Integrating Scheduling and Replication in Data Grids with Performance Guarantee

Lakshmi Ravi Anikode and Bin Tang
Department of Electrical Engineering and Computer Science
Wichita State University, Wichita, KS 67260
lxravianikode@wichita.edu, bintang@cs.wichita.edu

Abstract— **Data Grid consists of geographically distributed computing and storage resources that are used in large scale scientific applications. Job scheduling and data replication are two well-known techniques to boost the performance of Data Grid. There has been extensive research on integrating both techniques to further improve performance in Data Grid. However, most of the current work are heuristic based without performance guarantees. In this paper, we propose to integrate data replication and job scheduling into one framework to minimize the total job execution time in Data Grid. We refer to the problem as *Integrated Scheduling and Replication Problem.* This problem is NP-hard. We first propose a job scheduling and data replication algorithm, which not only has theoretically provable performance but also dramatically reduces the time complexity compared to that of the optimal algorithm. We then design a series of polynomial heuristic algorithms. Using extensive simulations, we demonstrate that among the heuristic algorithms, the integrated replication and scheduling algorithm performs closest to the one with performance guarantee.**

Keywords – Data Grids, Job Scheduling, Data replication, Algorithms

I. Background and Motivation

Data intensive scientific applications, such human genome mapping [15], high energy particle physics and astronomy [10], and climate change modeling [12], are drawing much attention from research community in recent years due to their potentially profound scientific and engineering impact. In such applications, large amounts of data are generated, accessed, and analyzed by scientists worldwide. The Data Grid [16], [1], [9] is an enabling technology for data intensive applications. It consists of hundreds of thousands of geographically distributed computation, storage, and networking resources to facilitate data sharing and management in distributed applications. One important functionality of Data Grids is to manage very large amount of data sets, in the order of terabytes and petabytes. On one hand, the data files required to run scientific applications on Data Grid has been growing at an unprecedented rate in both volume and scale; on the other hand, the size of data generated by such applications is continuing to increase exponentially.

Job scheduling and data replication are two effective techniques to enhance the performance of Data Grid. Job scheduling is to map and dispatch a set of jobs onto a set of sites for executions, to minimize the maximum job execution time (or makespan) among all the sites. Each job usually requires multiple input files for its execution. Therefore, when scheduling a job to a site, the data transfer time of its input files has to be taken into consideration. Data replication, on the other hand, is to create multiple copies of the popular data in the Grid to reduce the data transfer time and bandwidth consumption, and to minimize the total job execution time in the Data Grid.

Intuitively, minimizing makespan is to utilize all the resources of the Grid to its maximum potential from system administrators' perspective; while minimizing total execution time is to complete users' submitted jobs as soon as possible, which is from the users' perspective. These two techniques are complementary with each other: doing scheduling without replication places an overhead of data transfer time as job's input data files have to be fetched remotely, while doing replication without scheduling of jobs does not result in effective utilization of the Data Grid resources, as moving large-sized data costs more bandwidths and takes longer transfer time than moving jobs does. Therefore, integrating scheduling and replication to optimize the system performance in Data Grid has been an active research [14], [8], [4], [7], [3], [2], [6], [11].

However, most current research in this field consider job scheduling and data replication as two independent and parallel mechanisms in Data Grid; one with the objective of minimizing makespan and the other minimizing total job execution time. Since these two objectives are different, it is difficult to optimize both simultaneously. As a result, most of work are heuristic based without performance guarantees. We observe that since replication brings the data files closer to the jobs and scheduling moving the jobs closer to its input data, it is worthwhile to integrate them into one framework to achieve a single objective. In this work, we propose to utilize both scheduling and replication to minimize the total job execution time of the entire Data Grid. We refer to the problem as the *Integrated Scheduling and Replication Problem*. To tackle this problem, we propose a job scheduling and data replication algorithm that not only gives a constant ratio performance guarantee, but also reduces the time complexity dramatically compared to that of optimal solution. We further propose a set of more time efficient heuristics and show their performance is comparable to the job scheduling and data replication algorithm with performance guarantee. The main results and contributions of our paper are as follows:

1) To the best of our knowledge, our work is the first one to formally formulate and integrate job scheduling and data replication into one framework to minimize the total job execution time in Data Grid.

