A Set Coverage-based Mapping Heuristic for Scheduling Distributed 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}@csse.unimelb.edu.au

Abstract

Data-intensive Grid applications need access to large datasets that may each be replicated on different resources. Minimizing the overhead of transferring these datasets to the resources where the applications are executed requires that appropriate computational and data resources be selected. In this paper, we introduce a heuristic for the selection of resources based on a solution to the Set Covering Problem (SCP). We then pair this mapping heuristic with the well-known MinMin scheduling algorithm and conduct performance evaluation through extensive simulations.

1 Introduction

Grids [8] aggregate computational, storage and network resources to provide pervasive access to their combined capabilities. Additionally, Data Grids [6, 12] provide services such as low latency transport protocols and data replication mechanisms to distributed data-intensive applications that need to access, process and transfer large datasets stored in distributed repositories. Such applications are commonly used by communities of researchers in domains such as high-energy physics, astronomy and biology.

Distributed data-intensive applications commonly consist of tasks that process datasets that are located on various storage repositories or *data hosts*. Each of these datasets may be replicated at several locations that are connected to each other and to the computational sites (or *compute resources*) through networks of varying capability. Also, the datasets are generally large enough (of the order of GigaBytes (GB) and higher) that transferring them from storage resources to the eventual point of execution produces a noticeable impact on the execution time of the application. Therefore, *mapping* tasks to the appropriate compute resources for execution and to the corresponding data resources for accessing the required datasets such that the overall execution time is minimized is a challenging problem.

The work in this paper is mainly concerned with scheduling applications that consist of a collection of tasks without interdependencies, each of which requires multiple datasets, onto a set of Grid resources. (Each task is translated into a job that is scheduled on to a computational resource and requests datasets from the storage resources so identified. Therefore, for the rest of the paper, we will refer to such tasks as jobs as well.) The scheduling strategy has to map a task on to a resource set consisting of one compute resource to execute the task and one data host each for transferring each dataset required for the computation. In this paper, we present such a mapping heuristic based on a solution to the SCP and evaluate it against other heuristics through simulation.

The rest of the paper is structured as follows: the next section presents the resource model and the application model that we target in the research presented in this paper. The mapping heuristic is presented in the following section and is succeded by details of experimental evaluation and consequent results. Finally, we present related work and conclude the paper.

2 Model

2.1 Resource Model

We model the target data-intensive computing environment based on existing production testbeds such as the European DataGrid testbed [12] or the United States Grid3 testbed [9]. As an example, Figure 1 shows a subset of European DataGrid Testbed 1 derived from Bell, et. al [2]. The resources in the figure are spread across 7 countries and belong to different autonomous administrative domains.

In such Grid networks, we consider a data-intensive

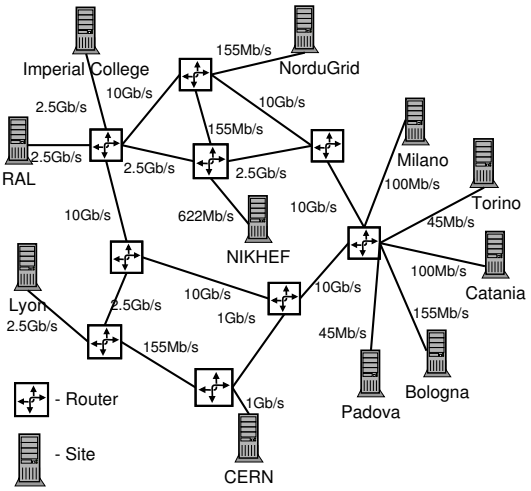


Figure 1. European Data Grid Testbed 1 [2].

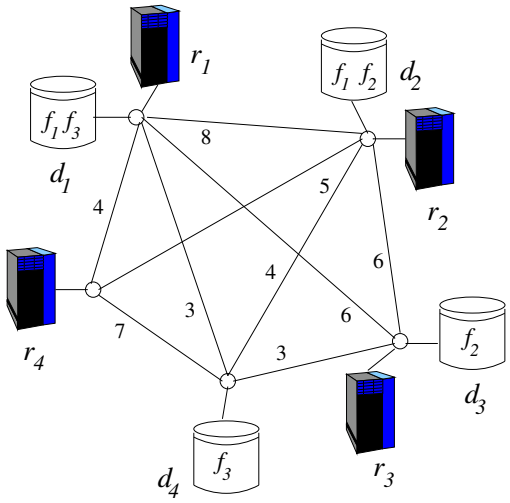


Figure 2. A data-intensive environment.

computing environment to consist of M compute resources $R = \{r_m\}_{m=1}^{M}$ and P data hosts,$D = \{d_p\}_{p=1}^{P}$, collectively referred to as *resources*. A compute resource is a high performance computing platform such as a cluster consisting of processing nodes that are connected in a private local area network and are managed by a batch job submission system hosted at the "head" or "front-end" node that is connected to the public Internet. A data host can be a storage resource such as a Mass Storage Facility connected to the Internet or may be simply a storage device attached to a compute resource in which case, it inherits the network properties of the latter. It is important to note that even in the second case, the data host is considered as a separate entity from the compute resource.

