An Economy-based Algorithm for Scheduling Data-Intensive Applications on Global Grids

Srikumar Venugopal and Rajkumar Buyya
Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
Email:{srikumar,raj}@cs.mu.oz.au

***Abstract*— Data Grids have become the de facto platform for the next generation of eScience experiments that will be carried out through large collaborations spread around the world. As the number of entities within a data grid increases, scheduling of applications in order to make the most efficient use of the available resources such as computational, storage and network facilities becomes a challenge. Previous work has suggested a computational economy metaphor for resource management within compute and data grids. However, the issue of scheduling jobs that require distributed data within an economy-based data grid has not been studied in detail so far.**

In this paper, we present a model and an algorithm for economy-based scheduling of distributed data intensive applications on data grids. The model takes into account the costs and times for transferring datasets required for a job from different data hosts to the compute resource on which the job will be executed and for its processing. The algorithm builds a resource set for a job that minimizes the cost or time depending on the user's preferences. We evaluate the algorithm on a Data Grid testbed and present the results.

I. Introduction

Data-intensive applications in areas such as high-energy physics, astronomy and bioinformatics are revolutionising the methodology of scientific computing. Executing these eScience [1] applications require mechanisms different from compute-intensive applications because of the requirements for access, storage and management of large distributed datasets. Thus, these create an environment in which data is as important as computation if not more [2].

Data Grids [3] have evolved to tackle the challenges of such data-intensive computing environments. Primarily, they provide mechanisms for replication and a high-speed transport layer. There are many projects around the world setting up data grids for specific scientific application domains [4][5][6]. Users, such as scientists, have access to the aggregated computational and data resources within such a grid project.

A. A Case for Economy in Data Grids

A data-intensive computing environment can be perceived as a real-world economic system wherein there are producers and consumers of data distributed geographically across multiple organisations. Producers are entities which generate the data and control its distribution via mirroring at various replica locations around the globe. They lay down policies for replication that are guided by various criteria such as minimum bandwidth, storage and computational requirements, data security and access restrictions and data locality issues. An example of such a system would be the tier-level model proposed by the MONARC [7] group within CERN for replicating the data produced by the Large Hadron Collider (LHC) [8] for use within the ATLAS and CMS collaborations. The consumers in this system would be the users or, by proxy, their applications which need to analyse this data to produce meaningful results. The users may want to investigate specific datasets out of a set of hundreds and thousands and may have specific application requirements that need not be fulfilled at every computational site.

In such large collaborations, there can be a lot of pressure on the data infrastructure (i.e. network and storage elements). The pressure becomes more acute when a non-trivial percentage of the users are interested in the same datasets simultaneously, thus causing heavy load on the servers on which the requested datasets and its replicas are hosted and denying service to requestors of other datasets on those servers. Such an effect is commonly observed in the Internet and the World Wide Web [9].

While a robust and adaptive replication mechanism can alleviate some of the above problems, the same problems of data access and transfer costs affect the effectiveness and efficiency of such a mechanism. Pricing resources to reflect supply and demand in order to regulate their usage has been explored in previous work [10]. On the consumer side, the user would specify his deadline for the analysis job, his budget and his preference for the cheapest or the fastest processing according to his needs and priorities. While studies have shown this to be effective for resource management in computational grids [11], no study has been made so far on the economic aspects of data processing by scheduling analysis jobs on various sites with varying execution, transfer and storage costs. In this paper, we propose a model and an algorithm for economy-based scheduling within Data Grids.

The rest of this paper is organised as follows. In Section II, we survey previous work in data grid scheduling and economy-based replication mechanisms. In Section III, we extend the notion of user-driven deadline and budget constrained scheduling within computational grids to data grids. In Section IV, the proposed algorithm is evaluated on a real Grid testbed and the results are reported. Finally, we conclude our paper and

outline the future work.

II. Related Work

Scheduling data intensive applications over wide-area networks has received lot of attention in the recent years. In [12], the authors evaluate various heuristics for parameter-sweep jobs which have files as input. They introduce a new heuristic, *XSufferage*, that takes into account file locality by scheduling jobs to those clusters where the files have already been transferred for a previous job. However, a restriction within this work is to limit the source of the files to the host which submits the jobs for execution. Ranganathan and Foster [13] have simulated job scheduling and data scheduling algorithms and recommend that it is best to decouple data replication from the job scheduling. In previous work [14], we have proposed an adaptive algorithm that schedules jobs while minimizing data transfer. It evaluates all known replica locations of the file and submits the job to the compute resource which is located closest to one of the replica locations. However, in all the works presented above, economic costs of transfer and processing of data have not been taken into account while scheduling.