2) We propose a job scheduling and data replication algorithm with constant performance ratio and with time complexity that is much lower than that of the optimal algorithm.

3) We propose a set of heuristics and show through extensive simulations that their performances are comparable to the job scheduling and replication algorithm with performance guarantee. Among them, the integrated replication and scheduling algorithm performs closest to the one with performance guarantee.

II. Related Work

Ranganathan and Foster [14] have developed a family of job scheduling and data movement and replication algorithms, and use simulation studies to evaluate various combinations. They empirically show that while it is necessary to consider the impact of replication on the scheduling strategies, it is not always necessary to couple data movement and computation scheduling. Elghirani et al. [8] present an intelligent data grid framework where job scheduling and data/replica management are coupled to provide an integrated environment for efficient access to data and job scheduling. Specifically, they predict the appropriate locations of replica and proactively replicates the datasets coupled with Tabu Search. They show their techniques improve both the makespan and average job execution time compared to other heuristics.

Chang et al. [4] develop a job scheduling policy, called HCS (Hierarchical Cluster Scheduling), and a dynamic data replication strategy, called HRS (Hierarchical Replication Strategy), to improve the data access efficiency in a cluster grid. They simulate their algorithms to evaluate various combinations of data access patterns and implement in the Taiwan Unigrid environment. Their results show that HCS and HRS successfully reduce data access time in comparison with other strategies in a cluster grid. Chakrabarti et al. [3] propose to iteratively improve the performance based on the coupling between the scheduling and replication strategies. At the end of scheduling the popularity of the files required by a set of tasks are calculated and replication is done accordingly to facilitate ease of data access for the next set of tasks.

Gaurav et al. in [11] has detailed a task scheduling and file replication mechanism for a batch of data intensive tasks that exhibit batch-shared I/O behavior. Batch shared I/O behavior means the same file is the input of multiple tasks in a batch. A 0-1 Integer Programming based approach is formulated and a BiPartition heuristic that decouples scheduling and replication is proposed. Desperez et al. [7] combine data management and scheduling using a steady state approach. The problem is defined as a linear program. The objective is to maximize the throughput. Heuristic for approximating integer solution of the linear program does not give a good mapping of data, in worst cases it may be far from optimal unlike our approach where we prove our Job Scheduling and Data Replication algorithm with performance bound of a constant ratio to the optimal. Also their work does not take storage of resources as a constraint whereas we take the storage limit of resources as a constraint.

All above work attempt to integrate job scheduling and data replication in Data Grid. However, none of them has established a formal mathematical framework within which both techniques are seamlessly integrated to minimize the total job execution time of the Data Grid.

III. Problem Formulation

Network Model and Assumptions. A Data Grid is modeled as a undirected graph $G = (V, E)$ where $V = \{1, 2, .., n\}$ represents the set of sites in the Grid and E is the set of weighted edges which may represent transmission bandwidth, distance between sites, or loss rate. In our model the weight of an edge represents the bandwidth. In this paper, we assume all edges have uniform bandwidth B, but our techniques can be applied to varying bandwidth case too. There are m data files $F = \{f_1, f_2, ..., f_m\}$ in the Grid, with data file f_j originally produced and stored at its *source site* $S_j \in V$.[1] Let s_j be the size of file f_j. Let c_i be the storage capacity of site $i \in V$ (for source site i, c_i is the storage available after storing its original data files).

Data Grid users submit jobs to their own sites. All the submitted jobs are executed in FIFO order. There are n_i independent jobs $J_i = \{j_{i1}, j_{i2}, ..., j_{in_i}\}$ submitted in site i, and each job j_{ik} $(1 \leq k \leq n_i)$ requires $F_{ik} \subseteq F$ as input files for its execution. This initial job placement in the Grid is denoted as s_{init}, and site i is also called the *source site* of job j_{ik} $(1 \leq k \leq n_i)$. Let J denote the entire set of jobs in the Data Grid, i.e. $J = \bigcup_{i=1}^{n} J_i$. Let $q = |J|$ be the total number of jobs in the Data Grid. For any job, if some of the input files are not stored locally (at the site where the job is executed), they need to be transmitted from other sites that have copies of the file. Note that before any replication taking place, each data file is only stored in its source site.