Data is organised in the form of datasets that are replicated on the data hosts by a separate replication process that follows a strategy (e.g. [2]) that takes into consideration various factors such as locality of access, load on the data host and available storage space. Information about the datasets and their location is available through a catalog such as the Storage Resource Broker Metadata Catalog [19].

Figure 2 shows a simplified data-intensive computing environment consisting of four compute resources and an equal number of data hosts connected by links of different bandwidths. We consider the logical network topology wherein each resource is connected to every other resource by a distinct network link. The time taken by a compute resource to access a dataset located on the storage resource at the same site is limited only by the intra-site bandwidth if the storage is a separate physical machine or by the bandwidth between the hard disk and other peripherals if the storage is on the compute machine itself. In both cases, it is considered to be an order of magnitude lower than the time taken to access a dataset through the Internet from other sites as there is contention for bandwidth among the various sites. Therefore, for the purpose of this study, only the bandwidth between different physical sites is taken into account .

2.2 Application Model

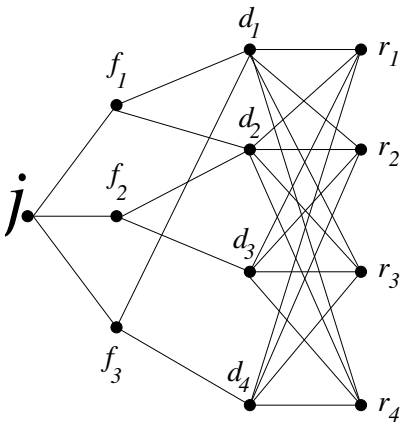


Figure 3. Job Model.

The application is composed of a set of N jobs without interdependencies, $J = \{j_i\}_{i=1}^{N}$. Typically, $N \gg M$. Each job $j, j \in J$ requires a subset $F^j = \{f_k^j\}_{k=1}^{K}$ of a set of datasets, F, which are each replicated on a subset of P data hosts, $D = \{d_p\}_{p=1}^{P}$. For a dataset $f \in F$, $D_f \subseteq D$ is the set of data hosts on which f is replicated. Each job requires one processor in a compute resource for executing the job and one data host each for accessing each of the K datasets required by the job. The compute resource and the data hosts thus selected are collectively referred to as the *resource set* associated with the job and is denoted by

$S^j = \{\{r\}, \{d_k\}_{k=1}^K\}$ where $r \in R$ is the compute resource where the job is to be executed and d_k is the data host selected for accessing $f_k^j \in F^j$. Figure 3 shows an example of such a job $j \in J$ that requires resources shown in Figure 2.

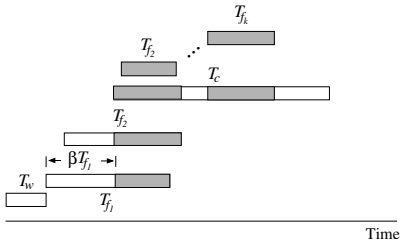


Figure 4. Job Execution Stages and Times.

Figure 4 shows an example of a data-intensive job with the times involved in various stages shown along a horizontal time-axis. T_w is the time spent in waiting in the queue on the compute resource and T_c is the time spent by the job in purely computational operations (also called computation time). T_w and T_c are functions of the load and processing speed of the compute resource. T_{f_i} is the time required to transfer the file f_i from its data host to the compute resource and is dependent on the available bandwidth between the two. The completion time for the job, T_j, is the wallclock time taken for the job to finish execution and is a function of these three times. For large datasets, the data transfer time impacts the completion time significantly. While the transfer time is determined by the manner in which the dataset is processed by the job, it is also influenced by the selection of data hosts. For example, many applications request and receive required datasets in parallel before starting computation. In this case, $T_j = T_w + \max_{1 \leq i \leq K}(T_{f_i}) + T_c$ However, the number of simultaneous transfers determines the bandwidth available at the receiver end for each transfer and therefore, the T_{f_i}. Transfer times can be minimized by locating a compute resource associated with a data host that has the maximum number of datasets required by the job so that the bulk of the data access is local. This would also benefit the case where the job accesses datasets sequentially.

We wish to minimize the total *makespan* [14] of the application consisting of N such data-intensive jobs. To that end, we follow the well-known MinMin heuristic, proposed in [14], to schedule the entire set of jobs. The MinMin heuristic submits the task with the minimum MCT to the compute resource that guarantees it. Therefore, our aim here is to select a resource set that produces the Minimum Completion Time (MCT) for a job. We adopt the strategy of finding the resource set with the least number of datahosts required to access the datasets required for a job and then, finding a suitable compute resource to execute it. We experimentally show that this approach produces schedules that are competitive with the best and is reasonably fast as well.