Several research works have explored the use of economy within data driven computing environments. Mariposa [15] was one of the earliest systems that experimented with an economy paradigm for query processing and storage management. Within Mariposa, a budget is associated with each query and its execution is conducted through a bidding process. There is no explicit scheduling within this system. While in our economic model there is a budget associated with each job, the job allocation is done by the scheduler which looks at the costs associated with using the compute and data resources and the user's deadline and budget and scheduling preference.

More recently, Stockinger et.al. [16] discuss a cost model for replicating data within Grid environments with particular emphasis on the CERN LHC experiment requirements. However, their cost model focuses on system and application dependent factors while our work focuses on the economic costs of data processing while taking into account many of the factors such as bandwidth, degree of replication of data and data server load that they have also considered. In addition, they pay more attention to data replication and do not explicitly consider the processing requirements of the application. Our focus is on the latter and we consider replication to be decoupled from the processing and done independently by the data service providers.

In [17], the authors propose an economy-based replication strategy based on a Vickerey auction to determine the optimal replica location to fetch a file that a job has requested. The file is then replicated at the local host so that other requests in the future emerging from the vicinity can be fulfilled. The economic considerations are limited to the data replication and the processing costs for the job are not considered in their model.

Economy-based resource management and scheduling in computational grids was proposed and evaluated in [18]. An important limitation to that work is the lack of consideration given to data while scheduling jobs on remote resources. This paper aims to extend the deadline and budget constrained cost and time minimization algorithms proposed in [18] to data grids by removing that limitation.

III. Scheduling

A. Model

Fig. 1 shows a typical data grid environment that is composed of storage nodes that store the data and compute resources that run the jobs that analyse the data. It is possible that the same node may contain both storage and computation capabilities. For example, it could be a supercomputing center which has a Mass Storage Facility attached to it. The datasets may be replicated at various sites within this data grid. We assume that the replication is carried out independently and is dependent on the policies set by the administrators of the storage resources and/or the producers of data. We also assume that the scheduler is able to query a data directory such as a Replica Catalog [19] or the SRB [20] Metadata Catalog that would contain information about the locations of the datasets and their replicas. We associate economic costs with the access, transfer and processing of data. The processing cost is levied upon by the computational service provider, while the transfer cost comes on account of the access cost for the storage node and the cost of transferring datasets from the storage node to the compute resource through the network.

We consider a model for scheduling independent jobs on a data grid. Independent jobs arise in execution models such as parameter-sweep model of computation. Parameter-sweep applications(PSA) are common within scientific studies and have been considered as being very suitable for Grid execution [21] because jobs have no intercommunication dependencies. We consider a job as the atomic unit of computation within this model. The steps for submitting a job to the grid shown in Fig. 1 are as follows: The scheduler gathers information about the available compute resources through a resource information service (1) and about the data through the data directory (2). It then makes decision on where to submit the job based on the availability and cost of the compute resource, the minimization preference and the location, access and transfer costs of the data required for the job (3). The job is dispatched to selected remote compute resource (4) where it requests for the dataset from the replica location selected by the scheduler (5 & 6). After the job has finished processing (7), the results are sent back to the scheduler host or another storage resource which then updates the data directory(8). This process is repeated for all the jobs generated by a PSA.

We consider, therefore, a set of N independent jobs $J = \{j_1, j_2, \ldots, j_N\}$ that have to be scheduled on M computational resources $R = \{r_1, r_2, \ldots, r_M\}$. Typically, $N \gg M$. Assume job $j, (j \in J)$ is scheduled to be executed on computational resource $r, (r \in R)$. Each job requires a set $F_j = \{f_{j1}, f_{j2}, \ldots, f_{jK}\}$ of K datasets that are each replicated on a subset of P data hosts,$D = \{d_1, d_2, \ldots, d_P\}$.

