DATA REPLICATION IN DATA INTENSIVE SCIENTIFIC APPLICATIONS WITH PERFORMANCE GUARANTEE

A Thesis by

Dharma Teja Nukarapu

Bachelor of Technology, DVRCET, India, 2007

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

December 2009

DATA REPLICATION IN DATA INTENSIVE SCIENTIFIC APPLICATIONS WITH PERFORMANCE GUARANTEE

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

_____

John Watkins, Committee Member

_____

Krishna K. Krishnan, Committee Member

DEDICATION

This thesis is dedicated to my parents and my brothers

ACKNOWLEDGEMENTS

I would like to express my heartfelt and sincere thanks to my advisor, Dr. Bin Tang, for his guidance, remarkable patience, support, encouragement and valuable suggestions throughout my thesis at Wichita State University.

I would also like to thank Dr. John Watkins and Dr. Krishna K. Krishnan for their time spent in reviewing this thesis work.

I'm also grateful to my advisor Keenan Jackson for helping me in my course work.

I would like to thank Dr. Liqiang Wang and Dr. Shiyong Lu for spending their valuable time in discussing important issues and helped me moving in the right direction.

I appreciate all my friends for their continued support, thanks to Rohini Kurkal, Archana Sridharan, Aashish Devalla, Shyam S Alugubelly, Prudhvi Danda and Murali Nallamothu for supporting me morally through the years I have known them.

Finally, I am very grateful to my family members for their love and support.

ABSTRACT

Data replication is well adopted in data intensive scientific applications to reduce the data file transfer time and the bandwidth consumption. However, the problem of data replication in Data Grids, an enabling technology for data intensive applications, is proved to be NP-hard and even non-approximable. Previous research in this field are either theoretical investigations without practical consideration, or heuristics-based with little or no theoretical background. In this paper, we propose a data replication algorithm which not only has provable theoretical performance guarantee, but also can be implemented in a distributed and practical manner. Specifically, we design a replication technique which reduces the total job execution time at least half of that obtained from the optimal solution. Our centralized replication algorithm is amenable to distributed implementation, which can be easily adopted in a distributed environment such as the Data Grid. We have done extensive simulations to validate the proposed replication algorithms. Using our own simulator, we show that the centralized greedy replication algorithm performs comparably to the optimal algorithm under different network parameters. Using GridSim, a popular distributed Grid simulator, we demonstrate that the distributed replication technique significantly outperforms an existing replication technique; moreover, it is more adaptive to the dynamic change of file access pattern in Data Grids.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

Data intensive scientific applications, which mainly answer some of the most fundamental questions faced by human being, are becoming increasingly prevalent in the domain of scientific and engineering research. These applications consume and produce huge amount of data (in the order of terabytes and petabytes). The data used for these applications, is spread on a very large geographical area. These applications are supported by storage units, processing units, connected together in a network [37, 38, 39]. Examples include human genome mapping [28], high energy particle physics and astronomy [1, 19], and climate change modeling [23]. In such applications, a very large amount of data sets are generated and accessed by scientists world-wide.

The Data Grid [5, 16] is an enabling technology for data intensive applications. It is composed of hundreds of geographically distributed computation, storage, and networking resources to facilitate the data sharing and management in data intensive applications [37, 38, 39]. One distinct feature of the Data Grid is that it produces and a manages very large amount of data sets, in the order of terabytes and petabytes. For example, the Large Hadron Collider (LHC) at CERN near Geneva is the largest scientific instrument on the planet. Since it began operation in August 2008, it was expected to produce roughly 15 petabytes of data annually, which are accessed and analyzed by thousands of scientists around the world [3].

Replication is an effective mechanism to reduce the file transfer time and bandwidth [48] consumption in Data Grids - placing most accessed data at the right location can greatly improve the performance of data access from user's perspective. Meanwhile, even though the memory

and storage capacity of modern computers are ever increasing, they are still not keeping up with the demand of storing a gigantic amount of data produced in the scientific applications.

With each Grid site having limited memory/storage space, the data replication becomes more challenging. However, previous works of data replication under limited storage in Data Grids mainly occupy two extremes of a wide spectrum - they are either theoretical investigations without practical consideration, or heuristics-based experiments without performance guarantees. Please read Section 2 for a comprehensive literature review. In this thesis, we endeavor to bridge the above gap by proposing an algorithm which has provable performance guarantee and can be implemented in a distributed manner.

We study how to replicate the data files onto the Grid sites with limited storage space in order to minimize the overall job execution time. In our model, the scientific data, in the form of files, are produced on some Grid sites as the result of the scientific experiments, simulations, or computation. Each Grid site executes a sequence of scientific jobs. To execute each job, Grid site usually needs some scientific data as its input files. If these input files are not in the local storage resource of the Grid site, they will be accessed and transferred from other sites. Each Grid node can cache multiple data files subject to its storage/memory capacity limitation. The objective of our file replication problem is to minimize the overall job execution time.

Specifically, we formulate this problem as a graph-theoretical problem and design a centralized greedy algorithm which provably gives the total data files execution time reduction at least half of that obtained from optimal algorithm. We show that our centralized algorithm is amenable to distributed implementation, which can be easily adopted in a distributed environment such as the Data Grid. The core idea of our algorithms is that if there are multiple replica of data files existing in the Data Grid to execute its jobs efficiently, each Grid site needs

to locally keep track of (and thus fetch the data from) the closest replica site. This can dramatically improve the Data Grid performance because transferring large-sized files takes huge amount of time and bandwidth. The central part of our distributed algorithm is a mechanism for each Grid site to accurately locate and maintain closest replica site. Our distributed algorithm can make data caching decision (i.e., replica creation and deletion) by observing the recent data access traffic going through it. Thus the caching adapts well to the dynamic change of user access behavior.

We envision that in a closely collaborative scientific environment upon which the data-intensive applications are built, each participating institution site could well be a data producer as well as a data consumer. Currently, for most Data Grid scientific applications, the massive amount of data is generated from few sites (such as CERN for high energy physics experiments) and accessed by all other scientific institutions. In future, for a large scale and closely collaborative environment, each participating institution generates data as a result of scientific experiments or simulations and meanwhile access data from others to run its own applications.

Therefore the data transfer is no longer from a few data-generating sites to all other client sites, but could take place between arbitrary pair of Grid sites. It is a great challenge in such environment to efficiently share all the widely distributed and dynamically generated data files in terms of compute resource, bandwidth usage and job execution time. Our network model reflects such vision (please refer to Section 3 for the detailed Data Grid model).

The main results and contributions of our paper are as follows:

1. We identify the limitation of the current research of data replication in Data Grids: they are either theoretical investigations without practical consideration, or heuristics-based implementations without provable performance guarantees.

2. We propose a data replication algorithm that not only has provable theoretical performance guarantee, but can also be implemented in a distributed manner as well.

3. Via simulation, we show that our proposed replication strategies perform comparably with the optimal algorithm and significantly outperform previous existing replication technique.

4. Via simulation, we show that our replication strategy adapts well to the dynamic access pattern change in Data Grids.

Chapter 2

LITERATURE SURVEY

2.1 Data Intensive Scientific Applications

Data intensive scientific applications have huge consumption and production of data to run their experiments. Several grid sites are connected together to process scientific experiments, requiring a great number of computer processing cycles or access to large amounts of data. These scientific experiments are broken down into several pieces of programs and distributed among collaborating participants connected together and then processed.

The components of data grids are distributed in a large geographical area in the form of grid sites. Each grid site has their own resource, which includes their processing units and storage units. These grid sites are connected to each other by reliable means of communication through which data transfer is guaranteed. Any grid site can get the required data from any other connected grid site and run the experiment locally, or send its job and data to a remote grid site, run the experiment, gets the result back to it. In this way, data grid is able to drive the maximum power of the network by using maximum possible resources of all the grid sites.

