A Deadline and Budget Constrained Scheduling Algorithm for eScience Applications on Data 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
{srikumar, raj}@cs.mu.oz.au

Abstract. In this paper, we present an algorithm for scheduling of distributed data intensive Bag-of-Task applications on Data Grids that have costs associated with requesting, transferring and processing datasets. The algorithm takes into account the explosion of choices that result due to a job requiring multiple datasets from multiple data sources. The algorithm builds a resource set for a job that minimizes the cost or time depending on the user's preferences and deadline and budget constraints. We evaluate the algorithm on a Data Grid testbed and present the results.

1 Introduction

Multi-institutional scientific projects in domains such as high energy physics, astronomy and bioinformatics are increasingly generating data in the range of Tera Bytes (TB) which is replicated at various sites for improving reliability and locality. Grid computing [1] has made it possible to aggregate heterogeneous, geographically distributed compute and storage resources for executing large-scale applications in such eScience [2] projects. Data Grids [3] are instances of Grids where access and management of distributed data resources have equal or higher priority than computational requirements. A well-cited example of a Data Grid is the one being set up for processing the output of the ATLAS experiment at the Large Hadron Collider(LHC) at CERN [4].

The execution of data-intensive applications involves requirements for discovering, processing, storing and managing large distributed datasets and is guided by factors such as cost and speed of accessing, transferring and processing data. There may be multiple datasets involved in a computation, each replicated at multiple locations that are connected to each other and to the compute resources by networks with varying cost and capability. Consequently, this explosion of choices makes it difficult to identify the most optimal resources for retrieving and performing the required computation on the selected datasets.

In large collaborations that form Data Grids, there can be a lot of pressure on the network, storage and processing elements. This can lead to overloading of resources and appearance of network "hot spots" as is commonly observed in the World Wide Web [5]. Previous work has suggested a computational economy metaphor for resource management within computational grids [6]. Resource providers price their goods to reflect supply and demand in order to make a profit or to regulate consumption. On the consumer

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 60–72, 2005.

side, users specify their deadlines for completing their jobs, the budget available to them and their preference for the cheapest or the fastest processing according to their needs and priorities. While such strategies have been proposed and evaluated for computational grids [7], no study has yet been made for similar requirements on Data Grids.

In this paper, we introduce an algorithm for scheduling a Bag-of-Tasks(BoT) application on a set of geographically distributed, heterogeneous compute and data resources. Each of the tasks within the application depends on multiple datasets that may be distributed anywhere within the grid. Also, there are economic costs associated with the movement and processing of datasets on the distributed resources. The algorithm minimizes either the overall cost or the time of execution depending on the user's preference subject to two user-defined constraints - the deadline by which the processing must be completed and the overall budget for performing the computation.

The rest of this paper is organised as follows. In Section 2, we survey previous work in data-intensive Grid scheduling. We extend the notion of user-driven deadline and budget constrained scheduling within computational grids to data grids in Section 3. In Section 4, the proposed algorithm is evaluated on a real Grid testbed and the results are reported. Finally, we conclude our paper and outline future work.

2 Related Work

Several approaches have been proposed to schedule data-intensive applications on distributed resources. In [8], 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. Takefusa, et. al [9] explore various combinations of scheduling and replication algorithms and come to the conclusion that for large files, moving computation close to the source of data is the best strategy. Ranganathan and Foster [10] have simulated job scheduling and data scheduling algorithms and recommend that it is best to decouple data replication from the job scheduling. In Chameleon [11], the scheduling strategy executes a job on one site while taking into account computation and communication costs. Kim and Weissman [12] explore a Genetic Algorithm-based approach for decomposing and scheduling a parallel data-intensive application. In previous work [13], 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 the case of applications having to deal with data from multiple sources, the problem is executing the application such that it is optimal with respect to all the data sources rather than a single source as has been considered in the works presented before.