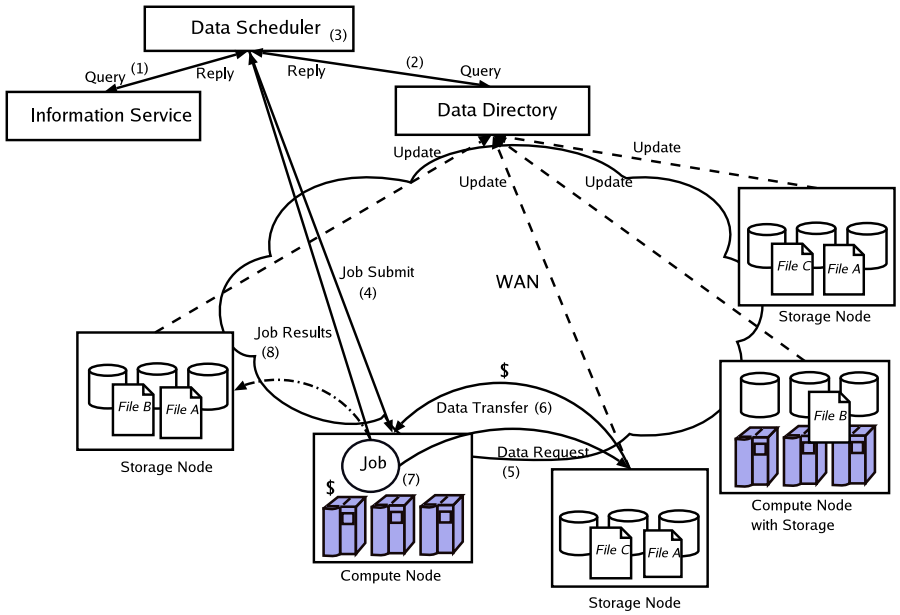


Fig. 1. An economy-based data grid environment

For $f_{jk} \in F_j$, $D_{jk} \subseteq D$ is the set of datahosts from which dataset f_{jk} may be obtained.

Therefore, the time taken to execute the jobs is the sum of the execution time and the times taken to transfer each of the K files from the respective storage nodes to the compute node. That is, if the computation time is denoted by t_{jr} and the transfer time for the k^{th} dataset f_{jk} is denoted by $t_{f_{jk}r}$, then the total time required for executing the job j,

$$t_j = t_{jr} + \sum_{k=1}^{K} t_{f_{jk}r}$$

where

$$t_{f_{jk}r} = Response\_time(d_{jk}) + Size(f_{jk})/BW(Link_{d_{jk}r})$$

In the above equation, d_{jk} is the data host from which f_{jk} will be obtained. $Response\_time(d_{jk})$ is the difference between the time when the request was made and the time when the first byte of the f_{jk} is received at r. This is function of the load on the data host. If it is heavily loaded then the response time will be higher. $BW(Link_{d_{jk}r})$ is the available bandwidth for the network connection between the data host d_{jk} and the compute resource r.

To calculate the economic cost of executing the job, we denote the processing cost of the job j on the compute node r by e_{jr} and cost of transferring the dataset f_j by $e_{f_{jk}r}$. Therefore, the total execution cost for job j,e_j is given by

$$e_j = e_{jr} + \sum_{k=1}^{K} e_{f_{jk}r}$$

where

$$\begin{aligned} e_{f_{jk}r} &= Access\_cost(d_{jk}) \\ &\quad + Size(f_{jk}) * Cost\_per\_unit\_size(Link_{d_{jk}r}) \end{aligned}$$

$Access\_cost(d_{jk})$ is the cost of requesting a dataset which is levied by the data host. It can be an increasing function on either the size of the requested dataset or the load on the data host or both. This cost regulates the size of the dataset being requested and the load which the data host can handle. $Cost\_per\_unit\_size(Link_{d_{jk}r})$ is the cost of transferring a $unit\_size$(eg. MB or GB) of the requested dataset through the network link between the data host and the compute resource. The cost of the link may increase with the Quality of Service(QoS) being provided by the network. For example, in a network supporting different channels with different QoS as described in [22], the cost of a faster link may be higher than that of a slower link. Hence, the file is transferred faster but at a higher expense. We consider all traffic within a Local Area Network(LAN) to be essentially free, that is, no cost is levied upon them.

The model given above is similar to the model described within [16] with two major differences. One, as stated in Section II, is their focus on data replication and the other is the fact that they have considered bandwidth of the LAN in their model. We consider the WAN bandwidth and cost as to be dominant and therefore, we have ignored LAN parameters within our model.

We associate two constraints with the schedule, the deadline by which the entire set must be executed (denoted by $T_{Deadline}$) and the maximum budget, $Budget$, for processing the jobs. The deadline constraint can therefore be expressed in terms of job execution time as $max(t_j) \leq T_{Deadline}, \forall\ j \in J$. The budget constraint can be expressed as $\sum_J e_j \leq Budget$.

B. Algorithm

Depending on the user-provided deadline, budget and scheduling preference, we can have two objective functions,

viz:

- **Cost minimization** We try to execute the jobs in the schedule that causes least expense while keeping the execution time within the deadline provided.
- **Time minimization** Here, the jobs are executed in the fastest time possible with the budget for the execution acting as the constraint.

It is obvious that in both cases the same algorithm can be applied to solve the different objective functions. We, therefore, introduce a greedy algorithm to schedule the set of jobs. This algorithm is based on the Min-Min heuristic discussed in [23].