2.2 Example Applications

Grid computing has been applied to computationally intensive scientific, mathematical and academic problems through volunteer computing. They are also used in commercial enterprises for diverse applications such as drug discovery, economic forecasting, seismic analysis and back-office data processing in support of e-commerce and Web services. For example, the Large Hadron Collider (LHC) at CERN near Geneva is the largest scientific instrument on the planet. Since it began operation in August 2008, it was expected to produce roughly 15 petabytes of data annually, which are accessed and analyzed by thousands of

scientists around the world [3]. Grid computing is used in the area of astronomy for digital sky surveys, area of gravity wave searches, and also used in time-dependent 3-D systems (simulation & data) for earth observation, climate modeling, earthquake modeling, fluids, aerodynamic design and dispersal of pollutants in atmosphere.

There are few scientific institutes which depend on donated CPU cycles from personal computers of people around the world. These experiments eventually require huge amount of CPU cycles to analyze billions of molecules to fight cancer. Oxford University's Centre for Computational Drug Discovery's project is one among these scientific institutes [41].

SETI (Search for Extraterrestrial Intelligence), is another scientific experiment which tries to monitor the signals coming from outer world to predict the existence of aliens.

2.3 Challenges

Grid computing uses every resource of all the clients throughout the world, as per the LHC application. Thus, some time a client has to get a file from far off distance or send its file to a far off distance in order to conduct the experiment and gather results. This costs a huge file transfer time and bandwidth. So, trying to reduce both the file transfer time and bandwidth is one of the important challenges. Some of the other important challenges are coordinated use of geographically distributed resources, managing the workflow across grid, data balancing policies, instantaneous capability to complete tasks, effective usage of resources, maintaining fast turnaround time for priority job, scheduling scenarios for distributed applications and many other [43].

2.4 Replication

Replication is an effective mechanism to reduce the file transfer time and bandwidth consumption in Data Grids - placing most accessed data at the right location can greatly improve

the performance of data access from user's perspective. Replication is a technique that locally stores a duplicate file in the local computer for later access, saving the travel to the main server every time it needs. Replication is purely server side action, as server will replicate the file to the client side computer.

The server can replicate all the files to all the clients only when they have unlimited memory. But it is different when all the clients have limited memory, when the memory of the clients are filled up with the files they will not be able to store any files. Meanwhile, even though the memory and storage capacity of modern computers are ever increasing, they are still not keeping up with the demand of storing a gigantic amount of data produced in the scientific applications.

2.5 Related Work

Replication has been an active research topic for many years in World Wide Web [25], peer-to-peer networks [4], ad hoc and sensor networking [18, 31], and mesh networks [20]. In Data Grids, the gigantic scientific data and complex scientific applications call for new replication algorithms, which have attracted lots of research recently. The most closely related work to ours is by Cibej, Slivnik, and Robic [34]. The authors study data replication on Data Grids as a static optimization problem. They show that this problem is NP-hard and non-approximable, which means there is no polynomial algorithm that provides an approximation solution if $P \neq NP$. The authors discuss two solutions: integer programming and simplifications. They only consider static data replication for the purpose of formal analysis. The limitation of the static approach is that the replication cannot adjust to the dynamically changing user access pattern. Furthermore, their centralized integer programming technique cannot be easily implemented in a distributed Data Grid. Some economical model based replica schemes are

proposed. The authors in [8, 6] use an auction protocol to make the replication decision and to trigger long-term optimization by using file access pattern. You et. al. [36] proposes utility-based replication strategies. Lei et al. [22] and Schintke et al. [29] address the data replication for availability in the face of unreliable components, which is different from our work. Tang et al. [32] present dynamic replication algorithm for multi-tier Data Grids. They propose two dynamic replica algorithms {Single Bottom Up and Aggregate Bottom Up}. Performance results show both algorithms reduce the average response time of data access compared to a static replication strategy in a multi-tier Data Grid. A dynamic data replication mechanism called Latest Access Largest Weight (LALW) is been proposed by Chang and Chang [10]. LALW associates a different weight to each historical data access record - a more recent data access has a larger weight. LALW gives a more precise metric to find a popular file for replication. Park et al. [24] propose a dynamic replica replication strategy, called BHR, which benefits from network-level locality" to reduce data execution time by avoiding networking congestion in a Data Grid. Lamehamedi et al. [21] propose a lightweight Data Grid middleware at the core of which are some dynamic data and replica location and placement techniques. Ranganathan and Foster [27] present six different replication strategies: Best Client, No Replication or Caching, Plain caching, Cascading Replication, Caching plus Cascading Replication, and Fast Spread. All above strategies are evaluated with three user access patterns (Random Access, Temporal Locality, and Geographical plus Temporal Locality). Via the simulation, the authors find Fast Spread performs the best under Random Access, and Cascading would work better than others under Geographical and Temporal Locality. Due to its wide popularity in the literature and its simplicity to implement, we compare our distributed replication algorithm to this work. System-wise, there are some real systems utilizing data replication. One is the Data Replication Service

(DRS) designed by Chervenak et al. [13]. DRS is a higher-level data management service for scientific collaborations in Grid environments. It replicates a specified set of files onto a storage system and registers the new files in the replica catalog of the site. The other is Physics Experimental Data Export (PheDEx) system [15]. PheDEx supports both the hierarchical distribution and subscription-based transfer of data. To execute the submitted jobs, each Grid site either gets the needed input data files to its local compute resource, or schedules the job close to the sites where the needed input data files are stored, or transfers both the data and job to a third site that performs the computation and returns the result. In this paper, we focus on the first approach. We leave the job scheduling problem [35], which studies how to map the jobs into Grid resources for execution, and its coupling with data replication, as our future research. There has been very active research studying the relationship between these two [9, 26, 12, 17, 11, 33, 14, 7]. The focus of our paper, however, is data replication strategy with provable performance guarantee. As demonstrated by the experiments done by Chervenak et al. [13], the time to execute a scientific job is mainly the time to transfer the needed input files from server sites to local site. In our work, we assume the job execution time only includes the input file transfer time. A similar assumption has been made in [22, 29, 7]. Since the data are read only for most Data Grid applications [26], we do not consider consistency maintenance between the master file and the replica files.

Chapter 3

PROPOSED MODEL

A Data Grid consists of a set of sites. There is one *top level site*. Other sites are called *institutional sites*, which correspond to different scientific institutions participating in the scientific project. The top level site is the centralized management entity in the whole Data Grid environment and its major role is to manage the *Replica Catalogue* (RC). RC provides information about each data file and its replicas in the Data Grid and it is essentially a mapping between each data file and all the institutional sites where the data is replicated. Each site (top level site or institutional site) may contain multiple grid resources. A grid resource could be either a computation resource, which allows users to submit and execute jobs, or a storage resource, which allows the users to store data files. We assume that within each site, the bandwidth is high enough that the communication delay inside the site is negligible. We do not further differentiate individual computation and storage resource in each site and assume each site has both computation and storage capacities.

For the data file replication problem addressed in this article, there are multiple data file and each data file is produced by its *source site* (a Grid node may act as a source site for more than one data files). Each Grid node has limited storage capacity and can cache multiple data files subject to its storage capacity limitation.