The transmission time of sending data file $f_j \in F$ along any edge is s_j/B. Let t_{ij} be the number of transmissions needed to transmit a data file from site i to site j (which is equal to the number of edges between i and j). Therefore the time transmitting f_j from site i to j is $t_{ij} \times s_j/B$. As demonstrated by the experiments of Chervenak et al. [5], in data intensive applications, the time to execute a scientific job is mainly the time it takes to transfer the needed input files from source sites to local sites. Therefore, in this work, we assume that once the input files of a job are ready in the local storage, processing the job takes negligible time compared to the data file transmission time. We define the *job execution time* of a job as the sum of the transmission time fetching each input file from its source site. The *total*

[1] Note that a site can be the source site of multiple data files. We assume that the source site always keeps its initial data files.

job execution time of site i without data replication and job scheduling is the time taken to transmit all the input files of all its jobs from their source sites to site i:[2]

$$\sum_{k=1}^{n_i} \sum_{f_l \in F_{ik}} t_{iS_l} \times s_l / B. \quad (1)$$

Aggregate Demand. We define the *aggregate demand* of site i to data file f_j as the number of times site i accesses f_j as an input file to execute its jobs, and denote it as w_{ij}. $w_{ij} = \sum_{k=1}^{n_i} x_k$, where $x_k = 1$ if $f_j \in F_{ik}$ and $x_k = 0$ otherwise. Equation 1 can be rewritten as

$$\sum_{j=1}^{m} w_{ij} \times t_{iS_j} \times s_j / B. \quad (2)$$

The *total job execution time of the entire Data Grid without data replication and job scheduling* is the sum of the total job execution time of each site:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} \times t_{iS_j} \times s_j / B. \quad (3)$$

The objective of the *Integrated Scheduling and Replication Problem* is to minimize the total execution time of the Data Grid by replicating data files and scheduling jobs in the Data Grid. With data replication, multiple copies of the same data file exist in the Grid while the source site keeps the original copy of a data file; whereas for job scheduling, there is only one copy of each job, which is dispatched from its source site to another site to execute (note that a job can also be executed in its source site). *For each site, if the data file needed for job execution is not located in its local storage, it always accesses it from its closest site that has a replica copy*. Below is a formal definition of the Integrated Scheduling and Replication Problem.

Problem Formulation. A scheduling function is defined as $s : J \to V$, indicating that a job $i \in J$ is dispatched to node $s(i) \in V$ for execution. As mentioned, the initial job placement in the Grid is referred to as s_{init}. After job scheduling s, let $J_i^s = \{j_{i1}^s, j_{i2}^s, ..., j_{in_i^s}^s\}$ be the set of n_i^s jobs in site i, where job j_{ik}^s $(1 \le k \le n_i^s)$ needs a subset F_{ik}^s of F as its input files for execution. If we use w_{ij}^s to denote the aggregate demand of site i towards f_j after scheduling s, then $w_{ij}^s = \sum_{k=1}^{n_i^s} x_k$, where $x_k = 1$ if $f_j \in F_{ik}^s$ and $x_k = 0$ otherwise.