The listing for the algorithm is given in Figure 2.The scheduling algorithm consists of two parts, one is the mapping of the jobs to the resources (lines 10-27) and the second is the actual dispatching of the jobs to the resources itself(lines 30-48). The mapping part deals with creating a set of resources consisting of one compute node and one data host each for every dataset required by the job. That is, for each job j, we create a **resource set** $S_j = \{r_j, d_{j1}, d_{j2}, \ldots, d_{jK}\}$ that represents the compute and data resources to be accessed by the job in execution. In the dispatching part, we ensure that the deadline and budget constraints on the schedule are enforced while submitting the jobs to the remote resources.

J_U, J_A, J_C and J_F are subsets of J consisting of jobs in $Unsubmitted$, $Active$, $Completed$ and $Failed$ states respectively. A job can be in only one of these states at a time. The scheduling algorithm completes when all jobs are either in $Completed$ or $Failed$ states.

We introduce a variable Min which allows us to change the decision variables depending on the minimization chosen within the algorithm. We define a function f_{min} that returns the smallest value within A depending on the minimization applied. Formally,

$$f_{min}(Min, CVar, TVar, A) = \begin{cases} min(CVar, A) \\ \text{if } Min = Cost \\ min(TVar, A) \\ \text{if } Min = Time \end{cases}$$

Here, $CVar$ and $TVar$ represents variables deal with cost and time respectively. $min(CVar, A)$ and $min(TVar, A)$ will return the element of A with the smallest value of $CVar$ and $TVar$ respectively. Hence, by changing the value of Min we can determine what objective function the algorithm will minimize. Consequently, Min is a parameter to the scheduling function.

At every polling interval, we update the performance data of the compute resources by taking into account the jobs that have been completed or failed since the last polling interval. Based on this, we compute the $AvgComputationTime$ required for a job at each server. Since within a set, the jobs are similar to each other in terms of processing requirements, we can safely assume that the average computation time holds true for the remaining unsubmitted jobs. For each data resource, we update the network conditions between itself and the computational resources. Then, we sort the computational resources either by the cost of the network link or the bandwidth between the compute resource and the data host depending on the minimization required. The advantage of this step will be explained shortly.

In the mapping section, we try to build the resource set S_j by selecting a data resource for each dataset required by the job and a compute resource, optimally situated from the selected data resources, for executing the job. The best way to go about this would be to try all possible combinations of compute and data resources till we reach at the right combination. This, however, increases the worst-case running time of the mapping loop to the exponential order of K .

We, therefore, decrease the complexity by making a greedy choice at each step within the mapping section. For a job, we iterate through the list of datasets it requires. For each dataset, we select the best data host to retreive the file from in terms of access cost or response time depending on the minimization applied (line 13). Then, for a selected data resource, we choose the best compute resource based on the cost of the network or the available bandwidth to the resource (line 14). Since the list of compute resources has already been sorted for each data resource, we only have to take the first element of the sorted list. However, this may not be the best or even close to it for the entire resource set including the data hosts selected in the previous iterations. Therefore, we create two resource sets, S_j and S'_j, the former with the current selected compute and data resources and the latter with the current selected data resource but with the compute resource selected in the previous iteration (lines 15 - 22) . Then, we compare the two sets on the basis of the expected cost or execution time and select the resource set which gives us the minimum value (line 23). This procedure ensures that the choice of compute resource we make with the current dataset does not worsen the optimality with respect to the other datasets and that the resource set selected at the end of each iteration is better than that selected in all previous iterations.

In the dispatching section, we first sort all the job in the ascending order of the value of the minimization function for their respective combinations. Then, starting with the job with the least cost or least execution time, we submit the jobs to the compute resources selected for them in the mapping step. For cost minimization, we see if the deadline is violated by checking whether the current time($T_{Current}$) plus the expected execution time exceeds $T_{Deadline}$ (line 35). If so, the job goes back into the unsubmitted list in the expectation that the next iteration will produce a better combination. If $Budget$ is exceeded by the current job then we stop dispatching any more jobs and return to the main loop since the rest of the jobs in the list will have higher cost (lines 36-37). For time minimization, we check if the budget spent (including the budget for all the jobs previously submitted in current iteration) plus the budget for the current job exceeds $Budget$. If the deadline is violated by the current job then we stop dispatching and return to the main loop.