3 Scheduling

Figure 5 lists a generic scheduling algorithm for scheduling a set of jobs on a set of distributed compute resources. Each of the steps can be implemented independently of each other and therefore, many strategies are possible. In this paper, we concentrate on the process within the *for* loop, i.e., finding the appropriate resource set for a job.

The scheduler forms a part of a larger application execution framework such as a resource broker (e.g.[24]). The resource broker is able to identify resources that meet minimum requirements of the application such as architecture (instruction set), operating system, storage threshold and data access permissions and these are provided as suitable candidates for job execution to the scheduler.

```
while there exists unsubmitted jobs do
    Update the resource performance data based on job
    scheduled in previous intervals
    Update network data between resources based on current
    conditions
    foreach unsubmitted job do
        Find the MCT and the resource set that guarantees
        that time
    end
    repeat
        Heuristically assign mapped jobs to each compute
        resource
    until all jobs are submitted or no more jobs can be
    submitted
    Wait until the next scheduling event.
end
```

Figure 5: A Generic Scheduling Algorithm.

3.1 Mapping Heuristic

For a job $j \in J$, consider a graph $G^j = (V, E)$ where $V = (\bigcup_{f \in F^j}\{D_f\}) \cup F^j$ and E is the set of all directed edges $\{d, f\}$ such that $d \in D_f$. As an example, for the job in Figure 3, we can derive the graph shown in Figure 6(a). Our aim here is to find the minimum set H of data hosts such that there exists an edge from a member of H to f for every $f \in F_j$ in G and no other set $H' \subset H$ satisfies that requirement. The set H is called the *minimal dominating vertex set* for graph G. However, it is possible that more than one such set exists for a graph. Our interest is in finding a minimal dominating set of datahosts such that the MCT is reduced.

From Figure 6(a), we can build a reduced adjacency matrix A for graph G wherein $a_{ik} = 1$ if data host d_i contains f_k. Such an adjacency matrix is shown in Figure 7(a). Therefore, the problem of finding the minimum dominating sets for G is now equivalent to finding the sets of the

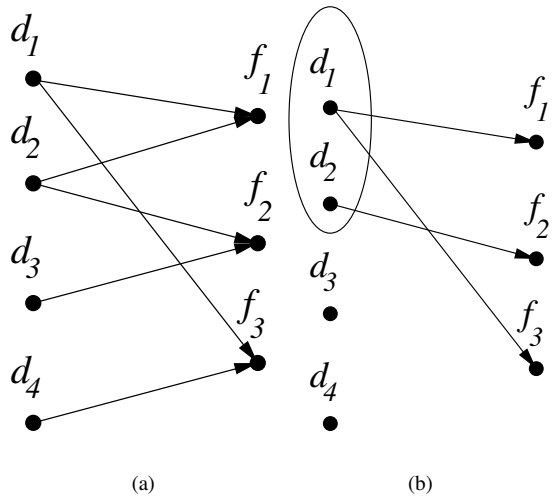


Figure 6. (a) Directed graph of data resources and data sets for job j.(b) A dominating set for the data graph.

$$\begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{array} \begin{array}{c} \begin{array}{ccc} f_1 & f_2 & f_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

(a)

$$\begin{array}{c} \\ d_1 \\ d_2 \\ - \\ d_2 \\ d_3 \\ - \\ d_1 \\ d_4 \end{array} \begin{array}{c} \begin{array}{ccc} f_1 & f_2 & f_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ - & - & - \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ - & - & - \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

(b)

Figure 7. (a) Adjacency Matrix. (b) Tableau.

least number of rows such that every column contains an entry of 1 in atleast one of the rows. Another way to look at the problem is to consider the data hosts as sets of datasets. Then, the problem becomes finding the minimum number of such sets (data hosts) such that all datasets can be "covered". This problem has been studied extensively as the *Set Covering Problem* [1].

Christofides [7] provides an approximate tree search algorithm for finding a solution to the general Set Covering Problem. Based on this algorithm, we propose a mapping heuristic to find a minimum dominating set that ensures the smallest makespan. The heuristic is listed in Figure 8.

At the start of the process, from the adjacency matrix, we create a tableau T consisting of K blocks of rows, where the k^{th} block consists of rows corresponding to data hosts that contain f_k. An example of a tableau generated from the adjacency matrix of Figure 7(a) is shown in Figure 7(b). The set of data hosts B keeps track of the current solution set of datahosts, the set E contains the datasets already covered by the solution set and the variable z keeps track of the makespan offered by the current solution set. The final solution set is stored in B_{final}. During execution, the blocks are searched sequentially starting from the k^{th} block where k is the smallest index, $1 \leq k \leq K$ such that $f_k \notin E$. Within the k^{th} block, let d_q^k mark the data host under consideration where q is a row pointer within block k. We add d_q^k to B and all the datasets for which the corresponding row contains 1, to E as they are already covered by d_q^k. These datasets are removed from consideration and the process then moves to the next uncovered block until $E = F_j$, that is, all the datasets have been covered. The function $MCT(B)$ computes the completion time for each compute resource combined with the solution set B and returns with the MCT so found.