The *total demand* of each file f_j, which is defined as the total number of time f_j being accessed by all the Grid sites to execute their jobs, does not change before and after each job scheduling s. That is, $\sum_{i=1}^{n} w_{ij} = \sum_{i=1}^{n} w_{ij}^s$ for any data file f_j. We call this *file demand constraint*.[3] With the job scheduling s and without data replication, the total execution time of the Data Grid is

$$\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij}^s \times t_{iS_j} \times s_j / B. \quad (4)$$

The integrated scheduling and replication problem in the Data Grid is to find a scheduling function s, and to select a set of sets $R = \{R_1, R_2, ..., R_m\}$ where $R_j \subseteq V$ is a set of Grid sites (or *replica set*) that contains a replica copy of file f_j, to minimize the *total execution time in the Data Grid:*

$$\tau(G, R, s) = \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij}^s \times \min_{k \in (S_j \cup R_j)} t_{ik} \times s_j / B, \quad (5)$$

under the storage capacity constraint that

$$|\{R_j | i \in R_j\}| \le c_i, \qquad \forall i \in V,$$

and file demand constraint that

$$\sum_{i=1}^{n} w_{ij} = \sum_{i=1}^{n} w_{ij}^s, \qquad \forall j \in J.$$

This problem in NP-hard, since the data replication problem without scheduling is NP-hard [13].[4] In next section, we propose a job scheduling and data replication algorithm and show its performance guarantee. We also present a series of efficient heuristic algorithms.

IV. Job Scheduling and Data Replication Algorithms in Data Grids

We first present a job scheduling and data replication algorithm with performance guarantee. Then we show a suite of polynomial heuristics, each of which integrates job scheduling and data replication in a different way.

Job Scheduling and Data Replication Algorithm With Performance Guarantee. For a Grid with total n sites and q jobs, there are n^q scheduling outputs (or job placements). For each of the scheduling output, we run a *greedy data replication algorithm* (line 4 to line 11 in Algorithm 1) as follows. First, all Grid sites have all empty storage space (except for sites that originally produce and store some files). Then, at each step, it places one data file into the storage space of one site such that the reduction of total job execution time in the Data Grid is maximized at that step. The algorithm terminates when all storage space of the sites has been occupied by replicated data files, or the total execution time in the Data Grid cannot be reduced further. We return the set of replica sets for each data file and the job placement that give the minimum total execution time. Below is the job scheduling and data replication algorithm.

Algorithm 1: Job Scheduling and Data Replication Algorithm

[2]Note that without data replication taking place in the local storage of site i, the same data file needed by different jobs in i needs to be accessed and transmitted multiple times from its source site. We show in following text that good replication strategy can be found using the aggregate demand towards a file from a site.

[3]However, because the input files of each job are the same before and after scheduling s, w_{ij}^s cannot change arbitrarily while just satisfying $\sum_{i=1}^{n} w_{ij} = \sum_{i=1}^{n} w_{ij}^s$, for any data file f_j.

[4]Note that however, when each job has only one input file, the problem becomes trivial, since all the jobs with input file f_j can all be scheduled to the source site of f_j, resulting in zero total execution time.

BEGIN

1. $R = \emptyset$ (empty set); $s_{min} = s_{init}$; τ_{min} = infinite;
2. **for** each of the n^q job scheduling s
3. $R_1 = R_2 = \ldots = R_m = \emptyset$ (empty set);
4. **while** (the total job execution time can still be
5. reduced by replicating data files in some sites)
6. Among all sites with available storage and
7. all data files, let replicating data file f_i on
8. site l give the maximum
9. $\tau(G, \{R_1, R_2, \ldots, R_i, \ldots, R_m\}, s) -$
10. $\tau(G, \{R_1, R_2, \ldots, R_i \cup \{l\}, \ldots, R_m\}, s)$;
11. $R_i = R_i \cup \{l\}$;
12. **end while;**
13. If $(\tau(G, \{R_1, R_2, \ldots, R_m\}, s) < \tau_{min})$
14. $R = \{R_1, R_2, \ldots, R_m\}$;
15. $s_{min} = s$;
16. $\tau_{min} = \tau(G, \{R_1, R_2, \ldots, R_m\}, s)$;
17. **end for;**
18. **RETURN** R, s_{min}, and τ_{min};

END. ◊

The time complexity of Algorithm 1 is $O(m^2 n^{3+q} \overline{c})$, where n, m, and q are the number of sites, data files, and jobs in the Data Grid, respectively, and $\overline{c}$ is the average storage capacity of a site. Before we show Algorithm 1's performance guarantee, we first present the exhaustive optimal scheduling and replication algorithm.