The problem of scheduling BoT applications on distributed systems is a very well-studied one [14][8][15]. Deadline and budget constrained scheduling algorithms for compute-intensive BoT applications were proposed and evaluated in [7]. In this paper, we extend the same notion to data-intensive BoT applications in the following manner. This paper proposes a detailed cost model for distributed data-intensive applications that builds on the models for system costs (processing and transferring overheads) pre-

viously discussed in [12][11][16] and takes into account expenses for storage, transfer and processing of data. It then proposes a new algorithm based on the Min-Min heuristic described in [14] that takes into account the deadline and budget constraints of the users and produces a schedule that minimises either cost or time depending on their requirements. The proposed algorithm also explicilty deals with the explosion of choices in scheduling Data Grid applications as is mentioned in the previous section. While this is similar in intent to the work presented in [17], there is a lot of difference in the methodologies. In [17], the search space is pruned by grouping resources into collections and then sorting the collections in the order decided by a certain performance metric. As will be shown later, within our algorithm, the resource sets are created in an incremental fashion and the search space is limited to only those resources that minimze the objective function.

3 Scheduling

Fig. 1 shows a typical Data Grid environment which is composed of storage resources, or *data hosts*, which store the data and compute resources which run the jobs that execute upon the data. This is based on the scenarios drawn up for users of the production Data Grid projects such as LHC Grid [18]. It is possible that the same resource 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 depending on the policies set by the administrators of the storage resources and/or the producers of data. The scheduler is able to query a data directory such as a Replica Catalog [19] or the SRB [20] Metadata Catalog for 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 data host and the cost of transferring datasets from the data host to the compute resource through the network.

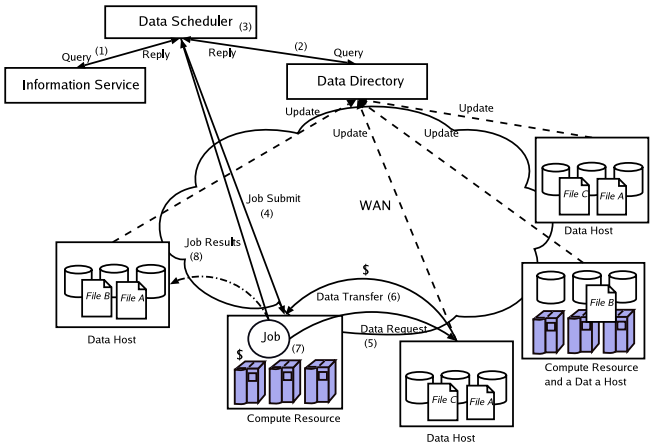


Fig. 1. An economy-based data grid environment

We consider a job (equivalent to a task in a BoT) as the atomic unit of computation within this model. Each job requires one or more datasets as input. Each dataset is available through one or more data hosts. 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 the 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 within the set. Here, only resources that meet minimum requirements of the application such as architecture(instruction set), operating system, minimum free memory and storage threshold are considered as suitable candidates for job execution.

We consider, therefore, a set of N independent jobs $J = \{j_1, j_2, \ldots, j_N\}$ which have to be scheduled on M computational resources $R = \{r_1, r_2, \ldots, r_M\}$. Typically, $N \gg M$. Each job $j, j \in J$ requires a subset $F_j = \{f_{j1}, f_{j2}, \ldots, f_{jK}\}$ of a set of datasets, F, which are each replicated on a subset of P data hosts,$D = \{d_1, d_2, \ldots, d_P\}$. For $f \in F$, $D_f \subseteq D$ is the set of data hosts on which f is replicated.

The time taken to execute a job is the sum of the execution time and the times taken to transfer each of the datasets required for the job from their respective data hosts to the compute node. If the execution time for job j on compute resource r is denoted by t_{jr} and the transfer time for a dataset $f_j \in F_j$ from a location $d_{f_j} \in D_{f_j}$ to compute resource r is denoted by $t_{f_j r}$, then the total time required for executing the job j is given by $t_j = t_{jr} + \sum_{f_j \in F_j} t_{f_j r}$ where $t_{f_j r}$ is the sum of the *response time* of d_{f_j} and the time taken for the actual data movement. We define *response time* as the difference between the time when the request was made to the data host and the time when the first byte of the file was received at the compute resource. It is an increasing function of the load on the data host. The time taken for the data movement is the size of the data divided by the available bandwidth between the data host and the compute resource. While we have considered a case of sequential data transfer in this model, it can be modified to consider a parallel data transfer model as presented in [12].