Begin Main
Step 1. For a job J, create the adjacency matrix A with data hosts forming the rows and datasets forming the columns.
Step 2. Sort the rows of A in the descending order of the number of 1's in a row.
Step 3. Create the tableau T from sorted A and begin with initial solution set $B_{final} = \phi$, $B = \phi$, $E = \phi$ and $z = \infty$
Step 4. `Search(`B_{final}*,B,T,E,z*`)`
Step 5. $S^j \leftarrow \{\{r\}, B_{final}\}$ where $r \in R$ such that $MCT(B_{final})$ is minimum
End Main

`Search(`B_{final}*,B,T,E,z*`)`
Step 1. Find minimum k, such that $f_k \notin E$. Let T_k be the block of rows in T corresponding to f_k. Set a pointer q to the top of T_k.
while *q does not reach the end of* T_k **do**
  $F_T \leftarrow \{f_i | t_{qi} = 1 \wedge 1 \leq i \leq K\}$
  $B \leftarrow B \cup \{d_q^k\}, E \leftarrow E \cup F_T$
  if $E = F_j$ **then**
    if $z > MCT(B)$ **then**
      $B_{final} \leftarrow B, z \leftarrow MCT(B)$
  else `Search(`B_{final}*, B,T,E,z*`)`
  $B \leftarrow B - \{d_q^k\}, E \leftarrow E - F_T$
  Increment q
end

Figure 8: Listing for SCP-based Mapping Heuristic.

Through the recursive procedure outlined in the listing, the heuristic then backtracks and discovers other solution sets. The solution set that guarantees minimum makespan is then chosen as the final . The search terminates when the first block is exhausted. Therefore, before the tableau is created, we sort the rows of the adjacency matrix (that is, the data hosts) in the descending order of the number of columns with 1's (or the number of datasets contained). Also, in the tableau, the same sorting order is applied to the rows in each block. As the minimal dominating sets would obviously contain atleast one of the datahosts with the maximum number of datasets, this increases the chances of more

dominating sets being in the path of the search function within the proposed heuristic. Overall, the running time of the mapping heuristic is given by $O(MK^2)$ where MK^2 is the number of resource sets that are searched by the heuristic to find one that provides the least completion time.

Other heuristics that are possible or have been proposed include the ones described below:

Compute-First - In this mapping, a compute resource that ensures minimum computation time (T_c) is selected for the job first followed by choosing data hosts that have the best bandwidths to the selected resource. This is in contrast to our approach that places more importance on selection of data hosts. The running time of this heuristic is $O(MK)$.

Greedy - This heuristic builds the resource set by iterating through the list of datasets and making a greedy choice for the data host for accessing each dataset, followed by choosing the nearest compute resource for that data host. At the end of each iteration, it checks whether the compute resource so selected is better than the one selected in previous iteration when the data hosts selected previously are considered. This heuristic was presented in [23]. The running time of this heuristic is $O(KP)$.

Brute Force - In this case, all the possible resource sets for a particular job are generated and the one guaranteeing the MCT is chosen for the job. While this heuristic guarantees that the resource set selected will be the best for the job, it searches through MP^K resource sets at a time. This leads to unreasonably large search spaces for higher values of K. For example, for a job requiring 5 datasets with 20 possible data hosts and 20 available compute resources, the search space will consist of $64 * 10^6$ resource sets.

A point to note is that the sets of datasets required by 2 or more jobs in the same set are not mutually exclusive. Any dataset that is transferred during from one resource to another is retained at the receiver and therefore, this presents an additional source of data to successive jobs requiring access to that dataset.

4 Experiments

We have used GridSim with its new Data Grid capabilities [22] to simulate the data-intensive environment and evaluate the performance of scheduling algorithms. For evaluation, we have used the EU DataGrid topology based on the testbed shown in Figure 1. The details of the Grid resources used in our evaluation is shown in Table 1. All the resources were simulated as clusters with a batch job management system using space-shared policy, as a front-end to single CPU processing nodes. The CPUs are rated in terms of MIPS (Million Instructions Per Sec). The resource at CERN was considered as a pure data source (data host) in our evaluation and hence, no jobs were submitted to it. To model resource contention caused by multiple users, we

Table 1. Resources within EDG testbed used for evaluation.

