Grids Project: JOb Scheduling Hierarchically (JOSH). http://gridengine.sunsource.net/project/gridengine/josh.html (2003)
13. Buyya, R.: The Gridbus Project. http://www.gridbus.org/ (2003)
14. Primet, P., Harakaly, R., Bonnassieux, F.: Experiments of network throughput measurement and forecasting using the Network Weather Service. In: Proc. of 2nd IEEE/ACM Int. Symposium on Cluster Computing and the Grid (CCGRID'02), Berlin (2002)
15. Le, H.: The Data-Aware Resource Broker: A resource management scheme for data intensive applications. Technical Report DHPC-140, Distributed and High-Performance Computing Group, University of Adelaide (2003) http://www.dhpc.adelaide.edu.au/reports/140/abs-140.html.