We analyse the worst-case complexity of the mapping loop as follows: The data resource selection step is $O(P)$ since

```
1  while J ≠ J_C ∪ J_F OR T_current < T_Deadline OR Budget_spent < Budget do
2      Update Budget_spent by taking into account the jobs completed in the last interval;
3      for each r ∈ R do
4          On the basis of jobs completed in last polling interval, compute job_limit and AvgCompletionTime for every server;
5      end
6      for each d ∈ D do
7          Update the network values within each data host;
8          Sort R_d in the ascending order of f(Min, Cost(Link_dr), 1/BW(Link_dr));
9      end
10     for j ∈ J_U do
11         S_j, S'_j ← {};
12         for f_j ∈ F_j do
13             d ← f_min(Min, Cost(d_fj), Response(d_fj), D_fj);
14             r ← r_1d_fj ∈ R_d_fj ;
15             if S_j = {} then
16                 S_j ← S_j ∪ {r, d};
17                 S'_j ← S_j;
18             end
19             else
20                 S_j ← (S_j − {r_prev}) ∪ {r, d};
21                 S'_j ← S'_j ∪ {d};
22             end
23             S_j ← f_min(Min, e_j, t_j, {S_j, S'_j});
24             S'_j ← S_j;
25             r_prev ← r ∈ S_j;
26         end
27     end
28     Sort J_U in the ascending order of f_min(Min, e_j, t_j, J);
29     Expected_Budget = Budget_spent;
30     for j ∈ J_U do
31         Take the next job j ∈ J_U in sorted order;
32         r ← r ∈ S_j;
33         if Alloc_Job(r) < Job_Limit(r) then
34             if Min = Cost AND (T_Current + t_j) < T_Deadline then
35                 if (Expected_Budget + e_j) ≤ Budget then submit j to r;
36                 else stop dispatching and exit to main loop
37             end
38             if Min = Time AND Expected_Budget + e_j ≤ Budget then
39                 if (T_Current + t_j) < T_Deadline then submit j to r;
40                 else stop dispatching and exit to main loop
41             end
42             Expected_Budget = Expected_Budget + e_j;
43             Remove j from J_U;
44             Increment Alloc_Job(r);
45         end
46     end
47     Wait for the duration of the polling interval;
48 end
```

Fig. 2: Pseudo-code for Economy-based Scheduling of Data Intensive Applications

there can be maximum of P data hosts for any file. Therefore, for N jobs, the worst-case complexity of the mapping loop is $O(NKP)$.

IV. Experiments and Results

We have implemented the scheduling algorithm presented in Section III within the Gridbus Broker [14]. We have conducted empirical evaluation of the algorithm using an experimental setup modified from the one used for evaluation in [14] . The testbed resources used in our experiments is detailed in Table I. The broker itself was extended to consider the price of transferring data over network links between the compute resources and the data hosts while scheduling jobs. In our experiments, although we have artificially assigned data transmission costs shown in Table III, they can be linked to

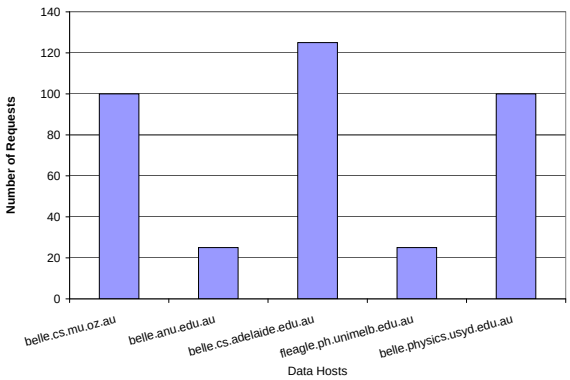
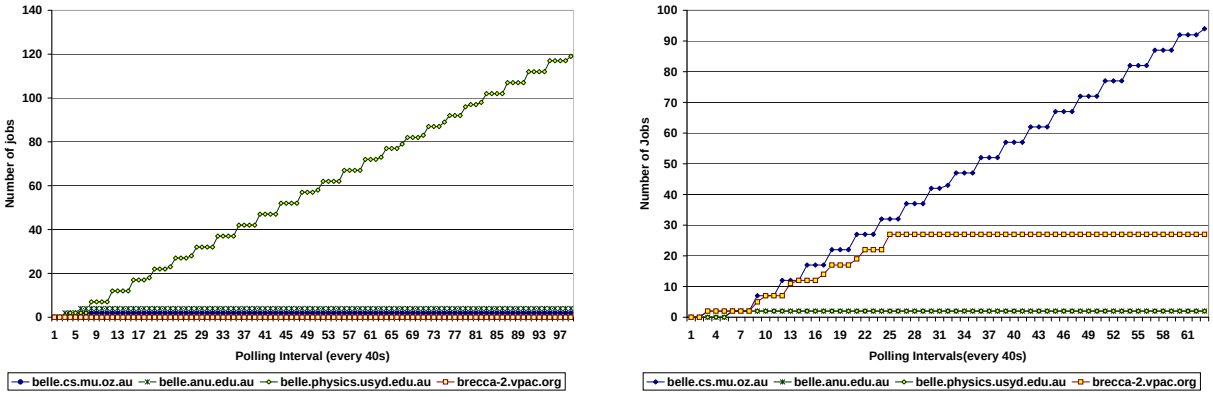