Optimal Job Scheduling and Data Replication Algorithm. For a Data Grid with n sites and q jobs, m data files of unit size, and each site has storage capacity $\overline{c}$, the optimal algorithm is for each of the n^q job placements, to enumerate all possible file replication placements in the Data Grid, and find the job placement and replication placement combination with minimum total job execution time. There are $(C_m^{\overline{c}})^n$ such file placements, where $C_m^{\overline{c}}$ is the number of ways selecting $\overline{c}$ files out of m files. Therefore the time complexity of the optimal algorithm is $O((C_m^{\overline{c}})^n \times n^q)$, which is much higher than that of above integrated scheduling and replication algorithm.

Performance Guarantee of Algorithm 1. Below we show that τ_{min} resulted in Algorithm 1 is close to the optimal total execution time with some constant performance ratio. Before we show that Algorithm 1 yields a constant performance ratio, we first present our previous result about the data replication algorithm (line 4 to line 11 in Algorithm 1) for a fixed scheduling s [13].

Theorem 1: For any fixed job scheduling s, and data size being unit one, if the total job execution time without replication is less than 40 times the optimal job execution time, then the total job execution time yielded by the data replication algorithm is less than 20.5 times the optimal total job execution time.

Now we show the performance guarantee yielded by Algorithm 1.

Theorem 2: For each of the n^q job scheduling solutions, if the total job execution time without replication is less than 40 times the optimal job execution time using optimal data replication algorithm, then the minimum total job execution time τ_{min} yielded by Algorithm 1 is less than 20.5 times the optimal total job execution time.

Proof: Let O be the optimal total job execution time, i.e., the minimum job execution time among all the $(C_m^{\overline{c}})^n \times n^q$ job scheduling and data replication combinations. Without loss of generality, assume that the x^{th} job scheduling is the job scheduling in the optimal solution. Let τ_x be the total job execution time corresponding to the x^{th} job scheduling in Algorithm 1. Let O_x be the optimal total job execution time corresponding to the x^{th} job scheduling in Algorithm 1, then $O_x = O$. Let τ_{min} be the minimum total job execution time yielded by Algorithm 1, then $\tau_{min} \leq \tau_x$. From Theorem 1, we have $\tau_x < 20.5 O_x$. We have therefore, $\tau_{min} \leq \tau_x < 20.5 O_x = 20.5 O$. ∎

Heuristic Algorithms. Even though the time complexity of Algorithm 1 is much lower than that of the optimal, it is still exponential. Therefore, we develop a suite of heuristics with polynomial time complexity.

Algorithm 2: Replication + Scheduling Heuristic

BEGIN

$R = \emptyset$ (empty set); $s_{min} = s_{init}$; τ_{min} = infinite;
$R_1 = R_2 = \ldots = R_m = \emptyset$ (empty set);
while (the total job execution time can still be
reduced by replicating data files in some sites)
Among all sites with available storage and
all data files, let replicating data file f_i on site l
give the maximum
$\tau(G, \{R_1, R_2, \ldots, R_i, \ldots, R_m\}, s_{init}) -$
$\tau(G, \{R_1, R_2, \ldots, R_i \cup \{l\}, \ldots, R_m\}, s_{init})$;
$R_i = R_i \cup \{l\}$;
end while;
for each job $i \in J$
Map job i to site j such that total execution time
for job i is minimized;
Update s_{min} and τ_{min};
end for;
RETURN R, s_{min}, and τ_{min};

END. ◊

In Algorithm 2, we first run the greedy data replication algorithm for initial job placement s_{init}. Then, we schedule each job to a site that gives the minimum execution time for that job. The time complexity of the greedy replication is $O(m^2 n^3 \overline{c})$, and time complexity of the scheduling is $O(nq)$. So the time complexity of Algorithm 2 is $O(m^2 n^3 \overline{c} + nq)$.