Resource Name (Location)	No. of Nodes	CPU Rating (MIPS)	Storage (TB)	Load
RAL (UK)	41	1140	2.75	0.9
Imperial College (UK)	52	1330	1.80	0.95
NorduGrid (Norway)	17	1176	1.00	0.9
NIKHEF (Netherlands)	18	1166	0.50	0.9
Lyon (France)	12	1320	1.35	0.8
CERN (Switzerland)	–	–	12	–
Milano (Italy)	7	1,000	0.35	0.5
Torino (Italy)	4	1330	0.10	0.5
Catania (Italy)	5	1200	0.25	0.6
Padova (Italy)	13	1,000	0.05	0.4
Bologna (Italy)	20	1140	5.00	0.8

associate a *mean load* with each resource. The load factor is the ratio of the number of CPUs that are occupied to the total number of CPUs available within a resource. During the simulation, for each resource, we derive the instantaneous resource load from a Gaussian distribution with its mean as the load shown in Table 1.

Similarly, we model the variability of the available network bandwidth by associating an availability factor with a link which is the ratio of the available bandwidth to the total bandwidth. During simulation, the instantaneous measure is derived from another Gaussian distribution centered around a mean availability factor is assigned at random to each of the links.

Within this evaluation, we consider a universal set of datasets, each of which are replicated on one or more of the resources. Studies of similar environments [16] have shown that the size of the datasets follow a heavy-tailed distribution in which there are larger numbers of smaller size files and vice versa. Therefore, we generate the set of datasets with sizes distributed according to the logarithmic distribution in the interval $[1GB, 6GB]$. The distribution of datasets depends on many factors itself including variations in popularity, the replication strategy employed and the nature of the fabric. Within our evaluation, we have used two commonly considered patterns of file distribution:

- *Uniform :* Here, the distribution of datasets is modeled on a uniform random probability distribution. In this scenario, each file is equally likely to be replicated at any site.

- *Zipf :* Zipf-like distributions follow a power law model in which the probability of occurence of the i^{th} ranked file in a list of files is inversely proportional to i^{-a} where $a \leq 1$. In other words, a few files are distributed widely whereas most of files are found in one or two places. This models a scenario where the files are replicated on the basis of popularity. It has been shown that Zipf-like distributions holds true in cases such as requests for pages in World Wide Web where a few of the sites are visited the most [3]. This scenario has been evaluated for a Data Grid environment in related publications [4].

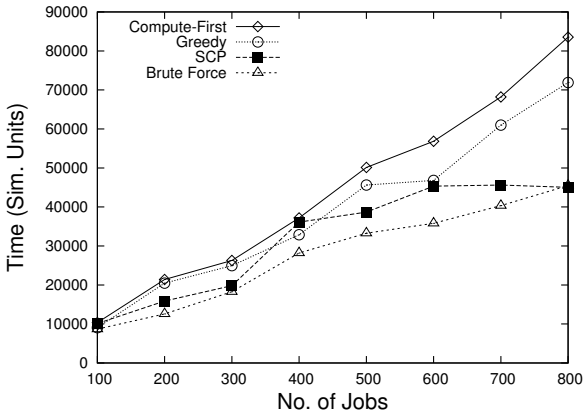
Henceforth, we will consider the distribution applied to be described by the variable $Dist$. We also control the distribution of datasets through a parameter called the *degree of replication* which is the maximum possible number of copies of any dataset in a Data Grid. The degree of replication in our evaluation is 5.

On the application side, there are three variables that determine the performance of the application: the size of the application or the number of jobs in the application (N), the number of datasets required by each job (K) and the computational size of a job ($Size(j)$) expressed in Million Instructions (MI). For each job, K datasets are selected at random from the universal set of datasets. For the purpose of comparison, we keep K a constant among all the jobs in a set although this is not a condition imposed on the heuristic itself. An experiment is described by the tuple $(N, K, Size, Dist)$. At the beginning of each experiment, the set of datasets, their distribution among the resources and the set of jobs are generated. This configuration is then kept constant while each of the four mapping heuristics are evaluated in turn. To keep the resource and network conditions repeatable among evaluations, we use the Colt random number generator [11] with a constant seed. As there are numerous variables involved, we have conducted evaluation with different values for $N, K, Size$ and $Dist$. We have conducted 50 such experiments and in the next section, we present results of our evaluation.

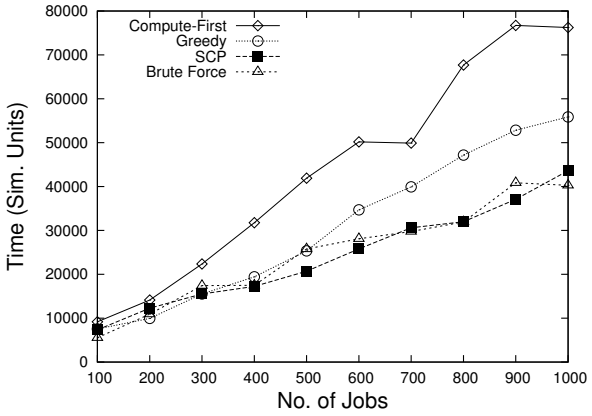
4.1 Results

Table 2. Summary of Evaluations

Mapping Heuristic	Geometric Mean	Avg. deg	Avg. rank
Compute-First	37593.71	69.01 (19.4)	3.63 (0.48)
Greedy	36927.44	71.86 (50.55)	3.23 (0.71)
SCP	24011.17	7.68 (10.42)	1.67 (0.6)
Brute Force	23218.49	3.87 (6.46)	1.47 (0.58)



