INTEGRATED REPLICATON AND SCHEDULING IN DATA GRIDS WITH PERFORMANCE GUARANTEE

A Thesis by

Lakshmi Ravi Anikode

Bachelors in Computer Science, Osmania University, 1996

Masters in Computer Applications, Osmania University, 1999

Submitted to the Department of Electrical Engineering and Computer Science

and the faculty of the Graduate School of

Wichita State University

in partial fulfillment of

the requirements for the degree of

Master of Science

May 2011

INTEGRATED REPLICATION AND SCHEDULING IN DATA GRIDS WITH PERFORMANCE GUANRANTEE

The following faculty members have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

_____

Bin Tang, Committee Chair

_____

Mehmet Yildirim, Committee Member

_____

Vinod Namboodiri, Committee Member

DEDICATION

This thesis is dedicated to my parents, my husband and my children.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Dr. Bin Tang, for guiding me in this work. I thank him for his patience, encouragement and motivation throughout the course of my study and research. He has always supported and encouraged me to do my best. I would also like to thank Mr. Keenan Jackson, who has extended advices and support thorough out my coursework.

I would like to thank my parents, parents in law, husband, children and my friends for their love, patience, help and encouragement.

ABSTRACT

Data Grid consists of geographically distributed computing and storage resources that are used in large scale scientific applications such as high energy physics, bioinformatics, climate modeling. Scheduling and Replication are two well-known techniques to boost the performance of Data Grid. There has been research on integrating both the techniques in Data Grids to improve performance. However, most of the work is heuristic based. In their work, data replication is used to minimize the file transfer time thus total job execution time of all the sites, while scheduling is used to minimize the maximum job execution time (so called makespan) among all the sites. We propose to utilize both data replication and job scheduling to minimize the total job execution time in Data Grid, and formulate our Data Replication and Job Scheduling Problem. Unlike previous work, our problem seamlessly integrates both techniques into one framework. This problem is NP-hard. We first propose a Job Scheduling and Data Replication algorithm whose performance is provable theoretically, and which also dramatically reduces time complexity compared to that of the optimal algorithm. We then design a series of heuristic algorithms to further reduce the time complexity of our Job Scheduling and Data Replication algorithm. Using simulations, we demonstrate that the heuristic algorithms perform comparably to the Job Scheduling and Data Replication algorithm.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

Much of the scientific research and applications currently pursued such as global climate change, high energy physics, climate modeling, earthquake monitoring, bioinformatics and astrophysics are data intensive. The data requirements for these scientific applications have been growing at an unprecedented rate in both volume and scale with huge input data sets. Data output generated by these applications, typically in the range of petabytes which is continuing to increase exponentially, are widely accessed globally for research and analysis. The Data Grid has emerged as a new technology to unify the numerous geographically distributed computation, storage and networking resources that offer comprehensive solutions for data intensive applications.

The major functionalities of Data Grid include high performance storage mechanism, reliable data transfer mechanism with low latency transfer protocols, scalable replica discovery and management mechanism that need to access, process and transfer large data sets stored in distributed repositories [1][2]. In data intensive applications data plays an important role. The jobs submitted by the users in these applications require huge input data sets distributed geographically and transferring these large-sized data takes tremendous amount of time. The data transfer time also depends on the site chosen for executing the job. If the input data sets for the job are located remotely in a storage site, then the time taken to transfer these data sets to the site of execution is huge and cannot be ignored. Performance is further throttled by the underlying network, bandwidth consumption, and failure of nodes.

Scheduling and replication are two effective mechanisms that can enhance the performance of Data Grid. Scheduling is to map a set of independent jobs each of which requires

multiple input files onto a set of sites to meet certain objectives under constraints. The objective of scheduling is usually to minimize the maximum job completion time among all the sites also called makespan. Minimizing makespan is to utilize all the resources of the grid to its maximum potential thus making it an objective from system administrator perspective. When scheduling a job to a site the data transfer time has to be taken into consideration. Replication is a technique that creates multiple copies of the most accessed data in selected sites thereby reducing the data transfer time and bandwidth consumption. Minimizing total execution time is from the users' perspective to complete their jobs as soon as possible. Job is executed much faster when there are more replica copies of the data available, because in data intensive applications data transfer time dominates job execution time.

Integrating scheduling and replication to optimize the system performance in Data Grid has been an active research. Doing scheduling without replication places an overhead of data transfer time as each of the file in job's input data set has to be fetched remotely. Doing replication without scheduling of jobs does not result in effective utilization of all the Data Grid resources, as moving large-sized data costs more bandwidths and takes longer transfer time than moving jobs does. So it is essential to integrate the two as both the processes are complementary with replication bringing the input data sets closer to the jobs and scheduling moving the jobs closer to its input data sets. So in this work we focus on integrating scheduling and replication to boost the performance of the Data Grid.

Most of the previous work on integrating scheduling and replication have considered them as two independent mechanisms; 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; therefore most of them are heuristic based without performance guarantee. In

this work we propose to integrate scheduling and replication into one framework for the sole goal of minimizing the total job execution time of the entire Data Grid. We propose a Job Scheduling and Data Replication algorithm which gives us a constant ratio performance guarantee at the same time reducing the time complexity dramatically compared to optimal solution. We further propose a set of more time efficient heuristics and show their performance is comparable to our Job Scheduling and Data Replication algorithm. We validate our results by extensive simulations.

1.1 Contribution to Thesis

1. To the best of our knowledge, our work is the first one to formally formulate and study the Integrated Replication and Scheduling Problem in one framework with the objective of minimizing the total job execution time in Data Grid.
2. We propose a Job Scheduling and Data Replication algorithm with constant performance ratio whose time complexity much lower than that of the optimal.
3. We propose a set of heuristics and show through simulations that the performance is comparable to our job scheduling and replication algorithm.

Chapter 2

LITERATURE REVIEW

2.1 Related Work

Related work can be viewed from the aspects of scheduling and replication. In Data Grids each of the technique has been a widely researched topic. Scheduling has been studied in [4-13] and replication in [14-17]. However in this work we focus on those works that combine the two techniques [18-25].

Ranganathan and Foster [18] have considered an approach by decoupling computation from data scheduling. The paper proposes a set of independent scheduling and replication algorithms. They describe four scheduling algorithms namely JobRandom, JobLeastLoaded, JobDataPresent and JobLocal and three replication algorithms namely DataDoNothing, DataRandom, DataLeastLoaded. They have performed evaluations based on each combination of a scheduling and a replication algorithm. These algorithms are designed where a job requires a single input file. In our work we consider a more realistic environment where a job requires multiple input files which are distributed across different sites. The scheduling logic in this case has to take the minimum data transfer time of all the files in the data set into picture when calculating the job execution time in a site.