Fig. 3. Access distribution of jobs against data hosts

TABLE I

RESOURCES WITHIN BELLE TESTBED USED FOR EVALUATION AND THEIR COSTING

Organization	Resource details	Role	Cost (G$/CPUsec)	Total Jobs Executed	
				Time	Cost
Dept. of Computer Science, University of Melbourne	*belle.cs.mu.oz.au* 4 Intel 2.6 GHz CPU, 2 GB RAM, 70 GB HD, Linux	Broker Host, Data Host, Compute resource, NWS Server	6	94	2
School of Physics, University of Melbourne	*fleagle.ph.unimelb.edu.au* 1 Intel 2.6 Ghz CPU, 512 MB RAM, 70 GB HD, Linux	Replica Catalog host, Data host, NWS sensor	N.A. (Not used as a compute resource)	–	–
Dept. of Computer Science, University of Adelaide	*belle.cs.adelaide.edu.au* 4 Intel 2.6 GHz CPU, 2 GB RAM, 70 GB HD, Linux	Data host, NWS sensor	N.A. (Not used as a compute resource)	–	–
Australian National University, Canberra	*belle.anu.edu.au* 4 Intel 2.6 GHz CPU, 2 GB RAM, 70 GB HD, Linux	Data Host, Compute resource, NWS sensor	6	2	4
Dept of Physics, University of Sydney	*belle.physics.usyd.edu.au* 4 Intel 2.6 GHz CPU(1 avail), 2 GB RAM, 70 GB HD, Linux	Data Host, Compute resource, NWS sensor	2	119	2
Victorian Partnership for Advanced Computing, Melbourne	*brecca-2.vpac.org* 180 node cluster (only head node utilised)	Compute resource, NWS sensor	4	0	27



(a) cost minimization scheduling

(b) time minimization scheduling

Fig. 4. Cumulative number of jobs completed vs time for economy scheduling in data grids.

TABLE II

AVERAGE AVAILABLE BANDWIDTH BETWEEN DATA HOSTS AND COMPUTE RESOURCES AS REPORTED BY NWS(IN MBPS)

Compute / Data	UniMelb CS	ANU	UniSyd Physics	VPAC
ANU	6.99	10000	10.242	6.33
Adelaide CS	3.45	1.68	2.29	6.05
UniMelb Physics	41.05	6.53	2.65	20.57
UniMelb CS	10000	6.96	4.77	36.03
UniSyd Physics	4.78	12.57	10000	2.98

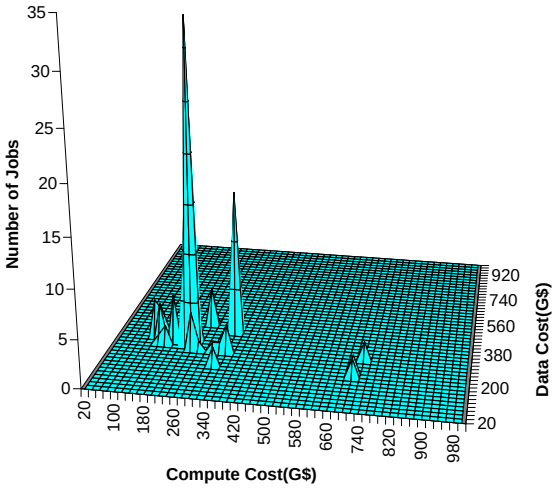
TABLE III

NETWORK COSTS BETWEEN DATA HOSTS AND COMPUTE RESOURCES (IN G$/MB)

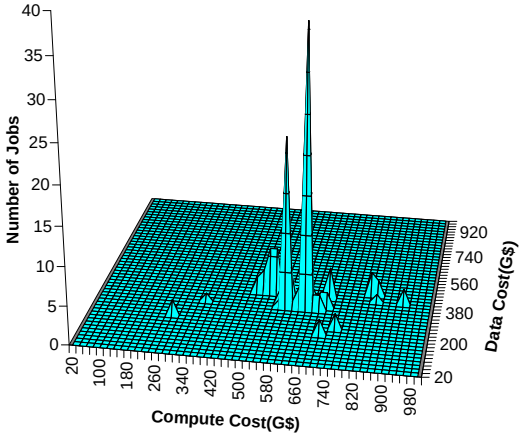
Compute / Data	UniMelb CS	ANU	UniSyd Physics	VPAC
ANU	34.0	0	31.0	38.0
Adelaide CS	36.0	34.0	31.0	33.0
UniMelb Physics	40.0	32.0	39.0	35.0
UniMelb CS	0	30.0	36.0	33.0
UniSyd Physics	33.0	35.0	0	37.0

TABLE IV
SUMMARY OF EVALUATION RESULTS

Scheduling strategy	Total Time Taken (mins.)	Compute Cost (G$)	Data Cost (G$)	Total Cost (G$)
Cost minimization	80	31198.27	39126.65	70324.93
Time Minimization	54	76054.90	43821.64	119876.55



(a) cost minimization scheduling



(b) time minimization scheduling

Fig. 5. Distribution of jobs against compute and data costs

real costs as prescribed by ISPs (Internet Service Providers). We have used NWS (Network Weather Service) [24] for measuring the network bandwidths between the computational and the data sites. A number of the data hosts were also functioning as compute resources. The average available bandwidth between the compute resources and the data hosts is given in Table II. The network bandwidth between a data host and a compute resource located on the same resource was set to an arbitrarily high value (10000 Mbps) within the broker and the network transmission cost in this case was set to zero. There was no access cost specified for the data hosts within this evaluation. The response time for the data hosts was also not taken into account within this evaluation as the data were not replicated and there was no choice between data hosts for a single dataset.

There were 100 datasets or files, of size 30 MB each, divided equally between the five data hosts listed in Table I. These were not replicated and thus, each dataset was uniquely hosted. We have used a synthetic application called *datacalc* for the evaluation. This program requests the data from the remote data host specified by the resource broker, executes some calculations and produces a small output file (of the order of KB). Each job, consisting of a instance of the application, required 3 datasets out of the 100 and there were 125 jobs in total. These data sets were specified through Logical File Names (LFNs) within a replica catalog and were resolved to the actual physical locations by the broker at runtime. Fig. 3 gives the distribution of the number of requests for data made by the jobs versus the data hosts. The distribution is the same for both cost and time minimization. The datasets were transferred in sequence, that is, the transfer of one dataset was started after the previous had completed. The computation times for the jobs were randomly distributed within 60-120 seconds.

The experiments were carried out on 29th November 2004 between 6:00 p.m. and 10:00 p.m. AEDT. The deadline and budget values for both cost and time minimization were 2 hours and 500,000 G$ respectively. Table IV shows the summary of the results that were obtained. The average costs per job incurred during cost and time minimization are 562.6 G$ and 959 G$ with standard deviations of 113 and 115 respectively. Mean wall clock time taken per job(including computation and data transfer time) was 167 secs for cost minimization and 135 secs for time minimization with standard deviations 16.7 and 19 respectively.

As expected, cost minimization scheduling produces minimum computation and data transfer expenses whereas time minimization completes the experiments in the least time. The graphs in Figs. 4(a) and 4(b) show the number of jobs completed versus time for the two scheduling strategies for data grids. Since the computation time was dominant, within cost minimization, the jobs were executed on the least economically expensive compute resource. This can be seen in Fig. 4(a) where the compute resource with the least cost per CPU sec, the resource at University of Sydney, was chosen to execute 95% of the jobs. Since a very relaxed deadline

was given, no other compute resource was engaged by the scheduler as it was confident that the least expensive resource alone would be able to complete the jobs within the given time. Within time minimization (Fig. 4(b)), the jobs were dispatched to the compute resources which promised the least execution time even if they were expensive as long as the expected cost for the job was less than the budget per job. Initially, the scheduler utilised two of the faster resources, the University of Melbourne Computer Science(UniMelb CS) resource and the VPAC resource. However, as seen from Fig. 3, 26.67% of the requests for datasets were directed to the UniMelb CS resource. A further 6.67% were directed to the resource in UniMelb Physics. Hence, any jobs requiring one of the datasets located on either of the above resources were scheduled at the UniMelb CS resource because of the low data transfer time. Also, the UniMelb CS resource had more processors. Hence, a majority of the jobs were dispatched to it within time minimization.

Figs. 5(a) and 5(b) show the distribution of the jobs with respect to the compute and data costs. For cost minimization, 95% of the jobs have compute costs less than or equal to 400 G$ and data costs between 250 G$ to 350 G$. In contrast, within time minimization, 91% of the jobs are in the region of compute costs between 500 G$ to 700 G$ and data costs between 300 G$ to 400 G$. Thus, it can be inferred that the broker utilized the more expensive compute and network resources to transfer data and execute the jobs within time minimization as more jobs are in the region of high compute costs and medium data costs.

V. Conclusion and Future Work

We have presented here an economy-based model for executing jobs on data grids which takes in to account both processing and data transfer costs. We have discussed an algorithm which greedily creates a resource set, consisting of both compute and data resources, that promises the least cost or least time depending on the minimisation chosen. We have evaluated the algorithm on a Data Grid testbed and presented the empirical results. The results show that the algorithm utilizes the available resources to minimise its objective function by building resource sets incrementally. It does so without involving the complexity of checking every possible combination of resources.

We plan to conduct further evaluations to conclusively state that the algorithm minimizes its objective functions. We also plan to evaluate the algorithm with a testbed with replicated data along with specifying access costs for the data hosts.

References

[1] T. Hey and A. E. Trefethen, "The UK e-Science Core Programme and the Grid," *Journal of Future Generation Computer Systems(FGCS)*, vol. 18, no. 8, pp. 1017–1031, 2002.

[2] R. Moore, C. Baru, R. Marciano, A. Rajasekar, and M. Wan, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998, ch. 5, "Data Intensive Computing", pp. 105–131.

[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," *Journal of Network and Computer Applications*, vol. 23, no. 3, pp. 187–200, 2000.

[4] Particle Physics Data Grid (PPDG). [Online]. Available: http://www.ppdg.net/

[5] European Union Data Grid. [Online]. Available: http://www.eu-datagrid.org/

[6] The Belle Data Grid Project. [Online]. Available: http://epp.ph.unimelb.edu.au/epp/grid/presentation/project1.html

[7] MONARC Project, CalTech. Accessed Nov 2004. [Online]. Available: http://monarc.cacr.caltech.edu/

[8] P. Lebrun, "The Large Hadron Collider, A Megascience Project," in *38th INFN Eloisatron Project Workshop on Superconducting Materials for High Energy Colliders*, Erice, Italy, October 1999.

[9] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *Computer Communications Review*, vol. 3, July 2002.

[10] R. Buyya, J. Giddy, and D. Abramson, "A case for economy grid architecture for service-oriented grid computing," in *10th IEEE International Heterogeneous Computing Workshop (HCW 2001), In conjunction with IPDPS 2001*, San Francisco, California, USA, April 2001.

[11] R. Buyya, J. Giddy, and D. Abramson, "An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications," in *The Second Workshop on Active Middleware Services (AMS 2000)*, Pittsburgh, USA, 2000.

[12] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid environments," in *9th Heterogeneous Computing Systems Workshop (HCW 2000)*, Cancun,Mexico, 2000.

[13] K. Ranganathan and I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications," in *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC)*, Edinburgh, Scotland, July 2002.

[14] S. Venugopal, R. Buyya, and L. Winton, "A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids," in *Proceedings of the 2nd Workshop on Middleware in Grid Computing (MGC 04) : 5th ACM International Middleware Conference (Middleware 2004)*, Toronto, Canada, October 2004.

[15] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin, "An Economic Paradigm for Query Processing and Data Migration in Mariposa," in *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, Austin, TX, USA, Sept. 28-30 1994.

[16] H. Stockinger, K. Stockinger, E. Schikuta, and I. Willers, "Towards a cost model for distributed and replicated data stores," in *9th Euromicro Workshop on Parallel and Distributed Processing PDP 2001*. Mantova, Italy: IEEE Computer Society Press, February 2001.

[17] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, K. Stockinger, and F. Zini, " Evaluation of an Economy-Based File Replication Strategy for a Data Grid," in *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003 (CCGrid 2003)*, Tokyo, Japan, May 2003.

[18] R. Buyya, "Economic-based Distributed Resource Management and Scheduling for Grid Computing," Ph.D. dissertation, Monash University, Australia, 2002.

[19] S. Vazhkudai, S. Tuecke, and I. Foster, "Replica selection in the globus data grid," in *Proceedings of the First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001)*, Brisbane, Australia, May 2001.

[20] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The sdsc storage resource broker," in *Procs. of CASCON'98*, Toronto, Canada, Nov 1998.

[21] D. Abramson, J. Giddy, and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?" in *IPDPS'2000*, Cancun, Mexico, 2000.

[22] T. Hui and C. Tham, "Reinforcement learning-based dynamic bandwidth provisioning for quality of service in differentiated services networks," in *Proceedings of IEEE International Conference on Networks (ICON 2003)*, Sydney, Australia, Sept.-Oct. 2003.

[23] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing(JPDC)*, vol. 59, pp. 107–131, Nov 1999.

[24] R. Wolski, N. Spring, and J. Hayes, "The network weather service: A distributed resource performance forecasting service for metacomputing," *Journal of Future Generation Computing Systems*, vol. 15, pp. 757–768, 1999.