3.1 Data Grid Model

A Data Grid can be represented as an undirectional graph G (V, E) where the set of vertices $V = \{1, 2, \ldots, |V|\}$ represents the sites in the Grid, and E is the set of weighted edges in the graph. The edge weight may represent a link metric indicating loss rate, delay or transmission power. There are p data files $D = \{D_1, D_2, ..., D_p\}$ originally produced in the Data Grid. D_j is

originally produced and stored in site $S_j \in V$. Note a site can be the original producer of multiple data files. Each site *i* has a storage capacity of m_i. *For clarity of presentation,* we assume all the data files have uniform-size (occupying unit memory). But our algorithm and analysis can be easily generalized into the non-uniform case. The users of the Data Grid submit jobs to its own site and the jobs are executed in the FIFO (First In First Out) order. Let's assume the Grid site *i* has n_i submitted jobs $\{ t_{i1}, t_{i2}..., t_{ini}\}$, and each job t_{ik} needs a subset f_{ik} of *D* as its input files for execution. If we use a_{ij} to denote the number of times site '*i*' requests the file D_j, then $a_{ij} = \sum_{k=1}^{n_i} c_k$ where $c_k = 1$ if $D_j \in F_{ik}$ and $c_k = 0$ otherwise. We use d_{ij} to denote the shortest distance between site *i* and *j*. We assume each site always goes to the nearest site to get file replica. The *total job execution time* in Data Grid before replication is:

$$\sum_{i \in V} \sum_{1 \le j \le p} a_{ij} \times d_{is_j}$$

The objective of our file replication problem is to minimize the total job execution time by replicating data files in the Data Grid. Below, we give a formal definition of the file replication problem addressed in this thesis.

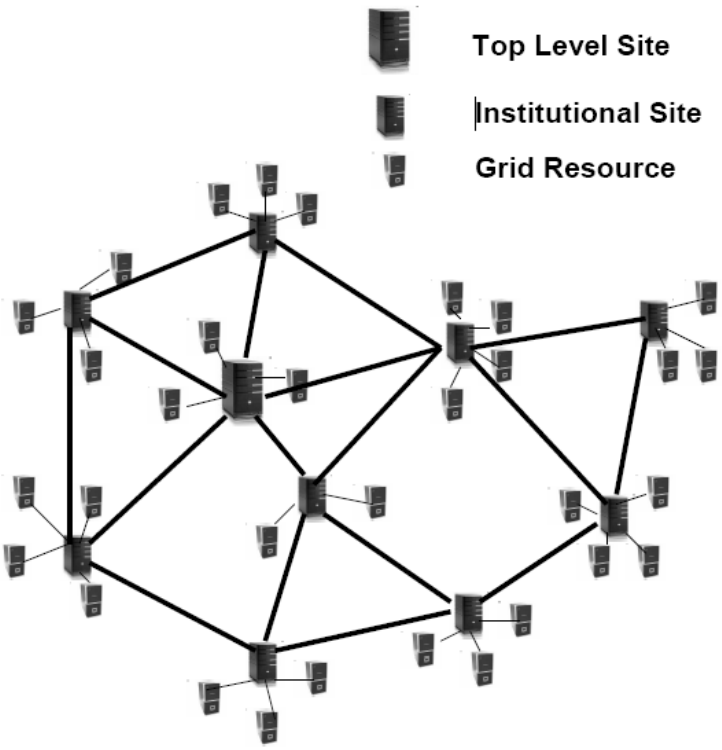

Figure 1: Data Grid Model

3.2 PROBLEM FORMATION

The *data replication problem in Data Grid* is to select a *set of sets* $M = \{M_1, M_2, \ldots M_p\}$, where M_j *subset of* V is a set of Grid nodes that store a replica copy of D_j, to minimize the *total job execution time*:

$$\tau\,(G, M) = \sum_{i \in V} \sum_{j=1}^{p} a_{ij} \times min_{l \in (\{s_j\} \cup M_j)}\, d_{il}$$

Under the storage capacity constraint

$$\left| \{M_j \mid i \in M_j\} \right| \leq m_i \qquad \text{for all} \quad i \in V$$

Which means each grid node i appears in at most m_i sets of M. Here each node accesses the data file from its closest site with a replica copy. Cibej, Slivnik, and Robic [34] have shown this problem is NP-hard [34]. Below we first present a centralized greedy algorithm with provable

performance guarantee. Then we extend and implement the greedy algorithm in a distributed manner.

3.3 Centralized Data Replication Algorithm in Data Grid

Our centralized data replication algorithm is essentially a greedy approach. Our greedy algorithm starts with all Grid nodes having all empty storage space (except the sites which originally store some files), and then, iteratively places data files into storage space of some site so that maximizing the reduction in total job execution time in a greedy manner at each step. Thus, at each step, the algorithm picks a data file D_j to place into an empty memory page r of a Grid node such that the execution time reduction of caching D_j at r is the maximum among all possible choices of D_j and r at that step. The algorithm terminates when all memory pages have been cached with data files, or the total execution time can not be reduced further. Below is the algorithm.

Algorithm: Centralized greedy algorithm

BEGIN

$M = M_1 = M_2 = \ldots = M_p = \Phi$(empty set);

While (the total job execution time can still be reduced by replicating data files in Data Grid)

Among all the sites with available storage capacity and all the data files, let site m and data file D_i be the ones which give the maximum $\tau(G, M) - \tau(G, \{ M_1, M_2, \ldots, M_i \cup \{m\}, \ldots\ldots, M_p\})$;

$M_i = M_i \cup \{m\}$;

end while;

RETURN $M = \{M_1, M_2, \ldots\ldots M_p\}$;

END.

The total running time of the Greedy Algorithm is $O(p^2 |V|^3 \bar{m})$, where $|V|$ is the number of sites in the Data Grid, $\bar{m}$ is the average number of memory pages in a site, and p is the total number of data files. Note that the number of iterations in the above algorithm is bounded by $|V| \bar{m}$, and at each stage we need to compute at most $p|V|$ benefit values where each benefit value computation may take $O(p|V|)$ time. Below we present a theorem showing that the Greedy Algorithm returns a solution with near-optimal total execution time reduction. The proof technique used here is similar to that used in [31] for the closely related problem of data caching in an ad hoc network.

Theorem 1: Under the Data Grid model we propose, the centralized data replication algorithm delivers a solution whose total access cost reduction is at least half of the optimal total access cost reduction.

3.4 Distributed Data Replication Algorithm in Data Grid

In this section, we modify the centralized greedy algorithm and extend it into a localized distributed algorithm, where each Grid Site observes the local network traffic to make intelligent caching decision. The advantage of distributed greedy algorithm is that it adapts to dynamic access traffic in Data Grid. The algorithm composes of two important components – nearest replica catalogue (NRC) maintained by each site and localized data caching algorithm running on each site. Besides the top level site maintains a *centralized replica catalogue*, which is essentially a list of *cache list* C_j for each data file D_j. The cache list C_j contains the set of sites (including S_j) that have cached D_j.

Nearest Replica Catalogue (NRC) and Its Initialization.

For each Grid node, we maintain the *nearest* node with a replica of data file D_j in the Grid. More specifically, each node i in the Grid maintains a NRC, where an entry in the NRC is

of the form (D_j, N_j) where N_j is the closest node that has a copy of D_j. At the initialization stage, the source sites send messages to the top level site about their original data files. The replica catalogue initially records each data file and its source site. The top level site then broadcasts the replica catalogue to the whole Data Grid and each grid node initializes its NRC to the source site of each data file. Note that if i is the source site of D_j or has cached D_j, then N_j is interpreted as *second-nearest replica*, i.e., the closest node (other than i itself) that has a copy of D_j. The second-nearest replica information is helpful when node i decide to remove the cached file D_j. The above information is in addition to any information (such as routing tables) maintained by the underlying routing protocol in Data Grid. The nearest-cache tables are maintained as follows.

When a node i caches a data file D_j, N_j is no longer the nearest-cache node, but the second- nearest replica. In addition, the node i sends a message to the top level site informing it is a cache node of D_j. When receiving the message, the top level site updates its replica catalogue by adding node i to data file D_j's cache list. Then it broadcasts the AddCache message to the whole Data Grid containing the tuple (i, $D_{j,}$) signifying the ID of the new cache node and the ID of the newly cached data file. Consider a node l that receives the AddCache message (i, $D_{j,}$). Let (D_j, N_j) be the nearest-replica catalogue entry at node l signifying that N_j is the cache node currently closest to l that has a copy of the data file D_j. If $d_{li} < d_{lNj}$, then the NRC entry ($D_{j,}$ N_j) is updated to (D_j, i).