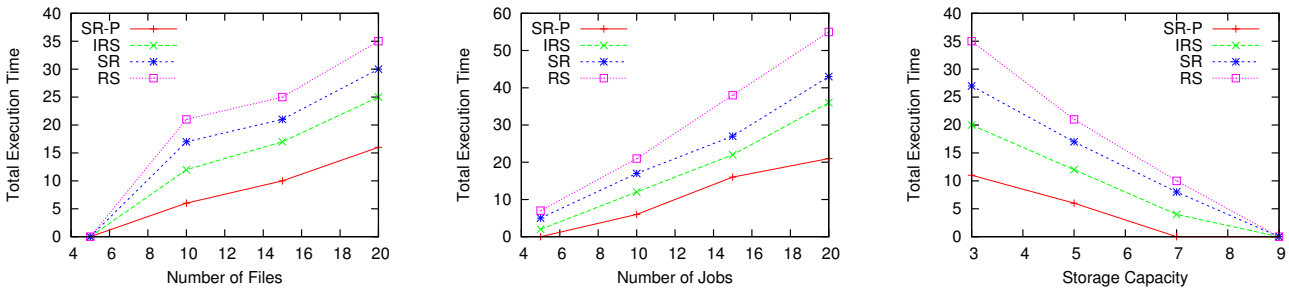
Algorithm 3: Scheduling + Replication Heuristic

BEGIN

First schedule each job to a site that gives the
minimum execution time for that job based on the
initial data file placement. Then run the greedy
data replication strategy for this job scheduling.

END. ◊

Algorithm 3 changes the order of the executions of the data replication and scheduling algorithms in Algorithm 2. The time complexity is the same as that of Algorithm 2.



(a) Varying number of data files. (b) Varying number of jobs. (c) Varying storage capacity.

Fig. 1. Performance comparison of SR-P, IRS, SR, and RS in a small Grid of 5 sites. Unless varied, the number of data files is 10, the number of jobs is 10, and storage capacity of each site is 5. Data file size is 1.

Algorithm 4: Integrated Replication and Scheduling Heuristic

BEGIN
  $R = \emptyset$ (empty set); $s_{min} = s_{init}$; τ_{min} = infinite;
  $R_1 = R_2 = \ldots = R_m = \emptyset$ (empty set);
  while (the total job execution time can still be reduced by replicating data files in some sites)
    Among all sites with available storage and all data files, let replicating data file f_i on site l give the maximum
    $\tau(G, \{R_1, R_2, \ldots, R_i, \ldots, R_m\}, s_{min}) -$
    $\tau(G, \{R_1, R_2, \ldots, R_i \cup \{l\}, \ldots, R_m\}, s_{min})$;
    $R_i = R_i \cup \{l\}$;
    for each job $i \in J$
      Map job i to site j such that total execution time for job i is minimized;
      Update s_{min} and τ_{min};
    end for;
  end while;
  RETURN R, s_{min}, and τ_{min};
END. ◊

Algorithm 4 takes place in rounds. In each round, first a data file is replicated into one site such that the reduction of total job execution time in the Data Grid is maximized, then we schedule each job to a site that gives the minimum execution time for that job (note if the current site of the job gives its minimum execution time, the job is not scheduled to other sites). The process of one replication followed by a scheduling terminates when all storage space of the sites has been replicated with data files, or the total job execution time cannot be reduced further. The time complexity of this algorithm is $O(m^2n^3\bar{c} \times nq) = O(m^2n^4q\bar{c})$, which is much lower than that of Algorithm 1 while higher than those of Algorithm 2 and 3.