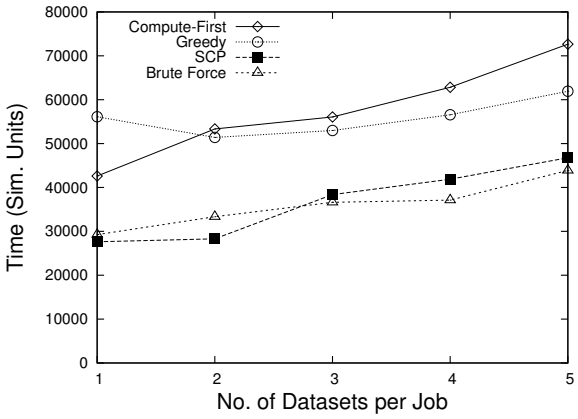
(a) Size=600000 MI,K=3, Dist=Uniform



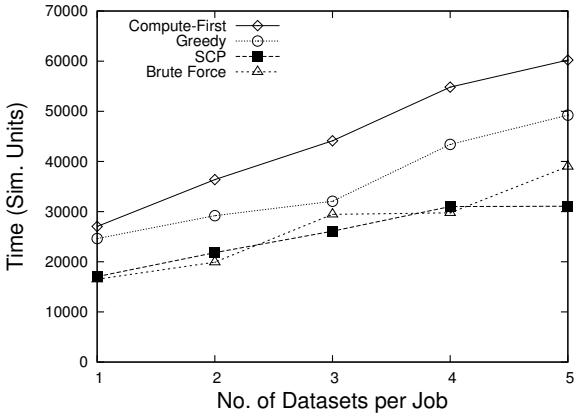
(b) Size=300000 MI,K=3, Dist=Zipf

Figure 9. Makespan vs Number of Jobs.

The results of our evaluations are summarised in Table 2 and are based on the methodology provided in [5]. *SCP* refers to the heuristic proposed in this paper. For each mapping heuristic, the table contains three values: 1) *Geometric Mean* of the makespans, 2) Average degradation *(Avg. deg.)* from the best heuristic and 3) Average ranking *(Avg. rank)* of each heuristic. The geometric mean is used as the makespans vary in orders of magnitude according to parameters such as number of jobs per application set, number of files per job and the size of each job. Degradation for a heuristic is the difference between the makespan of that heuristic and that of the best heuristic for a particular experiment and expressed as a percentage of the latter. The average degradation is computed as an arithmetic mean over all experiments and the standard deviation of the population is given in the parantheses next to the means in the table. This is the measure of how far a heuristic is away from the

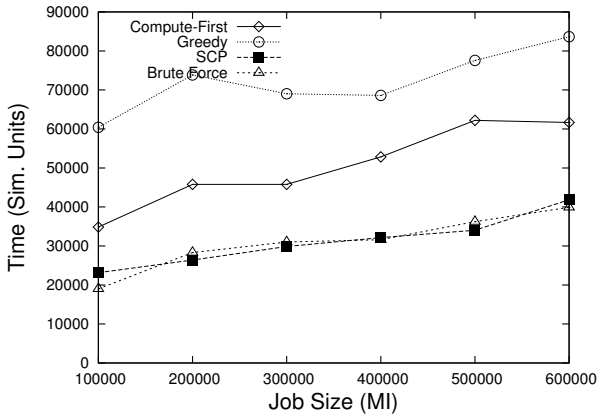


(a) N=600,Size=600000 MI, Dist=Uniform

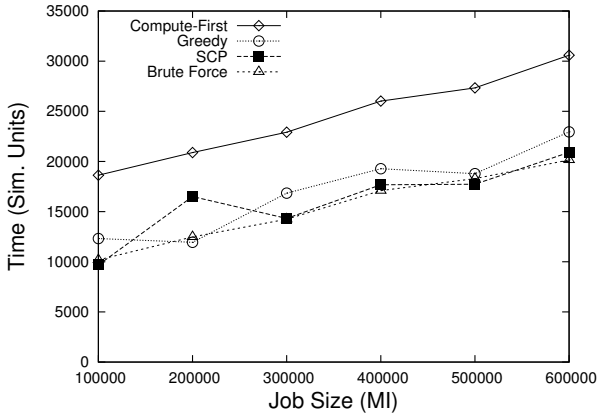


(b) N=600,Size=300000 MI, Dist=Zipf

Figure 10. Makespan vs Datasets per Job.