When a node i removes a data file D_j from its local cache, its second-nearest cache N_j becomes its nearest cache. In addition, the node i sends a message to the top level site informing it is no longer a cache node of D_j. The top level site updates its replica catalogue by deleting i from D_j 's cache list. And then, it broadcasts a DeleteCache message with the information (i,D_j , C_j) to the whole network, where C_j is the cache list for D_j. Consider a node l that receives the

DeleteCache message and let (D_j, N_j) be its NRC entry. If $N_j = i$, then the node l updates its nearest-cache entry using C_j (with the help of routing table).

Localized Data Caching Algorithm.

Since each site has limited storage capacity, a good data caching algorithm running distributedly on each site is needed. To do this, each node observes data access traffic locally for a sufficiently long time. The *local access traffic* observed by a node *i* includes its own local data requests, non-local data requests to data files cached at *i*, and the data request traffic that the node *i* is forwarding to other nodes in the network. Before we present the data caching algorithm, we give the following two definitions.

- **Benefit of caching a data file.** Benefit of caching a data item is the job reduction in execution time given by: access freq observed in local access traffic distance to the nearest cache node.
- **Cost of deleting a data file.** Cost of deleting a data item is the increase in job execution time given by: access freq observed in local access traffic distance to the second-nearest cache node.

For each data file D_j *not* cached at node *i*, the node *i* calculates the benefit gained (or reduction in execution time) by caching the file D_j, while for each data file D_j cached at node *i*, the node *I* computes the cost (or increase in execution time) of deleting the file. With the help of NRC, each node can compute the benefit (or the cost) of caching (or deleting) data files in a localized manner using only local information. Thus our algorithm is adaptive - each node makes data caching or deletion decision by observing the most recent data access traffic in the Data Grid.

Cache Replacement Policy. With above knowledge of benefit (or the cost) of caching (or deleting) data files, a node always tries to cache the most beneficial data files that can fit in its

local memory. When the local cache memory of a node is full, the following cache replacement policy is used. Let $|D|$ denote the size of a data file (or a set of data files) D. If the local benefit of a newly available data file D_j is higher than the total local benefit of some set D of cached data files where $|D| > |D_j|$ then the set D is replaced by D_j .

Chapter 4

PERFORMANCE EVALUATION

Via simulations, we demonstrate the performance of our designed data replication algorithms over a randomly generated network topology. We first compare the centralized optimal algorithm and the centralized greedy algorithm. Then, we evaluate our designed distributed algorithm with respect to an existing replication algorithm.

4.1 Greedy versus Optimal

For a Data Grid with *n* sites and *p* data files, and each site has memory capacity *m*, the optimal algorithm is to enumerate all the possible file replication configurations in the Data Grid and then find the one which gives the minimum total execution time. Obviously, there are $(C_p^m)^n$ such file configurations, where C_p^m is the number of ways selecting *m* files out of *p* files. Due to the high time complexity of the above algorithm, we create random Data Grids with small number of sites (ranging from 5 to 25 sites) and the average distances between two sites are 3 to 6 hops.

For both greedy and optimal algorithm, all *p* files are randomly allocated in some source nodes initially, and for simplicity we assume these initial files do not occupy the storage capacities of each source node. Each data point in the plots below is an average over five different initial locations of the *p* files. In all plots, we show error bars indicating the 95 percent confidence interval.

Varying Number of Data Files. In Figure-2, we first vary the number of data files in the Data Grid while keeping the storage capacity of each site and the Data Grid size the same. Due to the high time complexity of the above algorithm, we consider a small Data Grid and choose $n = 10$, $m = 3$, and vary *p* from 5, 10, 15, to 20. We observe that the total job execution time increases

with the number of data items as expected. Also, as expected, we see that optimal algorithm performs slightly better since it exhausts all the replication possibilities, but our greedy performs quite close to optimal. It also shows that when memory size of each site is 5 units and the total number of files if 5 (note each file occupies 1 unit of memory), our greedy algorithm will eventually replicate all the 5 files into each Grid site, resulting in 0 total execution time as the optimal algorithm.

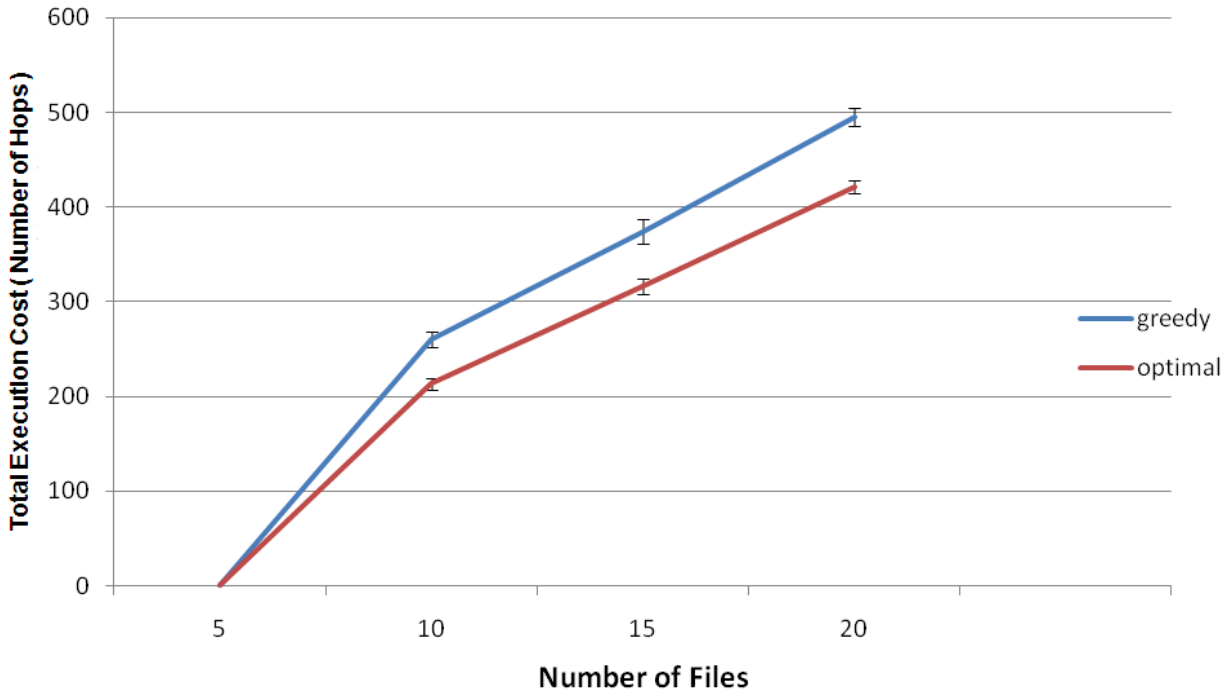

Figure -2 **Varying Number of Data Files**

Varying Storage Capacity. Figure-3 shows the result of varying the storage capacity of each site while keeping the number of data files and number of sites in the grid the same. Due to the high time complexity of the above algorithm, we consider a small Data Grid and choose n = 10, p = 10, and vary m from 1, 3, 5, 7, and 9. It's obvious that with the increase of memory capacity of each Grid site, the total execution time decreases since more file replicas are placed into the Data Grid.