V. Performance Evaluation

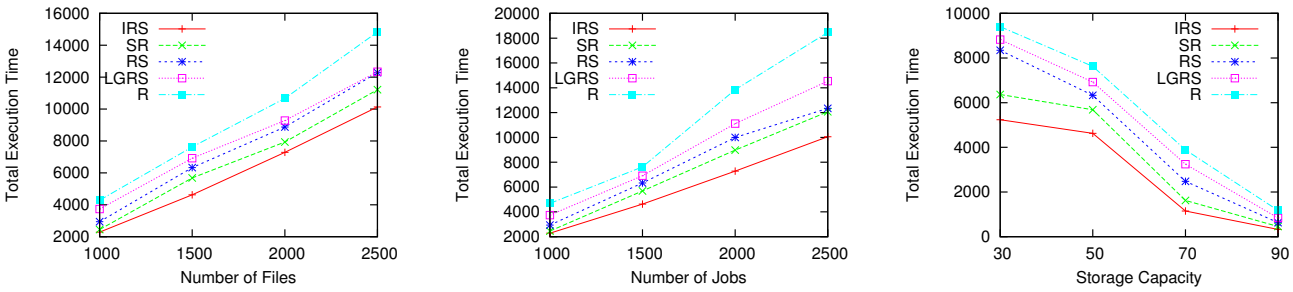
In this section, we refer to Algorithm 1 as **SR-P**, Algorithm 2 as **RS**, Algorithm 3 as **SR**, and Algorithm 4 as **IRS**. We first compare different heuristics (RS, SR, and IRS) with SR-P. Then we compare all the heuristics in large scale.

Comparing Heuristics With SR-P. Due to SR-P's high time complexity we experiment on a small Grid with 5 sites. We vary the number of files in the Grid, number of jobs in the Grid, and storage capacity of each site. Unless otherwise varied, we set number of files $m = 10$, number of jobs $q = 10$, and storage capacity of each site $c = 5$. Here we assume each file has unit size. Initially one copy of each file is placed randomly in any of the site. We assume that each job has five input files, which are randomly chosen from all the files. The jobs are also randomly submitted at each site.

Varying Number of Data Files. We vary m to be 5, 10, 15, and 20. As shown in Figure 1 (a), with the increase of the number of files the total job execution time in Data Grid increases for all the algorithms, with SR-P $<$ IRS $<$ SR $<$ RS. This demonstrates that SR-P performs better than other heuristics due to its performance guarantee. We observe that IRS performs next to SR-P, because IRS constantly couples scheduling with replication for the purpose of minimizing total execution time. We also observe that SR performs better than RS. In SR, small-sized jobs are first dispatched closer to their input files, which results in less data replication and movement; whereas in RS, large-sized data files are replicated and moved closer to their jobs, which takes much more time than moving jobs. It also shows that when the storage capacity of each site is five and the total number of files is five, all four algorithms eventually replicate all five files into each Grid site, resulting in zero total execution time.

Varying Number of Jobs. We vary the number of jobs as 5, 10, 15, and 20 while keeping the number of files 10 and storage capacity 5. As seen in Figure 1 (b), with the increase of the number of jobs, the total execution time increases for all the algorithms. This is as expected since as more jobs need to be executed, more data files need to transmit from remote site to local sites. Again, we observe that SR-P $<$ IRS $<$ SR $<$ RS.

Varying Storage Capacity. We vary c from 3, 5, 7, to 9 units, as shown in Figure 1(c). It is obvious that with the increase of memory capacity of each Grid site, the total execution time decreases since more file replicas are able to be placed into the Data Grid. We observe that when $c = 9$, the total execution time becomes zero. This is because in average, each of the five site already has two of 10 data



(a) Varying number of data files. (b) Varying number of jobs. (c) Varying storage capacity.

Fig. 2. Performance comparison of IRS, SR, RS, LGRS, and R in a Grid with 30 sites. Unless varied, the number of data files is 1500, the number of jobs is 1500, and storage capacity of each site is 50. Data file size is 1.

files, therefore for each of the site, its storage space of 9 can store the rest 8 data files.

Comparing Heuristics in Large Scale. We study the scalability of different heuristics by considering a Data Grid with 30 sites, 500 to 2,000 data files, storage capacity of each site varying from 30 to 90, and 1000 to 2500 jobs. We also compare the heuristics with a) the greedy data replication algorithm only **R** (that is, using only initial job placement), and b) local greedy replication and scheduling algorithm **LGRS**: for each site, the data files are replicated locally in the ascending order of their access frequencies until the storage of the site is full; then each job is scheduled to a site that minimizes its job execution time. Figure 2 shows that for all different scenarios, IRS $<$ SR $<$ RS $<$ LGRS $<$ R.

VI. Conclusion and Future Work

We have formulated and studied the problem of integrating job scheduling and data replication in data intensive scientific applications. The objective is to minimize the total job execution time by placing the data replicas and scheduling jobs onto the right sites. We designed a job scheduling and data replication algorithm with provable performance guarantee. We also developed a set of efficient heuristics and validated our results with simulations and analysis. As ongoing and future directions, we are exploring polynomial-time approximation algorithm that integrates both scheduling and replication. We also plan to implement the integrated algorithm in a distributed environment. Finally, this work focuses on scheduling independent jobs; the algorithms can be extended to incorporate job dependencies in workflow environment.

VII. Acknowledgements

The research described in this paper has been partially supported by NSF Grants CNS-1116849 and EPS-0903806. We also thank Dr. Himanshu Gupta and the anonymous reviewers for the helpful suggestions.

References

[1] B. Allcock, J. Bester, J. Bresnahan, A.L. Chervenak, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, and I. Foster. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *Proc. of IEEE Symposium on Mass Storage Systems and Technologies*, 2001.

[2] W.H. Bell, D.G. Cameron, R. Cavajal-Schiaffino, K. Stockinger, and F. Zini. Analysis of scheduling and replica optimization strategies for data grids using optorsim.

[3] R. Chakrabarti, A. Dheepak, and S. Sengupta. Integration of scheduling and replication in data grids. In *Proc. of In International Conference on High Performance Computing (HiPC)*, 2004.

[4] R.S. Chang, J.S. Chang, and S.Y. Lin. Job scheduling and data replication on data grids. *Future Generation Computer Systems*, 23:846–860, 2007.

[5] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, and B. Moe. Wide area data replication for scientific collaboration. In *Proc. of IEEE/ACM International Workshop on Grid Computing (Grid 2005)*.

[6] N.N Dang and S.B. Lim. Combination of replication and scheduling in data grids. *International Journal of Computer Science and Network Security*, 7:304–308, 2007.

[7] F. Desprez and A. Vernois. Simultaneous scheduling of replication and computation for data-intensive applications on the grid. *Technical Report*, 2005.

[8] A. Elghirani, R. Subrata, and Albert Y. Zomaya. Intelligent scheduling and replication in datagrids: a synergistic approach. In *Proc. of Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, 2007.

[9] I. Foster. The grid: A new infrastructure for 21st century science. *Physics Today*, 55:42–47, 2002.

[10] J. C. Jacob, D.S. Katz, T. Prince, G.B. Berriman, J.C. Good, A.C. Laity, E. Deelman, G.Singh, and M.-H Su. The montage architecture for grid-enabled science processing of large, distributed datasets. In *Proc. of the Earth Science Technology Conference*, 2004.

[11] G. Khanna, N. Vydyanathan, U. V.Catalyurek, T. M. Kurc, S. Krishnamoorthy, P. Sadayappan, and J. H. Saltz. Task scheduling and file replication for data-intensive jobs with batch-shared io. In *Proc. of the 15th IEEE International Symposium on High-Performance Distributed Computing (HPDC-15)*, 2006.

[12] M. Mineter, C. Jarvis, and S. Dowers. From stand-alone programs towards grid-aware services and components: A case study in agricultural modelling with interpolated climate data. *Environmental Modelling and Software*, 18(4):379–391, 2003.

[13] D.T Nukarapu, B. Tang, L. Wang, and S. Lu. Data replication in data intensive scientific applications with performance guarantee. *IEEE Transactions on Parallel and Distributed Systems*, In Print.

[14] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proc. of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, 2002.

[15] A. Rodriguez, D. Sulakhe, E. Marland, N. Nefedova, M. Wilde, and N. Maltsev. Grid enabled server for high-throughput analysis of genomes. In *Proc. of Workshop on Case Studies on Grid Applications*, 2004.

[16] S. Venugopal, R. Buyya, and K. Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys*, 38(1), 2006.