To calculate the economic cost of executing the job, we denote the economic cost of executing the job j on the compute resource r by e_{jr} and cost of transferring the dataset f by $e_{f_j r}$. Therefore, the total execution cost for job j is given by $e_j = e_{jr} + \sum_{f_j \in F_j} e_{f_j r}$ where $e_{f_j r}$ is the sum of *access cost*, which is the price levied by the data host for serving the requested dataset and network transfer cost dependent on the size of the file and the cost of transferring unit data from data host to compute resource . The access cost can be an increasing function of 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. The cost of the network 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 [21], a channel with a higher QoS may be more expensive but the data may be transferred faster.

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 $\sum_J e_j \leq Budget$.

3.1 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.

In both cases the same algorithm can be applied to solve the different objective functions. This is done by means of a switch 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 set of values, 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. The functions $min(CVar, A)$ 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 the objective function to be minimized by the algorithm. Consequently, Min is a parameter to the scheduling algorithm.

The listing for the algorithm is given in Figure 2. J_U, J_A, J_C and J_F are subsets of the set of jobs J consisting of jobs in $Unsubmitted$, $Active$, $Completed$ and $Failed$ states respectively. Jobs initially are in the $Unsubmitted$ state, once they are submitted, they become $Active$ and finally end up being $Completed$ or $Failed$. The scheduling algorithm exits if all jobs are in the final states or if the deadline or budget constraints are violated. The initial part of the loop does bookkeeping. At every polling interval, we update the performance data of the compute resources and calculate the allocation for the current polling interval. 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 rest of the algorithm is in two parts : the first part maps the jobs to the selected compute resources depending on selected minimization objective (cost or time) while the second dispatches the jobs while enforcing the deadline and budget constraints. These are described as follows:

```
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          Calculate performance data on the basis of resource performance in previous
           polling interval;
5      end
6      for each d ∈ D do
7          Based on current network values, sort R in the increasing order of
           Cost(Link_dr) or 1/BW(Link_dr)) depending on whether Min is Cost or
           Time;
8          Maintain this list as R_d;
9      end
10     MAPPING SECTION;
11     for j ∈ J_U do
12         S_j, Temp_j ← {};
13         for f_j ∈ F_j do
14             Select {r, d_fj} such that, depending on Min, either e_fjr or t_fjr is
               minimised;
15             if S_j = {} then
16                 S_j ← S_j ∪ {r, d_fj};
17                 Temp_j ← S_j;
18             end
19             else
20                 S_j ← (S_j − {r_prev}) ∪ {r, d_fj};
21                 Temp_j ← Temp_j ∪ {d_fj};
22             end
23             S_j ← f_min(Min, e_j, t_j, {S_j, Temp_j});
24             Temp_j ← S_j;
25             r_prev ← r ∈ S_j;
26         end
27     end
28     DISPATCHING SECTION;
29     Sort J_U in the ascending order of e_j or t_j depending on Min;
30     Expected_Budget ← Budget_spent;
31     for j ∈ J_U do
32         Take the next job j ∈ J_U in sorted order;
33         r ← r ∈ S_j;
34         if r can be allocated more jobs then
35             if Min = Cost AND (T_Current + t_j) < T_Deadline then
36                 if (Expected_Budget + e_j) ≤ Budget then submit j to r;
37                 else stop dispatching and exit to main loop
38             end
39             if Min = Time AND Expected_Budget + e_j ≤ Budget then
40                 if (T_Current + t_j) < T_Deadline then submit j to r;
41                 else stop dispatching and exit to main loop
42             end
43             Expected_Budget = Expected_Budget + e_j;
44             Remove j from J_U;
45         end
46     end
47     Wait for the duration of the polling interval;
48 end
```

Fig. 2. Pseudo-code for Deadline and Budget Constrained Cost-based Scheduling of Data Intensive Applications.

Mapping: We require one compute resource to execute the job 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. However, if we try all possible combinations of compute and data resources for each job, this results in a $O(N(MP)^K)$ mapping where K is the maximum number of datasets for each job.