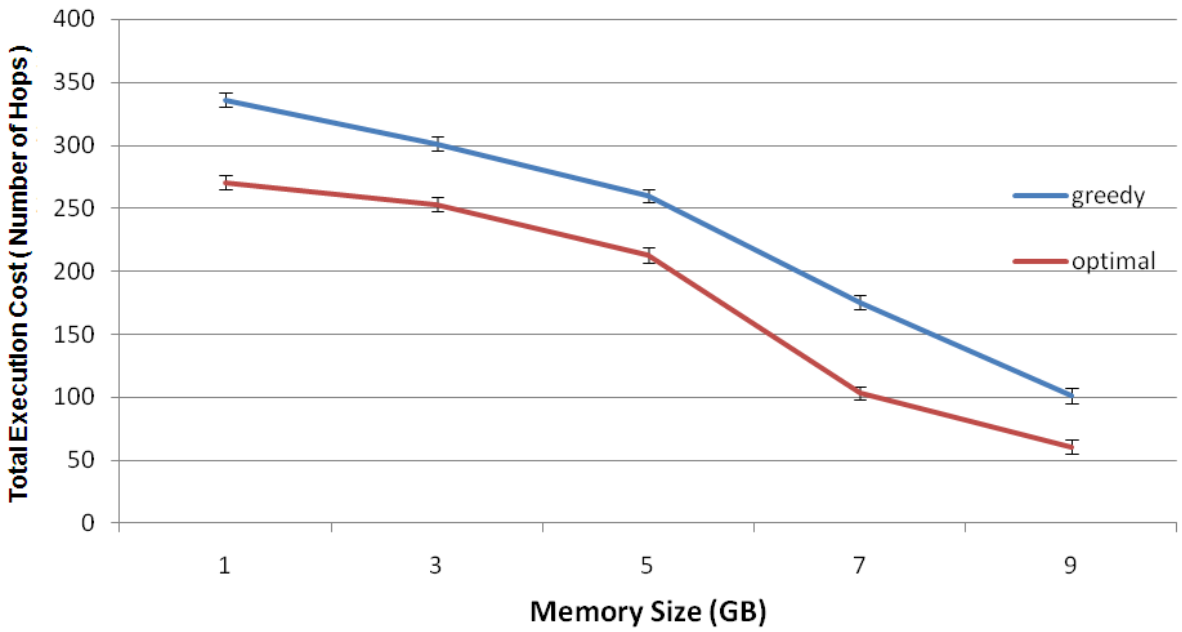

Figure -3 **Varying Storage Capacity**

Varying number of Grid Sites. Figure-4 shows the result of varying the number of sites in Data Grid. We consider a small Data Grid and choose $n = 10$, $m = 5$, and vary p from 5, 10, 15, 20, to 25. Again, the optimal algorithm perform only slightly better that the greedy algorithm.

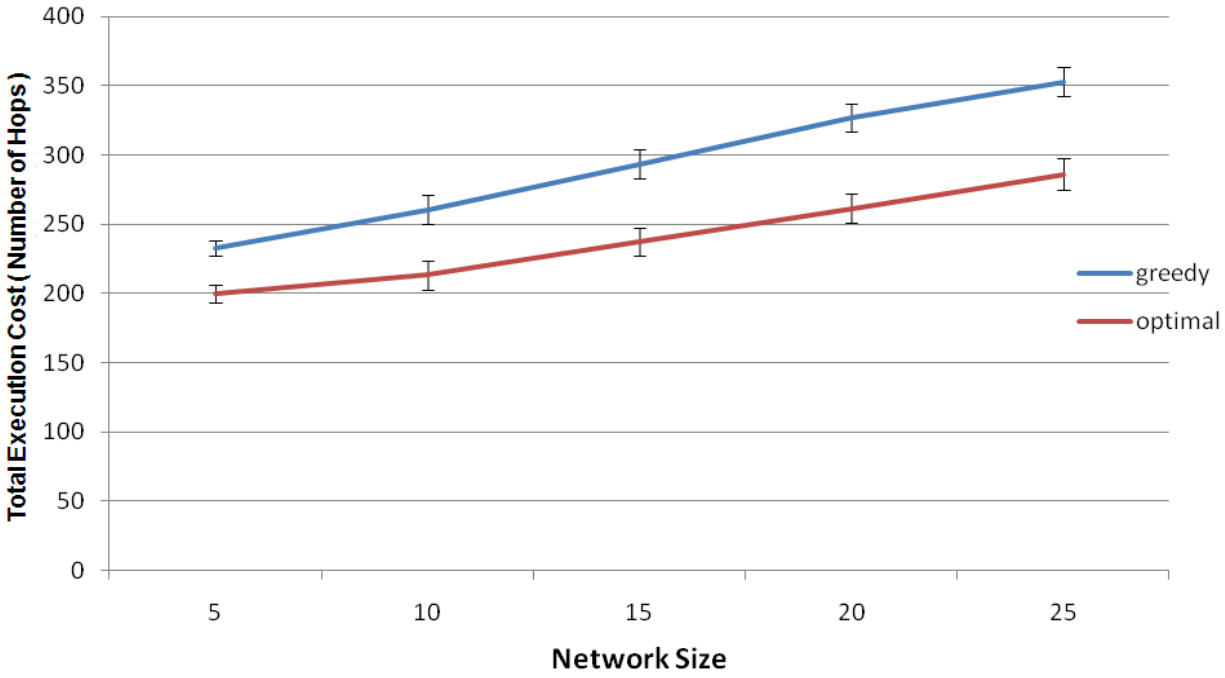

Figure - 4 **Varying number of Grid Sites**

4.2 Distributed Algorithm versus Replication Strategies by Ranganathan and Foster

We first present in some detail the replication strategies proposed by Ranganathan and Foster [27]. We then present our simulation environment and discuss simulation results.

4.2.1 Replication Strategies by Ranganathan and Foster

Ranganathan and Foster present six different replication strategies: (1) No Replication or Caching: none of the files are either replicated or cached, (2) Best Client: a file is replicated to the client having the highest access frequency for the file, (3) Cascading Replication: cascading is just like a water flow model. If the access frequency of any file exceeds the threshold then the file is replicated to the next best client in the next level, (4) Plain Caching: any client requesting the file will cache the file locally, (5) Caching plus Cascading Replication: any client requesting any file will cache locally and also the file will be replicated to the next level once exceeded the threshold, and (6) Fast Spread: replicas of the file are created at each node along its path to the

client. All above strategies are evaluated with three user access patterns: (1) Random Access pattern: No particular pattern of access exists, (2) Temporal Locality: data will have a small degree of temporal access pattern, (3) Geographical plus Temporal Locality: data will have a small degree of both geographical and temporal patterns. The simulation results show that Fast Spread performs best under Random Access, and Caching plus Cascading would work better than others under Geographical plus Temporal Locality. In our work, we compare our strategy with Caching plus Cascading under Geographical and Temporal Locality (Temporal Locality is achieved by zipf access pattern explained later).

They study the data replication in a hierarchical Data Grid model. The hierarchical model has been used in LHCGrid project [2], which serves the scientific collaborations performing experiments at the Large Hadron Collider (LHC) in CERN. In this model, there is a tier0 node at CERN and there are regional nodes which are from different institutes in the collaboration. For comparison, we also use this hierarchical model, and assume all the data files are at one node (CERN node) initially. Like [2], we also assume only the leaf sites execute jobs and thus request data files from the Grid.

4.2.2 Simulation Setup

In our simulation we use GridSim Simulator [30] (version 4.1). This version of GridSim incorporates a Data Grid extension to model and simulate Data Grids. Besides, GridSim offers extensive support for design and evaluation of scheduling algorithms, which suites our future work wherein a more comprehensive framework of data replication and job scheduling will be studied. As a result, we chose GridSim in our simulation. For comparison, we also implemented a naive approach, wherein each site caches any passing-by data file if there is free memory space and uses LRU (least recently used) policy for replacement of caches. Table 1 shows the

simulation parameters, most of which are from Ranganathan and Foster [27] for the purpose of comparison. The simulated Data Grid has 50 sites, one of them is randomly chosen as the designated CERN site. Each site (except the CERN site) has a corresponding user, which submits a sequence of jobs to the site periodically. The CERN site originally generates and stores 1000 – 5000 different data files in Data Grid, each with size 2 GB. Each site executes 400 jobs (here each site runs the jobs concurrently with other sites). Each job has 1-10 files as input files. To execute each job, the sites checks if the input files are in its local storage. If not, it first goes to the near cache site to get the file. The available storage at each site is varying from 100 GB to 500 GB. Our simulation is run on a DELL desktop with Core 2 Duo CPU and 1 G RAM.