Zomanya et al. in [19] define a replica framework which contains an application analyzer, application manger, replica manger and replica placement service to minimize the data access cost. Replicas are decided based on data access patterns, correlation, rank, data locations, and user and application behavior. Scheduling is done based on the popular tabu search heuristic that utilizes the replication information to dispatch tasks to resources to meet the objective of minimizing the makespan.

Chang et al. [20] developed the Hierarchical Cluster Scheduling algorithm (HCS) and Hierarchical Replication Strategy (HRS). The algorithm aims at creating local copies of data so that they can be fetched quickly. The idea of the HRS algorithm is to maximize the data availability within the cluster by creating replicas. The HCS algorithm takes into account the computational capacity, data location and also clusters' information as input to get the data availability within it.

Job scheduling and data placement are coupled in [22]. The authors have proposed an Integrated Replication and Scheduling strategy. Jobs and data are scheduled alternatively. The jobs are scheduled by the job scheduling algorithm optimizing the average job latency. At the end of scheduling the popularity of the files required by a set of tasks are calculated and replication is done to facilitate ease of data access for the next set of tasks. The drawback in the algorithm is that the next set of jobs may or may not access the same data sets.

In [23] Bell et al. have implemented the optimization techniques for scheduling and replication in a simulator called OptorSim and evaluated the results. Dang et al. in [24] have proposed heuristics for scheduling and replication independently. But they need to integrate the scheduling system with the replication to perform as a whole system together; the interaction between both is not detailed in the paper.

In all these work deciding replicas and location to put the replicas employ heuristics based on access patterns of data. In our work we not only consider the data access patterns but also use it to implement a replication technique with provable performance guarantee.

Gaurav et al. in [25] 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 Bi Partition heuristic that decouples scheduling and replication is proposed.

In [21] Desperez et al. 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. Below is a table comparing the features of the above literature.

References	Network Model	Scheduling			Replication			
		Centralized / Distributed	Objective	Methodology	Centralized / Distributed	Static/ Dynamic	Objective	Methodology
Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications [18]	Hierarchical topology, homogeneous network, single input file for each job	Distributed	Minimize Average job completion time	Random, Least Loaded, Data Available, Local scheduling	Distributed	Dynamic	Minimize total access cost	Do Nothing, Random, Least Loaded neighbor
Integration of Scheduling and Replication in Data Grids[22]	Hierarchical topology	Centralized	Minimize queuing latency	Match and Cost based Scheduling	Centralized	Dynamic	Minimize latency to access files	Replication based on data demand after a round of job scheduling
Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid[21]	Cluster topology, homogeneous network	Centralized, static	Maximize throughput	Integer Approximation Solution, Greedy Approximation	Centralized	Static	Maximize throughput	Integer Approximation Solution, Greedy Approximation
Job scheduling and data replication on data grids[20]	Hierarchical topology	Centralized	Minimize Total job execution time	Match based scheduling	Centralized	Dynamic	Minimize inter cluster communication cost	Replicates all the files required by job locally, LFU for replica replacement
Intelligent scheduling and replication: a synergistic approach [19]	Hierarchical topology	Centralized	Minimize makespan	Tabu Search	Dynamic	Centralized	Minimize access cost	Algorithms for replica placement and caching based on rank and correlation of data

Table 1: Summary of Previous Work on Integrating Scheduling and Replication

While the above work considers scheduling and replication as two separate mechanisms and proposes heuristics for each, ours is the first one to consider them as a combined mechanism that work together to achieve one objective of minimizing the total execution time. Thus in this work we integrate them in the true sense.

2.2 Organization of Thesis

The remaining thesis is organized as follows. In Chapter 3, we state the assumptions in the grid model and formulate the problem. In Chapter 4, we first propose a Job Scheduling and Data Replication algorithm and show its constant performance ratio. Even though it has a much lower time complexity than the optimal algorithm, it is still an exponential algorithm. Therefore we propose a series of heuristic algorithms and discuss their time complexity. In Chapter 5 we show the simulation results by comparing our Job Scheduling and Data Replication algorithm with the set of heuristics for different parameters such as job size, number of files, storage capacities. Finally, in Chapter 6 we draw conclusions and detail on possibilities of future work.

Chapter 3

MODELS AND PROBLEMS

3.1 Network Model and Assumptions

A Data Grid is modeled as a Graph $G = (V, E)$ where the set of vertices $V = \{1, 2, .., n\}$ represents the sites in the Grid. E is the set of weighted edges of the graph which may represent transmission bandwidth, distance between nodes, delay or loss rate. In our model the weight of an edge represents bandwidth. We assume all edges have uniform bandwidth represented by *B*. There are *m* data files $F = \{f_1, f_2, .. f_m\}$, and $S_j \in$ V is the site which contains the master copy of file f_j. A site can have master copy of multiple data files. Let $size_j$ be the size of file f_j. Let c_i be the storage capacity of each site *i* (note that for a site *i*, c_i is the storage available after storing its master files).

Data Grid users have jobs to be executed which they submit to their own sites. These jobs are executed in FIFO order within that site. As illustrated in Fig.1 below is a small example of a Data Grid with 5 sites, 10 files, and 5 jobs. The input files for each job are also shown.

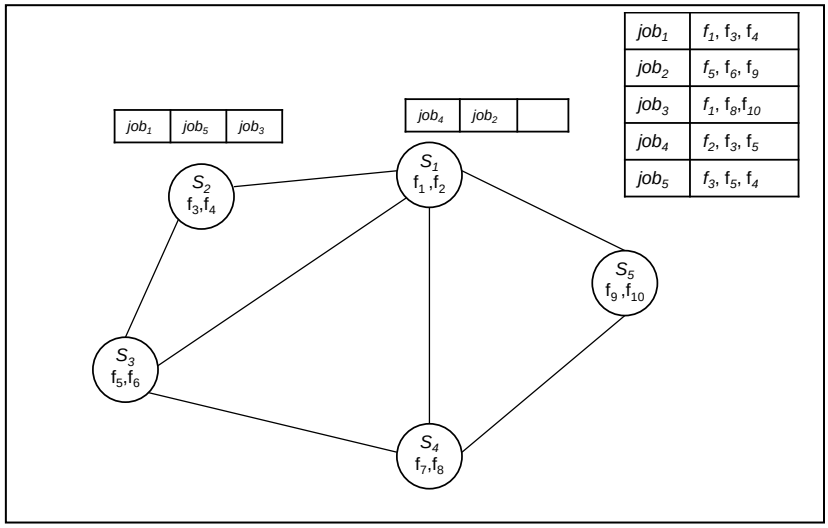

Figure 1: Data Grid Model

Let us assume there are n_i jobs submitted in site i given by the set $J_i = \{job_{i1}, job_{i2}, \ldots job_{in_i}\}$, and each job_{ik} $(1 \leq k \leq n_i)$ requires $F_{ik} \subset F$ as input files for execution. The transmission time of sending data file f_j where $f_j \in F_{ik}$ along any edge is $size_j/B$. Let t_{ij} be the number of transmissions to transmit a data file from site *i* to site *j* (which is equal to the number of edges between those sites). We do not consider the file propagation time along the edge, since it is negligible as compared to transmission time. Also in our model since we consider data intensive scenarios we assume processing time of a job to be comparatively negligible and hence ignore it. The *execution time of* job_{ik} in site *i* will be the time required to transfer all the input files from their source site to site *i*:

$$\sum_{j \in F_{ik}} t_{iS_l} \times size_l/B \text{ where } j = f_l \qquad -1$$

If we use w_{ij} to denote the number of times the site *i* needs f_j as an input file, then $w_{ij} = \sum_{k=1}^{n_i} count_k$, where $count_k = 1$ if $f_j \in F_{ik}$ and $count_k = 0$ otherwise. The *execution time of all jobs queued in site i before replication and job scheduling* is equivalent to the total transmission time spent to get all the files needed for site *i* . This will be:

$$\sum_{j=1}^{m} w_{ij} \times t_{iS_j} \times size_j/B \qquad -2$$

Let *J* denote the entire set of jobs in the Data Grid, i.e. $J = \cup_{i=1}^{n} J_i$. Let $q = |J|$ be the total number of jobs in the Data Grid. The *total job execution time before job scheduling and file replication* in the Data Grid is the sum of the job execution time of each site:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} \times t_{iS_j} \times size_j/B \qquad -3$$

The objective of our file replication and job scheduling problem is to minimize the total execution time of the Data Grid by replicating data files and scheduling independent jobs (i.e. moving the jobs from one site to another to execute) in the Data Grid. Given below, is a formal definition of the file replication and job scheduling problem.

Problem Formulation

A scheduling function is defined as $s : J \rightarrow V$, indicating that a job $i \in J$ is scheduled to node $s(i) \in V$ for execution. Assume that for scheduling, the Grid site i has a set of n_i^s jobs $J_i^s = \{job_{i1}^s, job_{i2}^s, job_{i3}^s \ldots job_{n_i^s}^s\}$ and each job job_{ik}^s $(1 \leq k \leq n_i^s)$ needs a subset F_{ik}^s of F as its input files for execution.

If we use w_{ij}^s to denote the number of times that site i needs f_j as its input file after scheduling s, then $w_{ij}^s = \sum_{k=1}^{n_i^s} count_k$, where $count_k = 1$ if $f_j \in F_{ik}^s$ and $count_k = 0$ otherwise. Since the set of input files for each job do not change before and after job is scheduled, we have $\sum_{i=1}^{n} w_{ij} = \sum_{i=1}^{n} w_{ij}^s$ for any data file f_j. With the job scheduling s, without any replication, the total execution time is

$$\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij}^s \times t_{is_j} \times size_j/B \qquad -4$$

Next we study integrating replication and job scheduling to minimize the total execution time. The *data replication and scheduling problem in the Data Grid* is to select a set of sets $M = \{R_1, R_2, \ldots . R_m\}$ where $R_j \subseteq V$ is a set of Grid sites that contains a replica copy of file f_j, and to find a scheduling function s, to minimize the *total execution time* in the Data Grid:

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

under the storage capacity constraint that $|\{R_j | i \in R_j\}| \leq c_i$ for all $i \in V$, which means that each Grid site i appears in, at most, c_i sets of M. The file accessed by each site is a replica copy available in the closest site.

Note that in the above equation w_{ij}^s cannot change arbitrarily as long as it satisfies $\sum_{i=1}^{n} w_{ij} = \sum_{i=1}^{n} w_{ij}^s$ for any data file f_j. w_{ij}^s is determined by the scheduling function s.

This problem in NP-hard, since the data replication problem without scheduling is NP-hard [17].

Chapter 4

INTEGRATED SCHEDULING AND REPLICATION

4.1 Algorithms and Theoretical Analysis

In this section we propose the Job Scheduling and Data Replication algorithm and show its constant performance ratio by theoretical analysis. Before that we present the replication algorithm from [17] and call it in the Job Scheduling and Data Replication algorithm. Even though the proposed algorithm has a much lower time complexity than the optimal algorithm, it is still an exponential algorithm. Therefore we propose a series of heuristic algorithms and discuss their time complexity.

Algorithm 1: **Replication Algorithm for Scheduling *s***

begin

$M = R_1 = R_2 = R_p = \emptyset$;

while (the total access cost can still be reduced by replicating data files in Data Grid)

Among all sites with available storage capacity and all data files, let replicating data file f_i in site j gives the maximum $\tau(G, \{R_1, R_2, \dots, R_m\}, s) - \tau(G, \{R_1, R_2, \dots R_i \cup \{j\}, \dots, R_m\}, s)$;

$R_i = R_i \cup \{j\}$;

end

return $M = \{R_1, R_2, \dots R_m\}$ and τ;

end

The greedy replication algorithm takes place in rounds. In each round, a file is replicated on a site that gives maximum reduction in the total execution time.

Next, for all of the n^q scheduling outputs, we run above centralized replication algorithm, and find the replica set and job placement which give the minimum total execution time, shown in below algorithm.

Algorithm 2: **Job Scheduling and Data Replication Algorithm**

begin

for each of the n^q scheduling s

Run **Algorithm 1**;

Find the one that gives the minimum τ, denoted as τ_{min};

Output the corresponding M, s, and τ_{min};

end for

end

Below we show that τ_{min} is close to the optimal total execution time with some constant performance ratio. Before we show that Algorithm 2 yields a constant performance ratio, we first show our previous result [17] for Algorithm 1, which is shown in Theorem 1 below.

Theorem 1: For each job scheduling, 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 total job execution time yielded by Algorithm 1 is less than 20.5 times the optimal total job execution time corresponding to that job scheduling.

Now we show the performance guarantee by Algorithm 2.

Theorem 2: For each job scheduling, 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 total job execution time yielded by Algorithm 2 is less than 20.5 times the optimal total job execution time in the optimal Job Scheduling and Data Replication algorithm.

Proof: We have n^q different job scheduling; each one has a corresponding replica set from Algorithm 1. Let O be the optimal total job execution time among all the $((C_m^{\overline{c}})^n \times (n)^q)$ job scheduling and replication solutions. Let τ_x be the total job execution time for the x^{th} job placement and greedy replication algorithm (Algorithm 1). Let O_x be the optimal job execution corresponding to the x^{th} job scheduling. Without loss of generality, assume that the x^{th} job

scheduling is the job scheduling in the optimal solution, then $O_x = O$. Let τ_{min} be the minimum total job execution time yielded by Algorithm 2. Then $\tau_{min} \leq \tau_x$. From Theorem 1, we have $\tau_x < 20.5\ O_x$. We have therefore, $\tau_{min} < 20.5\ O_x = 20.5O$.

Complexity of Algorithm

We calculate the complexity of our Scheduling and Replication Algorithm. If there are *m* files to be replicated in *n* sites each site having a storage capacity of $\overline{c}$ and there are *q* jobs to be executed then the complexity of the algorithm is $O(m^2 n^{3+q\overline{c}})$. This is lower than that of the optimal which is $O\ ((C_m^{\overline{c}})^n \times (n)^q)$

4.2 Heuristic Algorithms

Since the complexity of the algorithm is still exponential, we develop a suite of heuristics which reduce the time complexity dramatically while still keeping up with the performance. We validate the performance of these heuristics with the Job Scheduling and Data Replication algorithm via simulations. We make the following assumptions for the heuristics. Jobs originate in any of the grid sites. Each job requires a set of input files. The execution time of a site is the time required to transfer all the files required by the job. Initially one copy of master file is available in one of the sites in the Grid. Based on the jobs queued in each site, the site has a certain demand for the files. Each site has an access cost for a file which is the demand for file in the site multiplied by time to transfer the file from its originating site. In the beginning all sites have plenty of storage capacity.

Algorithm 3: Replication Scheduling Heuristic

begin

Run **Algorithm 1**;

Total Execution Time = Total Execution Time After Replication

for each unmapped job do

map <job_i $site_j$> such that $site_j$ gives minimum execution time for job_i

end for

generate new schedule $s : J \rightarrow V$

return $M = \{R_1, R_2, \ldots R_m\}$ and s

end

In this heuristic we first run the centralized greedy replication strategy from Algorithm 1. After the replication is finished, the availability of multiple replicas of each data gives more scope for scheduling. In the scheduling phase, we schedule each job to a site which gives the minimum execution time for that job.

The time complexity of heuristic is much lower than Job Scheduling and Data Replication while keeping up the performance. The complexity of the greedy replication is $O(m^2n^3\overline{c})$ and scheduling is $O(nq)$. So the complexity of the heuristic is $O(m^2n^3\overline{c} + nq)$.

Algorithm 4: Scheduling Replication Heuristic

begin

for each unmapped job do

map <job_i $site_j$> such that $site_j$ gives minimum execution time for job_i

end for

generate new schedule $s : J \rightarrow V$

Run **Algorithm 1** for schedule s

return $M = \{R_1, R_2, \ldots R_m\}$ and s

end

In this algorithm we do an initial scheduling to map the jobs to sites giving minimum execution time for the job. Then based on this job placement we do the greedy replication. This algorithm performs much better than Algorithm 3 since the jobs are placed at right sites closer to its input data. Replicating based on this job site configuration further helps in reducing the total execution time.The complexity is the same for this heuristic as the Replication Scheduling heuristic, $O(m^2n^3\overline{c} + nq)$.

Algorithm 5: Integrated Scheduling and Replication Algorithm

begin

$M = R_1 = R_2 = R_p = \emptyset$;

do

Among all the files, and sites select <$file_i$ $site_j$> such that TotalExecutionTime is minimized the most.

Let $file_i$ and $site_j$ give the most reduction in TotalExecutionTime.

if (storage available($site_j$))

Replicate $file_i$ in $site_j$; $R_i = R_i \cup \{j\}$

Total Execution Time = Total Execution Time After Replication

for each jobs

select $site_j$ such that $site_j$ gives minimum execution time for job_i

map <job_i $site_j$> (note: if $site_j$ is job_i current site then job_i is not scheduled).

end for

generate a new schedule $s : J \rightarrow V$

Total Execution Time = Total Execution Time After Schedule

while (Total Execution Time can be reduced further and storage is available in any site)

return $M = \{R_1, R_2, \ldots R_m\}$ and s

end

The Integrated Replication and Scheduling algorithm is a centralized algorithm based on the greedy approach. We sort the <File, Site> pair in descending order of demand for the file in the particular site multiplied by the transfer time from the data source site. In each round of replication a <File, Site> pair is selected provided the site has sufficient storage available to replicate the file such that the replica placement minimizes the total execution time the most. In each round we run a schedule to map a <Job, Site> pair to further reduce the total execution time. The process of replication and scheduling continues till all the sites do not have sufficient

storage capacity available or the total execution time cannot be reduced further. The complexity of this algorithm is at most $O(p^2 n^4 \overline{c}\, q)$ which is much lower than that of Algorithm 2.

Chapter 5

PERFORMANCE EVALUTION

5.1 Simulation and Analysis

We evaluate the performance of our algorithms with simulations. For all the simulations we assume the following. Initially master copy of each file is stored in some site. We assume size of each file to be 1 unit for simplicity. Also the transfer time of file is calculated in terms of number of hops multiplied by file size/bandwidth. We assume uniform bandwidth among all sites. Jobs originate randomly from any site. Each job requires a set of input files. We first compare our Job Scheduling and Data Replication algorithm (Algorithm 2) with heuristics Replication Scheduling (Algorithm 3), Scheduling Replication (Algorithm 4) and Integrated Replication Scheduling (Algorithm 5).

For our comparisons of the Algorithm 2 with the heuristics, due to its high time complexity we assume a Grid with 5 nodes. We perform comparisons by varying the number of files, number of jobs and storage capacity of the nodes. Unless otherwise varied we have 5 grid sites, 10 files and 5 jobs. Initially one copy of file is placed randomly in any of the site. We assume each job has an input of 5 files (randomly chosen from the 10 files) and each site has a storage capacity of 5. For the heuristic we assume that jobs randomly originate at a site and each site has its own initial job queue. For simplicity we will further address the Job Scheduling and Data Replication with constant performance algorithm as SR-P and heuristics as R, RS, SR and IRS respectively.

Comparison of SR-P vs. Heuristics RS, SR and IRS

Vary number of Files. We first vary number of data files randomly distributed in each of the site in the grid as 5, 10, 15, and 20 while keeping the number of Grid sites 5, the number of jobs 5 and the storage capacity as 5.

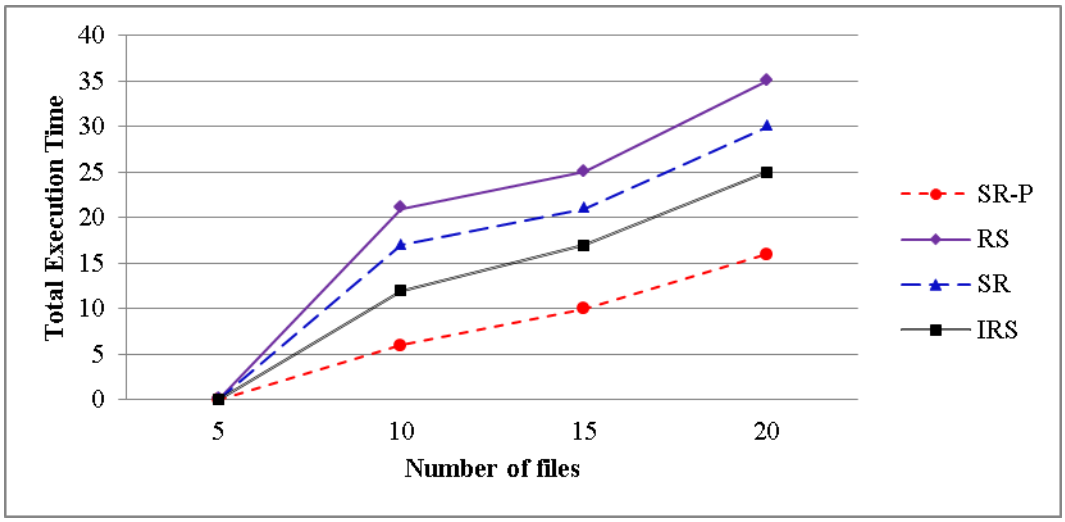

Figure 2: Vary number of files

Figure2. shows as we vary the number of files the total execution time increases. For less number of files more storage is available in the grid sites and lots of replica copies can be made. Making more files available reduces the total execution time. As the number of files increases fewer replicas are created due to storage constraint. As a result execution time is more.

Vary number of Jobs. We vary the number of jobs submitted to the sites for execution as 5, 10, 15, and 20 while keeping the number of Grid sites 5, files 10 and storage capacity 5 as constant.

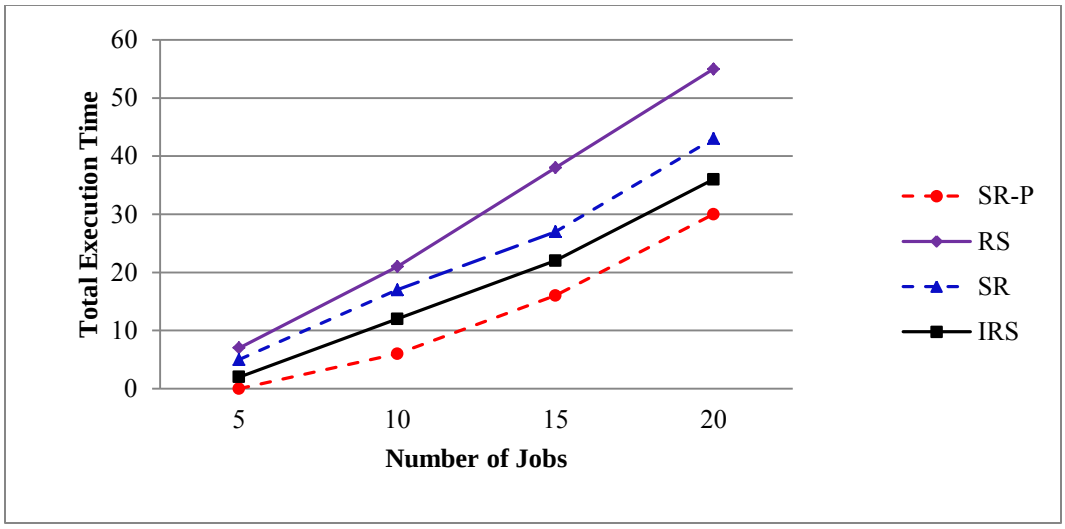

Figure 3: Vary number of jobs

As seen in Figure 3. As we vary the number of jobs the total execution time increases. This is natural since if there are more jobs the time taken to execute them will also be more.

Vary storage capacity of sites. We vary each site's storage capacity as 3, 5, 7, and 9 while keeping the number of Grid sites 5, number of jobs 5, and number of files 10 as constant.

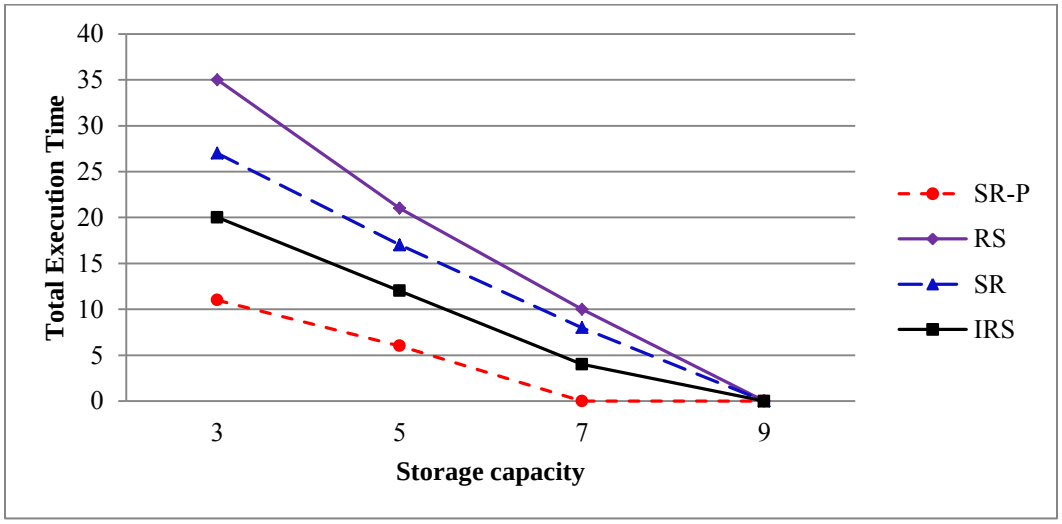

Figure 4: Vary storage capacity for Grid sites

As seen from Figure 4. when storage is more, lots of file replicas are placed in sites. This makes more data available for scheduling, hence reducing the total execution time.

Performance wise we note that JSDR gives a low total execution time as compared to the heuristics which is in accordance with the theoretical analysis of the algorithm. Among the heuristics IRS heuristic performs comparable to JSDR and gives a better result than RS and SR. SR is better than RS for all cases when we vary files, vary jobs and vary the storage capacity in each site. By scheduling we already move the jobs closer to its input data. Replication further helps by reducing the transfer time thus minimizing the total execution time. We also compare our heuristics RS, SR and IRS with Local Greedy Replication and Scheduling algorithm (LGRS), explained as below.

Local Greedy Replication and Scheduling Algorithm (LGRS) - In the local greedy replication, for each site, its most frequently accessed file is replicated locally. Then a greedy scheduling is performed.

Comparison of Heuristics R, RS, SR and IRS and LGRS

For our comparisons with the RS, SR, IRS and LGRS algorithm, we assume a Grid with 30 nodes. We perform comparisons by varying the number of files, number of jobs and storage capacity of the nodes. Unless otherwise varied we have 30 grid sites, 1500 files and 1500 jobs. We assume each job has input of 10 files and each site has a storage capacity of 10. Initially one copy of file is placed randomly in any of the site. We assume that jobs randomly originate at a site.

Based on the data patterns accessed by jobs we can classify it to two categories:

Case 1: Random Distribution

In this case input files for each job are chosen randomly from the set of available data files in the Data Grid.

Vary number of Files. We first vary number of data files in the grid as 1000, 1500, 2000, and 2500 while keeping the constant number of Grid sites as 30, number of jobs as 1500 and storage capacity as 50. Each job requires 10 input files which are initially placed randomly in the sites.

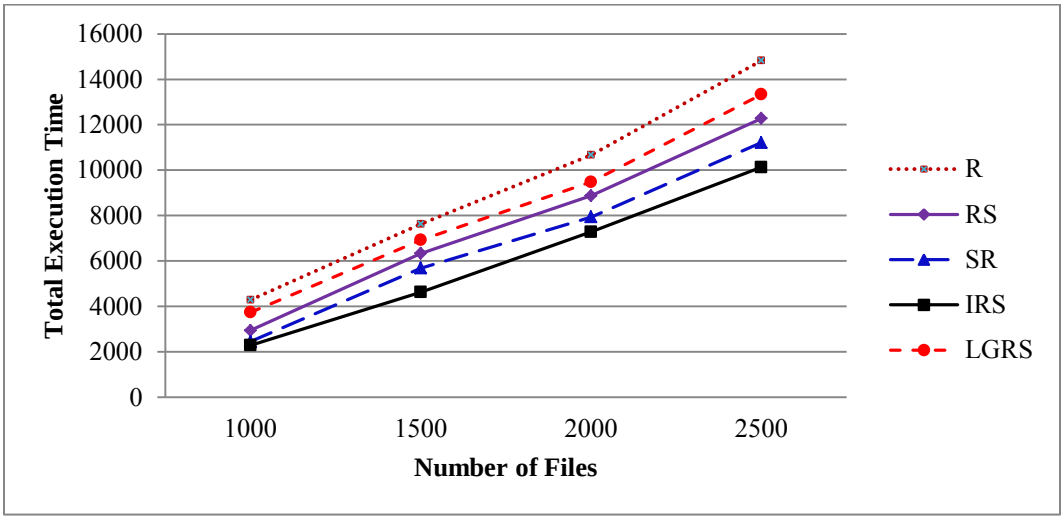

Figure 5: Vary number of files

As seen from Figure 5, IRS gives minimum execution time followed by SR and then by RS and LGRS. Clearly we can show that scheduling is important to take full advantage of replication.

Vary number of Jobs. We vary the number of jobs submitted to the sites for execution as 1000, 1500, 2000, and 2500 while keeping the constant number of Grid sites as 30, files as 1500 and storage capacity as 50. Each job requires 10 files. Each job initially originates randomly from any of the 30 sites.

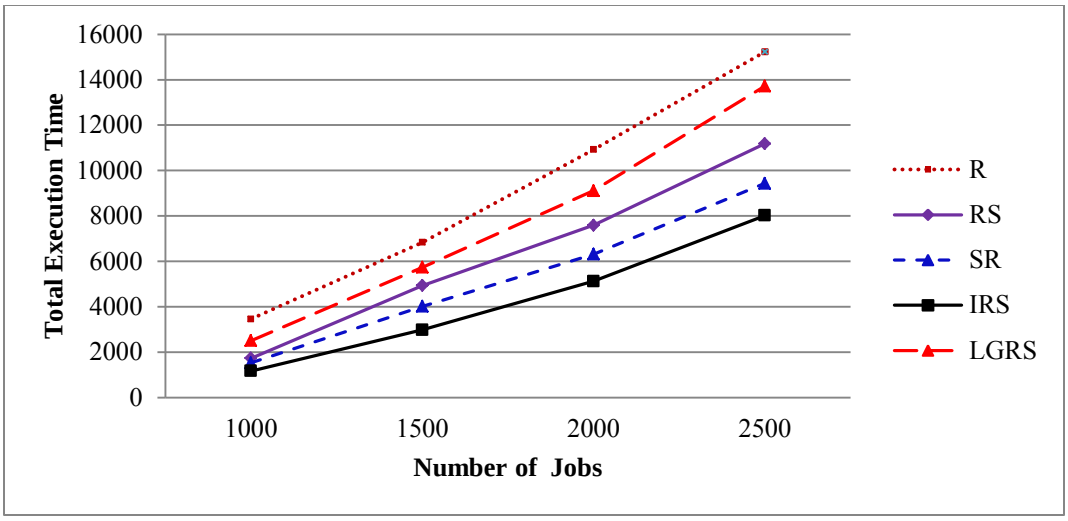

Figure 6: Vary number of jobs

When we vary the jobs too we find IRS, SR perform better that RS and LGRS.

Vary storage capacity of sites. We vary the storage capacity of each site. The parameters of Grid sites 30, number of files 1500 and number of jobs 1500 remain constant.

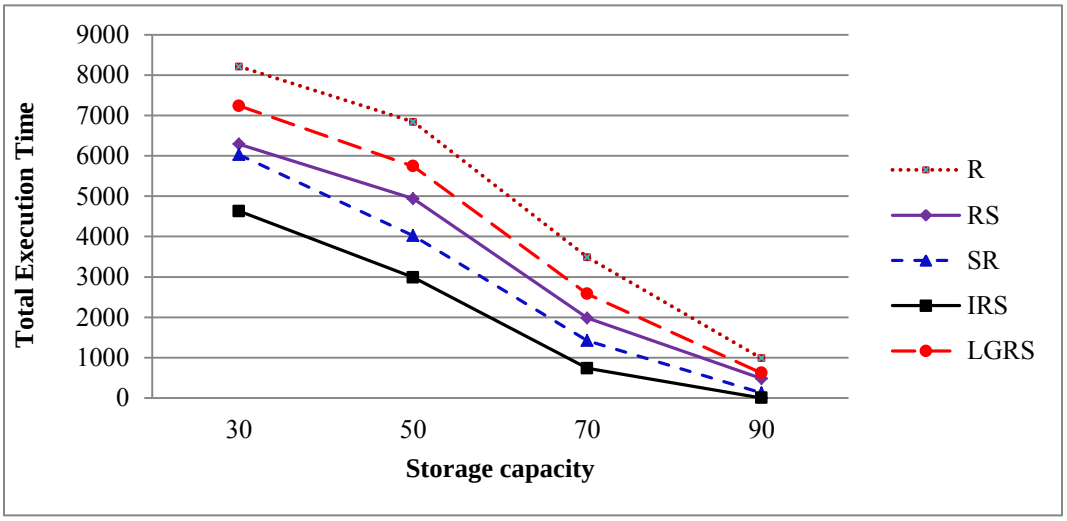

Figure 7: Vary storage capacity in each site

As seen from Figure7. when storage is more, lots of file replicas are placed in sites. This makes more data available for scheduling, hence reducing the total execution time.

We also observe for all the plots; since the scope of the replica placement in LGRS is limited locally, although it does reduce the total execution time but IRS outperforms much better than that.

Case 2: Zipf Distribution

The Zipf distribution, sometimes referred to as the zeta distribution, is a discrete distribution commonly used in linguistics, insurance, and the modeling of rare events. The input files for jobs are based in Zipf distribution. In Zipf distribution, for p data files, the probability for each job to access the jth $(1 \leq j \leq p)$ data file is represented by $P_j = \frac{1}{j^\theta \sum_{h=1}^{p} 1/h^\theta}$.

Vary number of Files. We first vary number of data files in the grid as 1000, 1500, 2000, and 2500 while keeping the constant number of Grid sites as 30, number of jobs as 1500 and storage capacity as 50.

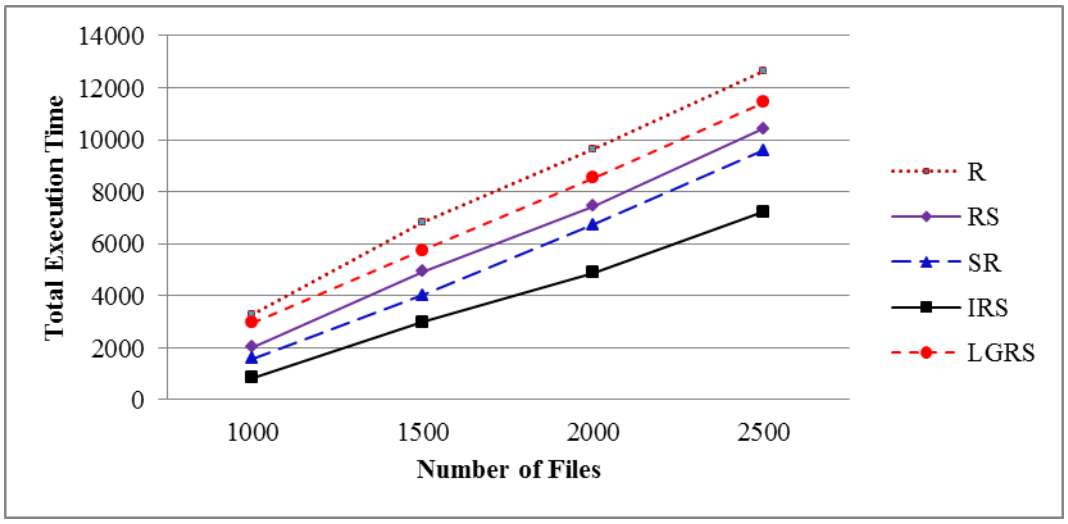

Figure 8: Vary number of files

Vary number of Jobs. We vary the number of jobs submitted to the sites for execution as 1000, 1500, 2000, and 2500 while keeping the constant number of Grid sites as 30, files as 1500 and

storage capacity as 50. Each job requires 10 files which are obtained from the Zipf distribution. Each job initially originates randomly from any of the 30 sites.

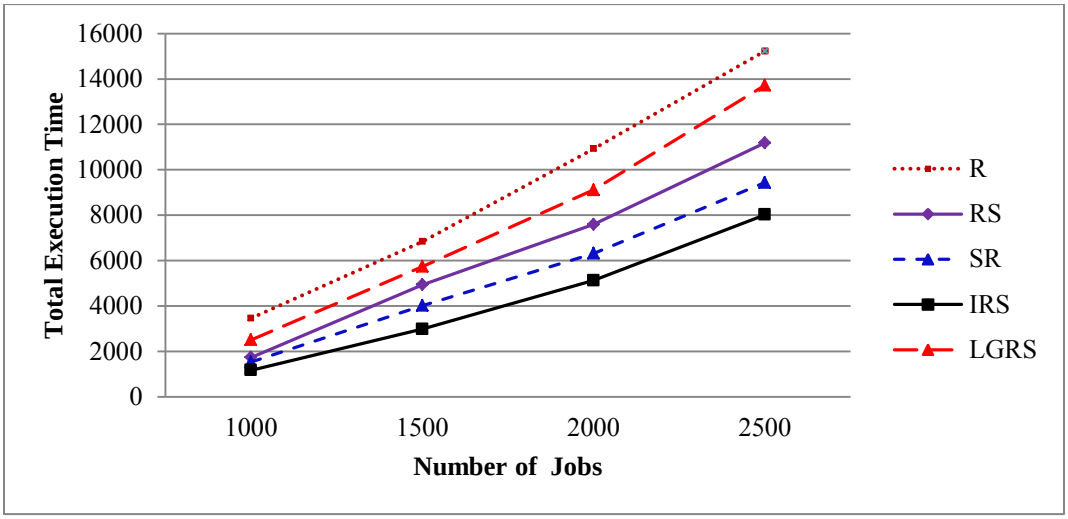

Figure 9: Vary number of jobs

Vary storage capacity of sites. We vary the storage capacity of each site. The parameters of Grid sites 30, number of files 1500 and number of jobs 1500 remain constant.

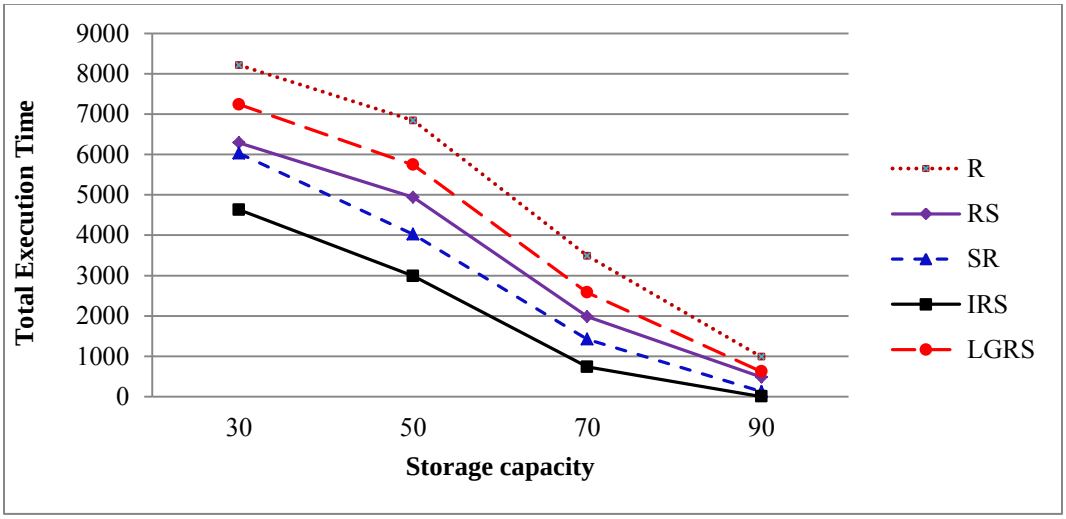

Figure 10: Vary storage capacity

The Zipf distribution to access files outperforms in comparison to random pattern to access files. The reason is similar jobs with similar files are scheduled to the same site closer to where the data files are replicated resulting in reducing the total completion time.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis we have formulated and studied a problem for integrating scheduling and replication in data intensive scientific applications. In data intensive scenarios where the data transfer time plays a very important role the right placement of data for job execution is extremely imperative. In this work we have focused on one objective of minimizing the total execution time both for placing the replicas at the right location and for mapping jobs for execution to the corresponding sites. We have a Job Scheduling and Data Replication algorithm with provable performance within a constant ratio of optimal. We also have developed a set of heuristics and validated our results with simulations and analysis.

6.2 Future Work

In future we want to find an Approximation Algorithm that achieves integration with same objective in reduced time complexity. We also want to implement the integrated algorithm in the distributed environment. This work focuses on scheduling independent jobs; the algorithm can be extended to incorporate job dependencies and workflows.

REFERENCES

LIST OF REFERENCES

[1] Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. (2001). The data grid: Towards architecture for the distributed management and analysis of large scientific datasets. Journal of Network and Computer Applications.

[2] W. Hoschek, F.J Jaen-Martinez, A.Samar, H.Stockinger, K.Stockinger. (2000). Data management in an international data grid project in: Proceedings of the First IEEE/ACM International Workshop on Grid Computing'00, Springer-Verlag, Berlin-Germany, Bangalore-India.

[3] S. Park and J. Kim. (2003). Chameleon : a Resource Scheduler in a Data Grid Environment, in Proc. of the 3rd IEEE/ACM International Symposium of cluster computing and the Grid (CCGrid'03) pp 253-260, Tokyo, Japan.

[4] F. Dong and S. G. Akl. (2006) Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Technical Report 2006-504, School of Computing, Queen's University, Kingston, Ontario.

[5] Beumont, L. Carter, J. Ferrante and Y. Robert. (2002). Bandwidth centric allocation of independent task on heterogeneous platforms. In Proceedings of the International Parallel and Distributed Processing Symposium, Fort Lauderdale, FL.

[6] M. Faerman, R. W .A Su, and F. Berman. (1999). Adaptive performance prediction for distributed data-intensive applications. ACM Press, In Proceedings of the ACM/IEEE SC99 Conference on High Performance Networking and Computing, Portland, OH.

[7] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. (2000). Heuristics for scheduling parameter sweep applications in grid environments. . IEEE Computer Society Press, In Proceedings of the 9th Heterogeneous Computing Workshop, pages 349-363, Cancun, Mexico.

[8] Oliker, Leonid, Biswas, Rupak, Shan, Hongzhang, & Smith, Warren. (2004). Scheduling in Heterogeneous Grid Environments: The Effects of Data Migration. Lawrence Berkeley National Laboratory: Lawrence Berkeley National Laboratory.

[9] S. Venugopal, and R. Buyya. (2006). A Set Coverage-based Mapping Heuristic for Scheduling Distributed Data-Intensive Applications on Global Grid, IEEE CS press,Proceedings of 7th IEEE/ACM International Conference on Grid Computing, Barcelona.

[10] P. Dutot, L. Eyraud, G. Mounie, D. Trystram. (2004). Bi-Criteria Algorithm for Scheduling Jobs on Cluster Platforms. SPAA'04, Barcelona, Spain.

[11] E. Saule, P. Dutot, G. Mounir. (2008). Scheduling with Storage Constraints. Proceedings of Parallel and Distributed Processing Symposium (IPDPS 2008), Pages(s):1 -8.

[12] T. Loukopoulas, P. Lampsas, F. Dimopoulos, M.Athanasiou. (2007). Scheduling Independent Tasks in Heterogeneous Environments under Communication Constraints. Parallel and Distributed Computing, Applications and Technologies 2007. PDCAT'07.

[13] S. Venugopal, R. Buyya, and L. Winton. (2004). A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids. In 2nd International Workshop on Middleware for Grid Computing, Middleware 2004, Toronto, Ontario - Canada, ACM Press, New York, NY, USA.

[14] W.H. Bell, D.G. Cameron, R. Cavajal-Schiaffino, K. Stockinger, and F. Zini. (2003). Evaluation of economy based file replication strategy for a data grid. In Proc. Of International Workshop in Agent Based Cluster and Grid Computing.

[15] A. Chervenek, E. Deelman, M. Livny, M.-H. Su, R. Schuler, S. Bharathi, G. Mehta, and K. Vahi. (2007). Data placement for scientific applications in distributed environments. In Proceedings of Grid Conference 2007, Austin, Texas.

[16] H. Lamehamedi, B.K. Szymanski, and B.Conte. (2005). Distributed data management services for dynamic data grids.

[17] D.T Nukarapu, B. Tang, L. Wang, and S. Lu. (2010). Data Replication in Data Intensive Scientific Applications with Performance Guarantee. IEEE Transactions on Parallel and Distributed Systems.

[18] K. Ranganathan and I. Foster. (2002). Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications, in Proc. of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), pp. 352-361, Edinburgh, Scotland.

[19] A. Elghirani, R. Subrata, Albert Y. Zomaya. (2007). Intelligent Scheduling and Replication in Datagrids: a Synergistic Approach, CCGRID, pp.179-182, Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07).

[20] R.S. Chang, J.S. Chang, S.Y. Lin. (2007). Job scheduling and data replication on data grids, Future Generation Computer Systems 23 (7) 846-860.

[21] F. Desprez and A. Vernois. (2006). Simultaneous scheduling of replication and computation for data-intensive applications on the grid. Technical Report RR2005-01, Lyon, France.

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

[23] W.H. Bell, D.G. Cameron, R. Cavajal-Schiaffino, K. Stockinger, and F. Zini. (2003). Analysis of scheduling and replica optimization strategies for data grids using optorsim.

[24] N.N Dang and S.B. Lim. (2007). Combination of replication and scheduling in data grids. International Journal of Computer Science and Network Security, 7(3):304-308.

[25] G. Khanna, N. Vydyanathan, U. V.Catalyurek, T. M. Kurc, S. Krishnamoorthy, P. Sadayappan, J. H. Saltz,. (2006). Task Scheduling and File Replication for Data-Intensive Jobs with Batch-shared IO, Proceedings of the 15th IEEE International Symposium on High-Performance Distributed Computing (HPDC-15) pp. 241-252.