We, therefore, decrease the complexity by making a choice at each step within the mapping section. For a job, we iterate through the list of datasets it requires. For each dataset, we pick the combination of a compute resource and a data host that returns the lowest value for expected transfer cost($e_{f_j r}$) or time($t_{f_j r}$) depending on either cost or time minimization. This is done in $O(P)$ time as for each data host, we only have to pick the first compute resource out of its sorted list of compute resources. Then we create two resource sets, S_j and $Temp_j$, 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, r_{prev} (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 the compute resource and the resource set so formed at the end of each iteration is better than those selected in all previous iterations. For N jobs, therefore, the above mapping loop runs in $O(NKP)$ time.

Dispatching: 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 if the allocation for the resources as determined in the initial part has not been exhausted. 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.

4 Experiments and Results

We have implemented the scheduling algorithm presented in Section 3 within the Gridbus Broker [13]. The testbed resources used in our experiments is detailed in Table 1. The *cost per sec* denotes the rate for performing a computation on the resource in Grid Dollars (G$). It can be seen that some of the resources were also used to store the replicated data and therefore, were performing the roles of both data hosts and compute resources. The average available bandwidth between the compute resources and the data hosts is given in Table 2. We have used NWS (Network Weather Service) [22]

Table 1. Resources within Belle testbed used for evaluation and their costing

Organization	Resource details	Role	Compute Cost(G$ /sec)	Total Jobs Done	
				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.*	–	–
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. *	–	–
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	2	119
Victorian Partnership for Advanced Computing, Melbourne	*brecca-2.vpac.org* 180 node cluster (only head node utilised)	Compute resource, NWS sensor	4	27	0

*Not used as a compute resource but only as a data host

for measuring the network bandwidths between the computational and the data sites. We have used only the performance data and not the bandwidth forecasts provided by NWS. It has been shown that NWS measurements with 64 KB probes cannot be correlated with large data transfers[23][24]. However, we consider the NWS measurements are indicative of the actual available bandwidth in our case. In the future, we hope to use regression models for more accurate measurements as has been shown in [23][24] .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 3, they can be linked to real costs as prescribed by ISPs (Internet Service Providers). During scheduling, data movement cost and time were explicitly taken into account when data and compute services were hosted on different resources.

Within the performance evaluation, we wanted to capture various properties and scenarios of Data Grids and applications. Accordingly we devised a synthetic application application that requests K datasets that are located on distributed data sources and are registered within a replica catalog. The datasets are specified as Logical File Names (LFNs) and resolved to the actual physical locations by the broker at runtime. The application then processes these datasets and produces a small output file (of the order of KB). In this particular evaluation, the datasets are files registered within the catalog. There are 100 files of size 30 MB each, distributed between the data hosts listed in Table 1. The BoT application here is a parameter-sweep application consisting of 125

Table 2. Avg. Available Bandwidth between Data Hosts and Compute Resources as reported by NWS(in Mbps)

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

Table 3. Network Costs between Data Hosts and Compute Resources (in G$/MB)

Data Hosts	Compute Resources			
	UniMelb CS	ANU	UniSyd	VPAC
ANU	34.0	0	31.0	38.0
Adelaide	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	33.0	35.0	0	37.0

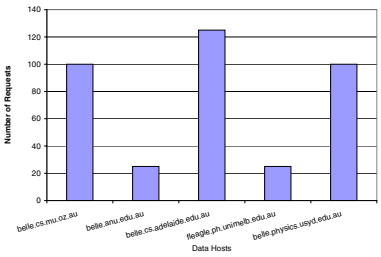


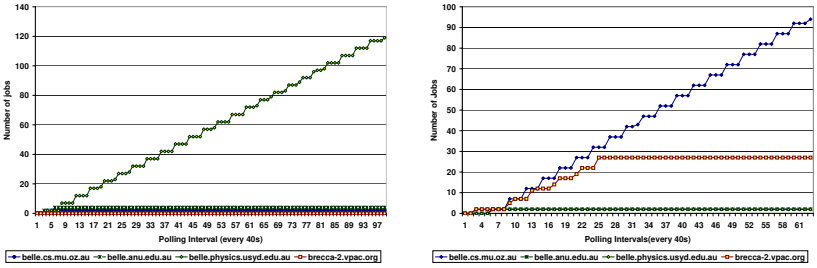
Fig. 3. Distribution of file access

Table 4. Summary of Evaluation Results

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

jobs, each job an instance of the application described before requiring 3 files (that is, $K = 3$ for all the jobs in this evaluation). 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.

There are two measures of performance that we are interested in: the first is the relative usage of the computational resources under cost and time minimization which indicates how the choice of minimization criteria impacts resource selection and the second, is the distribution of jobs with respect to the computational and data transfer costs and times incurred within each minimization which tells us the how effective the algorithm was in producing the cheapest or the fastest schedule. 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 4 shows the summary of the results that were obtained. The total time is the wall clock time taken from the start of the scheduling procedure up to the last job completed. All the jobs completed successfully in both the experiments. 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.



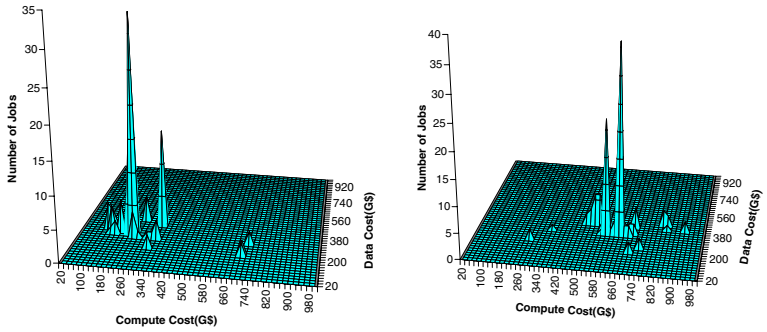
(a) cost minimization scheduling (b) time minimization scheduling

Fig. 4. Cumulative number of jobs completed vs time

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 and 4 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 where the compute resource with the least cost per 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, 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.

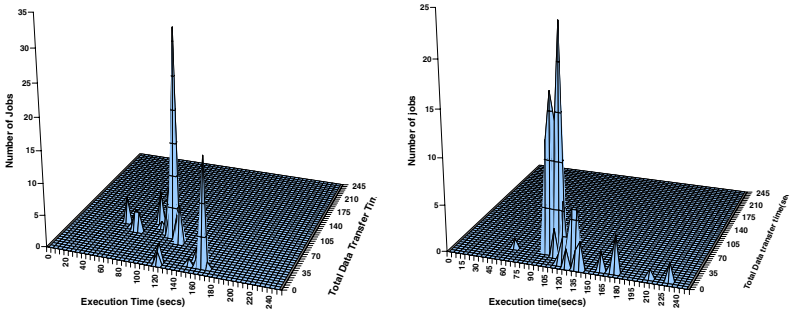
Figs. 5(a) and 5(b) show the distribution of the jobs with respect to the compute and data costs respectively. For cost minimization, 95% of the jobs have compute costs less than or equal to 400 G$ and data costs between 250 G$ to 350 G$. In contrast, within time minimization, 91% of the jobs are in the region of compute costs between 500 G$ to 700 G$ and data costs between 300 G$ to 400 G$. Hence, in time minimization, more jobs are in the region of high compute costs and medium data costs. 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.

Figs. 6(a) and 6(b) show the distribution of the jobs with respect to the total execution time and the total data transfer time for cost minimization and time minimization respectively. The execution time excludes the time taken for data transfer. It can be seen that within time minimization 6(b) the maximum data transfer time was 35s as com-



(a) cost minimization scheduling (b) time minimization scheduling

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



(a) cost minimization scheduling (b) time minimization scheduling

Fig. 6. Distribution of jobs against execution time and data transfer time

pared to 75s for cost minimization. Also, there are more jobs within time minimization that have had transfer time less than 10s which implies that the jobs were scheduled close to the source of the data. Therefore, from the results, it can be seen that given cost or time minimization, the algorithm presented in this work does minimize the objective function for upto 90% of the set of jobs.