Table 1: Simulation Parameters

Parameters	Value
Number of sites	20
Total Number of files in grid	1000 - 5000
File size	2G
Available storage of each site	100G -500G
Number of jobs executed by each site	400
Number of input files needed by each job	1-10
Link bandwidth	100M

Below we discuss the pattern of file access. First, we assume that input data files requested by each job follows *zipf-like* distribution, with the access frequencies of the data items in

descending order of the data file's ID. More specifically, the probability of accessing (or the access frequency) the j^{th} ($1 \leq j \leq 1000$) data file is represented by $P_j = \frac{1}{j^{\theta} \sum_{h=1}^{1000} 1/h^{\theta}}$, where ($0 \leq \theta \leq 1$). When $\theta = 1$, the above distribution follows the strict zipf distribution, while for $\theta = 0$, it follows the uniform distribution. We choose θ to be 0.8 based on real web trace studies. Like [27], we also study the geographical access pattern, which says that the data access frequencies at a node depends on its geographic location in the network area such that nodes that are closely located have similar data access frequencies. In our case, the leaf sites which are close to each should have similar data file access pattern.

4.2.3 Simulation Results

We mainly used two access patterns: geographic access pattern and random access pattern. The geographic access pattern here uses the zipf-like distribution and the random access pattern is purely random without zipf-like distribution. Geographic access pattern is subdivided into static and dynamic access pattern, and we call them *Geo-Static* and *Geo-Dynamic* respectively. In dynamic access pattern, the file access pattern of each site changes in the middle of the simulation while it does not change in static access pattern. So we used the following patterns:

- Random access pattern without zipf-like distribution (Random)
- Static geographic access pattern with zipf-like distribution (Geo-Static)
- Dynamic geographic access pattern with zipf-like distribution (Geo-Dynamic)

We compared both the algorithms and show how their performances vary in all the above access patterns. In our simulations, we vary the number of files while keeping the storage capacities

constant and also vary the storage capacity of each site while keeping the number of files constant.

Varying number of Data files:

In Figure 5, 6, 7, we vary the number of data files from 1000, 2000, 3000, 4000, to 5000 files while fixing the storage capacity of all the grid sites as 300GB.

Figure 5 and Figure 6 show the performance comparison of the two algorithms in Random access pattern and Geo-Static access pattern, respectively. We have the observations. First, with the increase of the number of files, the average file execution time increases for both algorithms. However, our distributed greedy data replication algorithm yields average file execution time three to four times less than that in cascading algorithm. Second, for both our distributed greedy and cascading algorithms, their average file execution time under geo-static is smaller than that under random. This is because Geo-Static incorporates both temporal and spatial access patterns, which benefits both replication algorithms.

Figure-5: **Varying number of files in the grid under Random access pattern.**

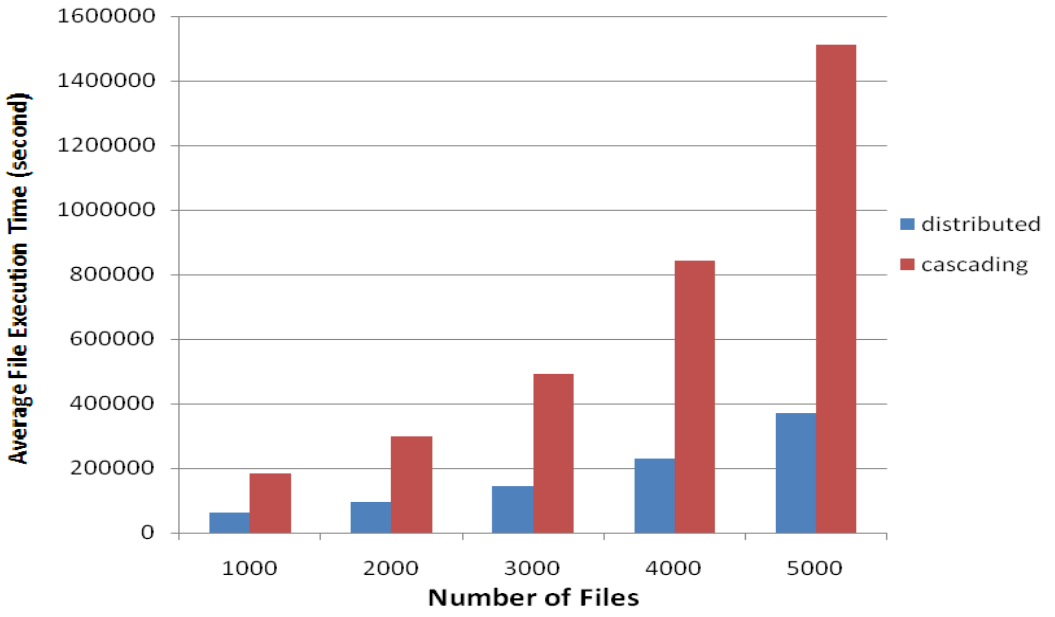

Figure-6 **Varying number of files in the grid under Geo-Static access pattern.**

Figure 7 shows the performance of both algorithms under Geo-Dynamic access pattern, wherein the file access pattern of each site changes during the simulation. Due to the change of the access pattern, both our distributed replication algorithm and cascading suffer from the file execution time increase. However, our algorithm adopts the access pattern change better than cascading. As shown in Figure 8, the percentage increase of the average file execution time of our algorithm is smaller than that of cascading, especially when the total number of files in the data grid is large (4000 and 5000 files).

Figure-7: **Varying number of files in the grid under Geo-Dynamic access pattern.**

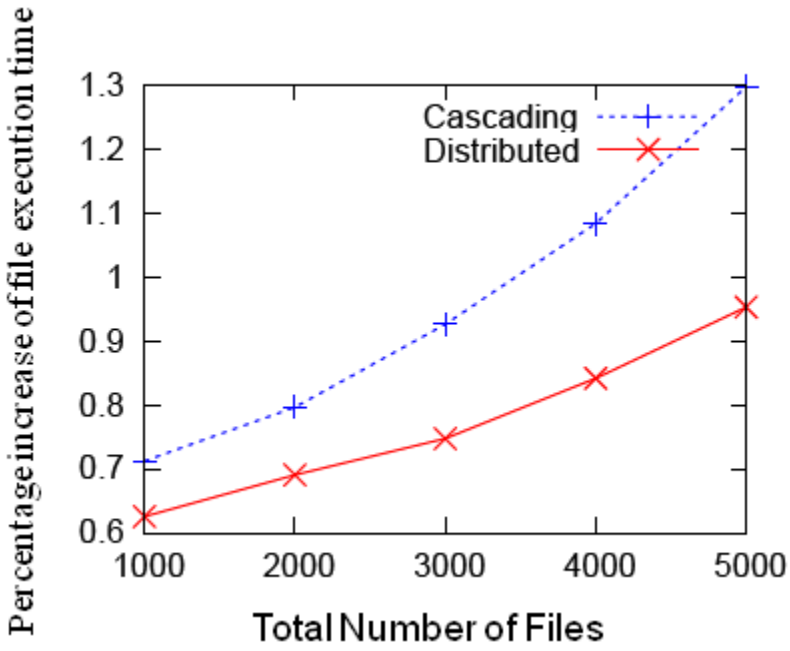

Figure-8 : **The percentage increase of average execution time due to dyanmic access pattern for fixed storage capacity.**

Varying storage capacitiy of each site:

In Figure 9, 10, 11, we vary the size of the storage capacity of the grid sites from 100, 200, 300, 400, to 500 GB while fixing the number of files in the grid as 3000. As expected, as the storage capacities increase, there is a decrease in the file execution time for both the algorithms, since bigger storage capacities allow each site to storage more replicated files which can be accessed locally.