(a) N=600, K=5, Dist=Uniform



(b) N=300, K=3, Dist=Zipf

Figure 11. Makespan vs Job Size.

best heuristic for an experiment. A lower number certainly means that the application is on an average the best one. The ranking is in the ascending order of makespans produced by the heuristics, that is, lower the makespan, lower the rank of the heuristic. The standard deviation of the population is provided alongside the averages in the table.

The three values together provide a consolidated view of the performance of each heuristic. For example, we can see that on average Compute-First and Greedy both perform worse than either SCP or Brute Force. However, the standard deviation of the population is much higher in the case of Greedy than that of Compute-First. Therefore, it can also be said that Compute-First can be expected to perform worst most of time. Indeed, in a few of the experiments, Greedy performed as good or even better than SCP while Compute-First never came close to the performance of the other heuristics.

Between SCP and Brute Force, as expected, the latter is the clear winner having a consistently lower score than the former. However, the computational complexity of Brute Force means that as the number of datasets per job increases, the number of resource sets that need to be considered by the Brute Force heuristic increases dramatically. The geometric mean and average rank of SCP is close to that of Brute Force heuristic. The average rank is less than 2 for both heuristics which implies that in many scenarios, SCP provides a better performance than Brute Force.

This view is reinforced from the graphs in Figures 9-11 which show the effect of varying one of the variables, all others kept constant. SCP and Brute-force give almost similar performance while either of Greedy or Compute-First is the worst in almost all cases. The effect of job distribution is most visible on the Greedy heuristic. When the files are distributed according to the Zipf distribution, the perfor-

mance of Greedy comes close to or in some cases, becomes as competitive as SCP. This is due to the fact that in Zipf distribution, there are most of the datasets are not replicated widely and therefore, there is not as much choice of datahosts as there is in Uniform distribution. In such a case, Greedy is able to form minimal resource sets. Also, it can be seen that as the number of jobs increases, the makespan of Compute-First and Greedy heuristic rise more steeply than the other two.

5 Related Work

Casanova, et.al [5] extend three well-known scheduling heuristics, (*Max-min*, *Min-min* and *Sufferage*) that were introduced previously in [14] for scheduling independent tasks onto heterogeneous resources, to consider data transfer requirements . A fourth heuristic, *XSufferage*, was also introduced to take into consideration sharing of files between tasks. However, the source of all the files for the tasks is the resource that dispatches the jobs. Ranganathan, et. al [20] discuss a decoupled scheduling architecture that has two components: one schedules jobs to the resources and the other replicates data on such resources in anticipation of the incoming jobs. Similar studies have been performed for different replication strategies in [2]. Park and Kim [17] propose a scheduler which schedules jobs close to the source of data or else replicates the data to the job execution site. The difference between our work and the ones presented before is that we explicitly consider the scenario in which a job requires multiple datasets whereas the others are restricted to one dataset per job.

Giersch, et. al [10] present a follow-up to [5] where they consider the general problem of scheduling tasks requiring multiple files that are replicated on several repositories. They prove that this problem is NP-complete and propose faster heuristics that are competitive with XSufferage. However, the approach followed in their paper is that of scheduling the jobs first and then replicating the data so as to minimize access time. This general approach, also followed by the previous papers and which we have evaluated as **Compute-First**, may not produce the best schedules as has been shown in the evaluation. On the other hand, we consider the selections of computational and data resources to be interrelated. Genetic Algorithm (GA) based heuristics were introduced in [13] for scheduling decomposable tasks and in [18] for sets of independent jobs that have data requirements. We consider non-decomposable jobs that are individually mapped to a set of resources. However, with modification, GA can be used in our context and will be the subject of a future evaluation. Mohamed and Epema [15] present a Close-to-Files algorithm which searches the entire solution space for a combination of computational and storage resources that minimizes execution time. Their job model is restricted to one dataset per file. This approach, which we evaluate as **Brute Force**, produces good schedules but becomes unmanageable for large solution spaces that occur when more than one dataset is considered per job. In a previous publication [23], we have introduced the greedy mapping heuristic (**Greedy**) for the problem presented in this paper.

Similar studies have been conducted for parallel I/O in clusters and other distributed systems in the presence of data replication [21, 25]. However, we claim that the research presented in this paper is substantially different from these. For example, our heuristic favours those resources which have more of the datasets required for a job and generally tends to produce mappings that utilize the least number of datahosts possible. This is different from optimizing parallel I/O which generally tends to spread the data transfers across a larger number of resources to optimize bandwidth usage. However, we do recognize that parallel I/O techniques can be applied in the context of the problem presented and an investigation of these will be a part of our future work.

6 Conclusion and Future Work

We have presented the problem of mapping an application with a collection of jobs that require multiple datasets that are each replicated on multiple data hosts to Grid resources. We have also proposed a heuristic based on a solution to the Set Covering Problem. We have shown via simulation that the proposed heuristic is better than Compute-First and Greedy approaches and leads to schedules that are competitive with the exhaustive search approach while being orders of magnitude faster.

As part of immediate future work, we plan to evaluate our heuristic against the GA strategy as has been presented in related work. The performance of the SCP-based heuristic in scenarios involving dependent tasks such as Directed Acyclic Graphs (DAGs) also needs to be investigated. In the long term future, we would like to explore the use of parallel I/O optimization techniques in the problem space presented in this paper.

Acknowledgement

We would like to thank Anthony Sulistio for his help with the use of Gridsim and Tianchi Ma for his comments on the paper.

References

[1] E. Balas and M. W. Padberg. On the Set-Covering Problem. *Operations Research*, 20(6):1152–1161, 1972.

[2] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Simulation of Dynamic Grid Replication Strategies in OptorSim. In *Proceedings of the 3rd International Workshop on Grid Computing(GRID 02)*, pages 46–57, Baltimore,MD,USA, 2002. Springer-Verlag.

[3] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99.)*, 1999.

[4] D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini. Evaluating Scheduling and Replica Optimisation Strategies in OptorSim. In *Proceedings of the 4th International Workshop on Grid Computing (Grid2003)*, Phoenix, AZ, USA Nov. 2003. IEEE CS Press.

[5] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid environments. In *Proceedings of the 9th Heterogeneous Computing Systems Workshop (HCW 2000)*, Cancun, Mexico, 2000. IEEE CS Press.

[6] 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*, 23(3):187–200, 2000.

[7] N. Christofides. *Graph Theory: An Algorithmic Approach*, chapter Independent and Dominating Sets – The Set Covering Problem, pages 30 – 57. Academic Publishers, London, UK, 1975. ISBN 012 1743350 0.

[8] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, USA, 1999.

[9] R. Gardner et al. The Grid2003 Production Grid: Principles and Practice. In *Proceedings of the 13th Symposium on High Performance Distributed Computing (HPDC 13)*, Hawaii, HI, USA, June 2004. IEEE CS Press.

[10] A. Giersch, Y. Robert, and F. Vivien. Scheduling tasks sharing files from distributed repositories. In *Proceedings of the 10th International Euro-Par Conference*, volume 3149 of *LNCS*, pages 246–253. Springer-Verlag, Sept. 2004.

[11] W. Hoschek et al. The colt project. Available at `http://dsd.lbl.gov/~hoschek/colt/`.

[12] W. Hoschek, F. J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger. Data Management in an International Data Grid Project. In *Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing (GRID '00)*, Bangalore, India, Dec. 2000. Springer-Verlag.

[13] S. Kim and J. Weissman. A GA-based Approach for Scheduling Decomposable Data Grid Applications. In *Proceedings of the 2004 International Conference on Parallel Processing (ICPP 04)*, Montreal, Canada, Aug. 2003. IEEE CS Press.

[14] 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)*, 59:107–131, Nov 1999.

[15] H. Mohamed and D. Epema. An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, San Diego, CA, USA, Sept. 2004. IEEE CS Press,.

[16] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proceedings of the 1996 International Conference on Network Protocols (ICNP '96)*, Atlanta, GA, USA, 1996. IEEE CS Press.

[17] S.-M. Park and J.-H. Kim. 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, May 2003. IEEE CS Press.

[18] T. Phan, K. Ranganathan, and R. Sion. Evolving toward the perfect schedule: Co-scheduling job assignments and data replication in wide-area systems using a genetic algorithm. In *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing*, Cambridge, MA, June 2005. Springer-Verlag.

[19] A. Rajasekar, M. Wan, and R. Moore. MySRB & SRB: Components of a Data Grid. In *Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, page 301, Edinburgh, UK, 2002. IEEE CS Press.

[20] 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. IEEE CS Press.

[21] J. R. Santos, R. R. Muntz, and B. Ribeiro-Neto. Comparing random data allocation and data striping in multimedia servers. In *Proceedings of the 2000 ACM International conference on Measurement and modeling of computer systems (SIGMETRICS '00)*, pages 44–55, 2000. ACM Press.

[22] A. Sulistio, U. Cibej, B. Robic, and R. Buyya. A tool for modelling and simulation of data grids with integration of data storage, replication and analysis. Tech. Rep. GRIDS-TR-2005-13, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Nov. 2005.

[23] S. Venugopal and R. Buyya. A Deadline and Budget Constrained Scheduling Algorithm for e-Science Applications on Data Grids. In *Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2005)*, volume 3719 of *Lecture Notes in Computer Science*, Melbourne, Australia, Oct 2005. Springer-Verlag.

[24] 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)*, Toronto, Canada, Oct. 2004. ACM Press.

[25] J.-J. Wu and P. Liu. Distributed Scheduling of Parallel I/O in the Presence of Data Replication. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Denver,CO,USA, April 2005. IEEE CS Press.