5 Conclusion and Future Work

We have presented here a model for executing jobs on data grids which takes in to account both processing and data transfer costs. We have also presented 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 minimization chosen. We have presented empirical results obtained from evaluating the algorithm on a Data Grid testbed.

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 different levels of replication of data and with varying resource prices.

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers (1999)
2. Hey, T., Trefethen, A.E.: The UK e-Science Core Programme and the Grid. Journal of Future Generation Computer Systems(FGCS) **18** (2002) 1017–1031
3. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. Journal of Network and Computer Applications **23** (2000) 187–200
4. Lebrun, P.: The Large Hadron Collider, A Megascience Project. In: 38th INFN Eloisatron Project Workshop on Superconducting Materials for High Energy Colliders, Erice, Italy (1999)
5. Mahajan, R., Bellovin, S.M., Floyd, S., Ioannidis, J., Paxson, V., Shenker, S.: Controlling high bandwidth aggregates in the network. Computer Communications Review **3** (2002)
6. Buyya, R., Giddy, J., Abramson, D.: 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)
7. Buyya, R., Giddy, J., Abramson, D.: 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)
8. Casanova, H., Legrand, A., Zagorodnov, D., Berman, F.: Heuristics for Scheduling Parameter Sweep Applications in Grid environments. In: 9th Heterogeneous Computing Systems Workshop (HCW 2000), Cancun,Mexico, IEEE CS Press (2000)
9. Takefusa, A., Tatebe, O., Matsuoka, S., Morita, Y.: Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications. In: Proceedings of the 12th IEEE international Symposium on High Performance Distributed Computing(HPDC-12), Seattle, USA, IEEE CS Press (2003)
10. Ranganathan, K., Foster, I.: 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, IEEE Computer Society (2002)
11. Park, S.M., Kim, J.H.: Chameleon: A Resource Scheduler in a Data Grid Environment. In: Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003 (CCGrid 2003), Tokyo, Japan, IEEE CS Press (2003)
12. Kim, S., Weissman, J.: A GA-based Approach for Scheduling Decomposable Data Grid Applications. In: Proceedings of the 2004 International Conference on Parallel Processing (ICPP 04), Montreal, Canada, IEEE CS Press (2003)
13. Venugopal, S., Buyya, R., Winton, L.: 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 (2004)
14. Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. Journal of Parallel and Distributed Computing(JPDC) **59** (1999) 107–131

15. Beaumont, O., Legrand, A., Robert, Y., Carter, L., Ferrante, J.: Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms. In: Proceedings of the 2002 International Parallel and Distributed Processing Symposium(IPDPS '02), Fort Lauderdale, California, USA, IEEE CS Press (2002)
16. Stockinger, H., Stockinger, K., Schikuta, E., Willers, I.: 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 (2001)
17. Dail, H., Casanova, H., Berman, F.: A Decoupled Scheduling Approach for the GrADS Environment. In: Proceedings of the 2002 IEEE/ACM Conference on Supercomputing (SC'02), Baltimore, USA, IEEE CS Press (2002)
18. Hoschek, W., Jaen-Martinez, F.J., Samar, A., Stockinger, H., Stockinger, K.: Data management in an international data grid project. In: Proceedings of the First IEEE/ACM International Workshop on Grid Computing(GRID '00), Bangalore, India, Springer-Verlag, Berlin (2000)
19. Vazhkudai, S., Tuecke, S., Foster, I.: 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 (2001)
20. Baru, C., Moore, R., Rajasekar, A., Wan, M.: The SDSC Storage Resource Broker. In: Procs. of CASCON'98, Toronto, Canada (1998)
21. Hui, T., Tham, C.: 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 (2003)
22. 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
23. Vazhkudai, S., Schopf, J.: Using Regression Techniques to Predict Large Data Transfers. International Journal of High Performance Computing Applications **17** (2003) 249–268
24. Faerman, M., Su, A., Wolski, R., Berman, F.: Adaptive Performance Prediction for Distributed Data-Intensive Applications. In: Proceedings of the 1999 IEEE/ACM Conference on Supercomputing (SC'99), Portland, Oregon, USA, IEEE CS Press (1999)