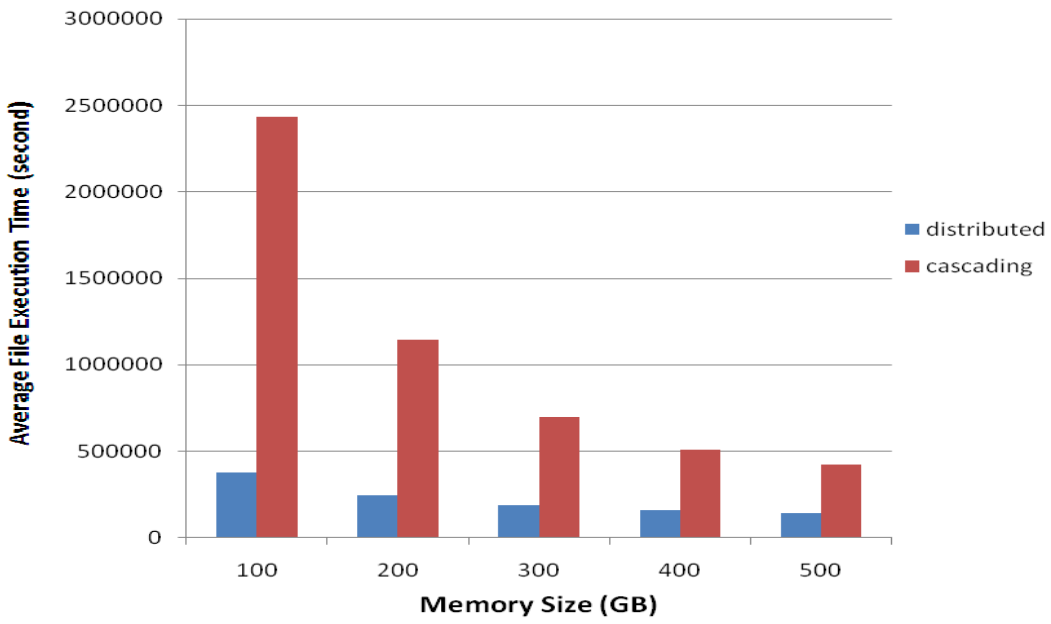

Figure -9**: Varying storage capacity in the grid under Random access pattern.**

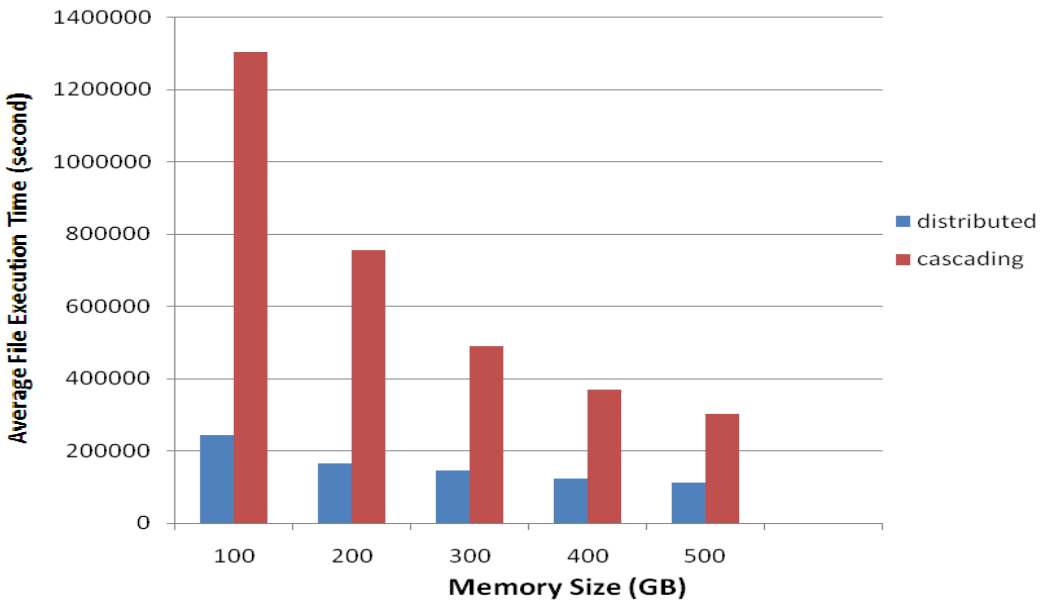

Figure-10**: Varying storage capacity in the grid under Geo-Static access pattern.**

Figure 9 and 10 give the same observation that distributed greedy replication algorithm performs much better than that of cascading under both Random and Geo-Static access pattern. Moreover, for both distributed greedy and cascading algorithms, the average file execution time under geo-static is smaller than that under random (between 20% to 40%).

Figure 11 shows the average file execution time with the increase of the storage capacity under Geo-Dynamic access pattern. Again, compare to Geo-Dynamic pattern, the execution time for both algorithms increase due to the dynamic change of the access pattern. However, as shown in Figure 12, for most of the part, our distributed replication algorithm adapts to the dynamic access pattern bettern than cascading except at the smallest storage capacity case.

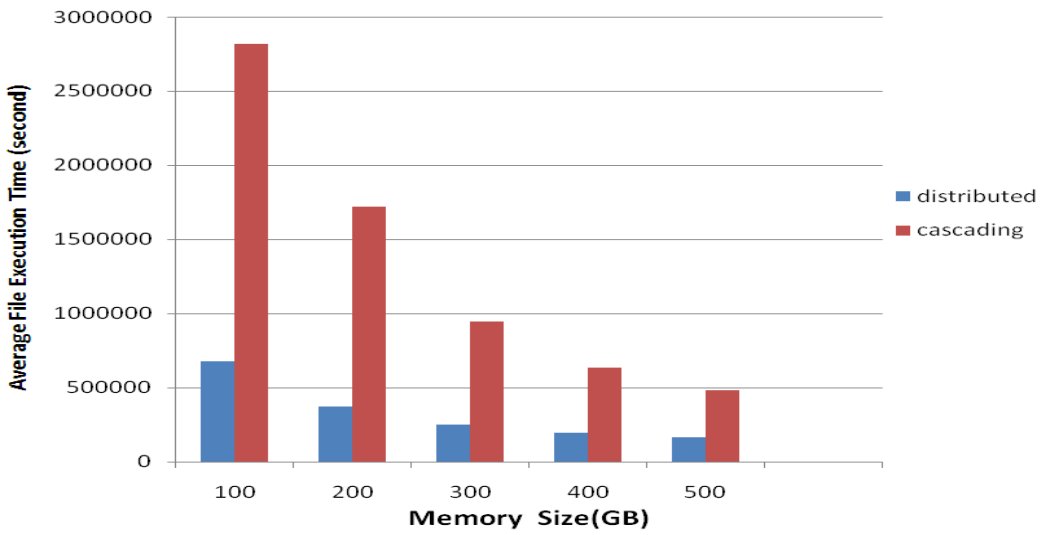

Figure-11 **Varying storage capacity in the grid under Geo-Dynamic access pattern.**

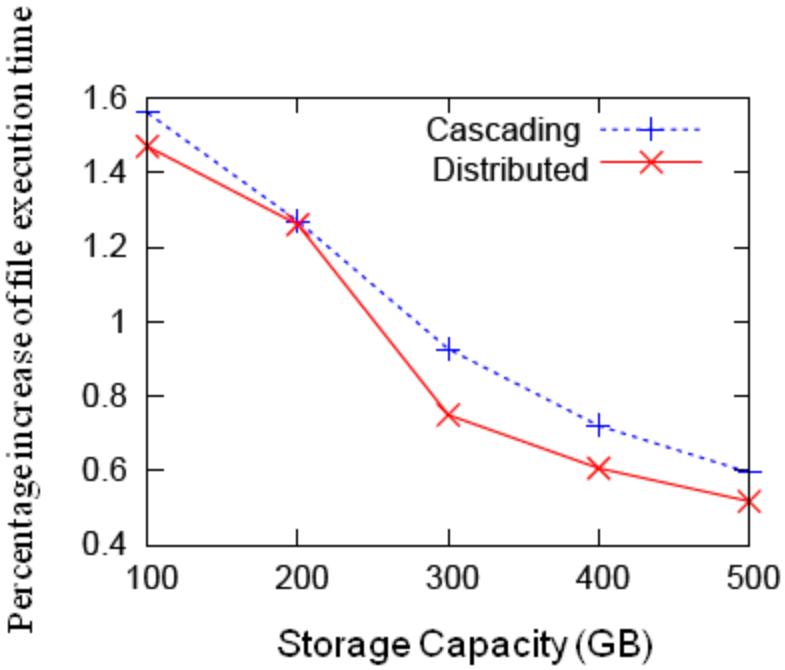

Figure-12: **The percentage increase of average execution time due to dyanmic access pattern for fixed number of files.**

Chapter 5

CONCLUSION

We studied data replication in data intensive applications. Our goal is to efficiently reduce the execution time of data files, which are needed for job executions at each Grid site. We propose a centralized greedy algorithm with performance guarantee, and shows it performs comparably with the optimal algorithm. We also propose a distributed algorithm and implement it in GridSim, a popular Grid simulator, and compare it with some existing distributed data replication technique ‘called cascading plus caching’ proposed by Ranganathan and Foster.

Chapter 6

FUTURE WORK

An effective job scheduling algorithm along with data replication would be more effective in reducing the time taken for simulation, as job scheduling will reduce the wait time of each job in a queue. In the above model, we only changed the job access patterns when all the old jobs are completely executed. But changing the job access pattern with respective to time stamps would be another interesting issue. Work flow in data grid would be another issue to be considered.

REFERENCES

REFERENCES

[1] The large hadron collider. http://public.web.cern.ch/Public/en/LHC/LHC-en.html [date retrieved: 14 December 2009].

[2] Lcg computing fabric. http://lcg-computing-fabric.web.cern.ch/lcg-computing-fabric [date retrieved: 14 December 2009].

[3] Worldwide lhc computing grid. http://lcg.web.cern.ch/LCG [date retrieved: 14 December 2009].

[4] A. Aazami, S. Ghandeharizadeh, and T. Helmi. Near optimal number of replicas for continuous media in ad-hoc networks of wireless devices. In Proc. International Workshop onMultimedia Information Systems, 2004.

[5] 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.

[6] W. H. Bell, D. G. Cameron, R. Cavajal-Schiaffino, A. P. Millar, K. Stockinger, and F. Zini. Evaluation of an economy-based file replication strategy for a data grid. In Proc. of International Workshop on Agent Based Cluster and Grid Computing at CCGrid 2003.

[7] D. G. Cameron, A. P. Miller, C. Nicholson, R. Carvajal-Schiaffino, K. Stockinger, and F. Zini. Analysis of scheduling and replica optimization strategies for data grids using optorsim. Journal of Grid Computing, 2:57-69, 2004.

[8] M. Carman, F. Zini, L. Serafini, and K. Stockinger. Towards an economy-based optimization of file access and replication on a data grid. In Proc. of International Workshop on Agent Based Cluster and Grid Computing at CCGrid 2002.

[9] A. Chakrabarti and S. Sengupta. Scalable and distributed mechanisms for integrated scheduling and replication in data grids. In 10th International Conference on Distributed Computing and Networking (ICDCN 2008).

[10] R.-S. Chang and H.-P. Chang. A dynamic data replication strategy using access-weight in data grids. Journal of Supercomputing, 45:277-295, 2008.

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

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

[13] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, and B. Moe. Wide area data replication for scientific collaboration. In Proc. of The 6th IEEE/ACM International Workshop on Grid Computing, 2005.

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

[15] J. Rehn et. al. Phedex: high-throughput data transfer management system. In Proc. of Computing in High Energy and Nuclear Physics (CHEP 2006).

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

[17] I. Foster and K. Ranganathan. Decoupling computation and data scheduling in distributed data-intensive applications. In Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, HPDC-11, pages 352-358, 2002.

[18] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In Proc. of ACM International Conference on Mobile Computing and Networking (MOBICOM), 2000.

[19] 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.

[20] S. Jin and L. Wang. Content and service replication strategies in multi-hop wireless mesh networks. In Proc. of ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM), 2005.

[21] H. Lamehamedi, B. K. Szymanski, and B. Conte. Distributed data management services for dynamic data grids.

[22] M. Lei, S. V. Vrbsky, and X. Hong. An on-line replication strategy to increase availability in data grids. Future Generation Computer Systems, 24:85-98, 2008.

[23] 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.

[24] S. M. Park, J. H. Kim, Y. B. Lo, and W. S. Yoon. Dynamic data grid replication strategy based on internet hierarchy. In Proc. of Second International Workshop on Grid and Cooperative Computing at GCC 2003.

[25] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas.In Proc. of IEEE Conference on Computer Communications (INFOCOM), 2001.

[26] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn,D. Meyers, and M. Samidi. Scheduling data-intensive workflows onto storage-constrained distributed resources. In Proc. of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007), pages 401 -409.

[27] K. Ranganathan and I. T. Foster. Identifying dynamic replication strategies for a high-performance data grid. In GRID '01: Proceedings of the Second International Workshop on Grid Computing, pages 75-86, London, UK, 2001. Springer-Verlag.

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

[29] F. Schintke and A. Reinefeld. Modeling replica availability in large data grids. Journal of Grid Computing, 2(1), 2003.

[30] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya. A toolkit for modelling and simulating data grids: An extension to gridsim. Concurrency and Computation: Practice and Experience, 2007.

[31] B. Tang, S. R. Das, and H. Gupta. Benefit-based data caching in ad hoc networks. IEEE Transactions on Mobile Computing, 7(3):289-304, March 2008.

[32] M. Tang, B.-S. Lee, C.-K. Yeo, and X. Tang. Dynamic replication algorithms for the multi-tier data grid. Future Generation Computer Systems, 21:775-790, 2005.

[33] M. Tang, B.-S. Lee, C.-K. Yeo, and X. Tang. The impact of data replication on job scheduling performance in the data grid. Future Generation Computer Systems, 22:254-268, 2006.

[34] U. Cibej, B. Slivnik, and B. Robic. The complexity of static data replication in data grids. Parallel Computing, 31(8-9):900-912, 2005.

[35] S. Venugopal and R. Buyya. An scp-based heuristic approach for scheduling distributed data-intensive applications on global grids. Journal of Parallel and Distributed Computing, 68:471-487, 2008.

[36] X. You, G. Chang, X. Chen, C. Tian, and C. Zhu. Utility-based replication strategies in data grids. In Proc. of Fifth International Conference on Grid and Cooperative Computing (GCC 2006), pages 500-507.

[37] I. Foster, C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kaufman, 1999.

[38] I. Foster. Grid Computing, Present at Advanced Computing and analysis Techniques in Physics Research (ACAT) 2000.

[39] I. Foster, C. Kesselman and S.Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing applications Vol-15 (200-22,200).

[40] A. Chervenak, E. Deelman, C. Kesselman, L. Pearlman, and G. Singh. A Metadata Catalog Service for Data Intensive Applications. GriPhyN technical report, 2002-11 2002.

[41] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris.Efficient Replica Maintenance for Distributed Storage Systems (2006).

[42] Kavitha Ranganathan and Ian Foster.Design and Evaluation of Dynamic Replication Strategies for a High-Performance Data Grid.

[43] Henri Casanova. Distributed Computing Research Issues in Grid Computing (2002).

[44] Ian Foster, Carl Kesselman.Computational Grids.

[50] K.Ranganathan, A.Iamnitchi, I.Foster. Improving data availability through dynamic model-driven replication in large peer-to-peer communities.

[51] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, Steven Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets.