2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing

FIRE: A File Reunion Based Data Replication Strategy for Data Grids

Abdul Rahman Abdurrab
Department of Computer Science
San Diego State University
San Diego, USA
abdulrahman004@gmail.com

Tao Xie
Department of Computer Science
San Diego State University
San Diego, USA
xie@cs.sdsu.edu

***Abstract*—Data grids provide large-scale geographically distributed data resources for data intensive applications such as high energy physics and bioinformatics. Optimizing the use of data resources to maximize the performance is critical for data grids. In this paper, we propose a novel dynamic data replication strategy called FIRE (file reunion), which is motivated by the observations that a group of jobs in a site tend to demand a common set of files distributed in a data grid. The basic idea of FIRE is to reunite the file set through data replication so that it resides with the job group on the same site. Extensive experiments using a well-known data grid simulator OptorSim and synthetic benchmarks demonstrate that compared with two existing schemes, LRU (Least Recently Used) and LFU (Least Frequently Used), FIRE performs obviously better in most scenarios.**

I. INTRODUCTION

All manuscripts Grid computing is an emerging technology with the goal of providing geographically distributed users with virtual organizations, which allow the effective sharing of computation and storage resources [10][29]. In particular, a data grid deals with the controlled sharing and management of large amounts of data in the form of files, which are distributed among geographically diverse grid sites [1][4][7][10][12][29]. Typical data grid applications like high energy physics [1][9], image processing [17], bioinformatics [23], data mining [20], and spatial processing [21] normally demand enormous computational power as well as a vast amount of data in terabyte or even petabyte scale[22][25][27].

Efficiently managing such huge and widely distributed data in a data grid, however, is a great challenge [9][26]. To solve this challenge, an array of techniques including dynamic data transfer [15][21], data-driven job scheduling[7][17][23], and data replication [1][6][9][11][18] have been proposed in the literature. The major goal of these techniques is to reduce the execution cost of a job, which depends not only on the computational resource assignment but also on the location of data files that are required by the job [1]. Data replication has long been considered as an important technique for improving performance in data grids because it can noticeably reduce the network bandwidth consumption and the access latency [2][6][7]. Besides, it can effectively enhance data availability, fault tolerance, system scalability, and load balancing by creating replicas and dispersing them among multiple sites [9][18]. A data replication process involves creating identical copies of a file and placing them onto multiple sites so that they can be accessed simultaneously from various locations. Three main decisions need to be made by a replication mechanism: which files to replicate, when to replicate, and where to replicate [23]. If there are several existing replicas of a given file, the replication mechanism also needs to determine the best replica to import. One key principle for data replication mechanisms is that a replica and the job that requires it should be allocated onto the same site whenever the storage element of the site can accommodate the replica. Otherwise, the replica should be stored onto a nearby site [12][12]. Since the popularity of a file could change over time, the replication service needs to replicate files with increasing popularity and to delete the ones with reduced popularity [1] [1]. More precisely, when the popularity of a file reaches a predefined threshold value, replication or deletion of replicas of the file has to be completed after analyzing the access patterns of previous file requests [13][13].

File access pattern analysis has always been employed as a powerful tool to design efficient dynamic data replication schemes [17][21]. Most of existing dynamic data replication mechanisms [6][9][13][24], however, mainly focus on identifying popular files based on file access patterns observed at application run times. We argue that file access pattern can provide us with more useful information than the popularity of files. For example, after analyzing a real data-intensive grid application *Coadd* [21], Ko *et al*. found that a significant number of files accessed by multiple tasks and a large number of tasks access the same set of files during their execution [17]. The important phenomenon was named *locality of interest* [17] and it has also been noticed in many other data-intensive grid applications like high energy physics and bioinformatics [16]. The implication of *locality of interest* is that a group of jobs has a strong correlation with a set of files in many data-intensive grid applications. Motivated by this critical observation, we propose a file reunion based dynamic data replication strategy for data grids.

Initially, only one set of files known as master files exist on a special site called data site, which normally only has a storage element. None of the master files has a replica at this moment. When a job submitted to a normal site (hereafter, site) with both storage element and computing element

978-0-7695-4039-9/10 $26.00 © 2010 IEEE
DOI 10.1109/CCGRID.2010.12

requires a file that is not locally available, the file will be replicated from the data site to the site. In this way replicas of the master files are spread out randomly across the sites after a period of time. After a site runs out of its storage space, a job on the site has to remotely access files that reside on other sites. Since they are copies of the master files, files on a site could be deleted or moved away when there is no enough storage space on the site to accommodate new files. Each site maintains a file access table to keep track of its local file access history. A site makes independent decisions on whether or not deleting some old files to accommodate new files if some benefits can be achieved. On each site, the FIRE strategy analyzes the site's file access table. If on a site FIRE detects that a common set of files are frequently accessed by a group of local jobs, in other words, a strong correlation between a group of local jobs and a set of distributed files exists, FIRE will reunite the set of files onto the site by replication. Since files that belong to the same set (or family) have been initially separated among various sites, FIRE makes the file set reunion after the strong correlation between the file set and the job group exhibits. Section 3.1 provides an example to show the basic idea of FIRE.

The rest of this paper is organized as follows. Section 2 gives a brief introduction of related work. Section 3 presents an overview of the grid architecture, which will be used in this work. Section 4 proposes the FIRE strategy with an illustrative example. An extensive simulation study will be discussed in Section 5. Finally, section 6 concludes the work with some future directions.

II. RELATED WORK

Compared with transferring a file from one site to the other in real time, data replication is a better technique as it can reduce bandwidth consumption and access latency[9] [9]. In addition, it improves data availability, data reliability, and system scalability. Data replication algorithms for data grids can be broadly classified into two camps: static and dynamic [18][19][22][26]. Static replications create algorithms based on a set of predefined rules, which requires a complete and prior knowledge of workload characteristics [9][12]. The implication is that they cannot adapt to the changing network or access pattern, which is common in data grids. Dynamic replication, on the other hand, makes data replications based on real-time network condition and access patterns, and thus, is more suitable for data grids where resources and workload statistics are continuously changing. In this section, we introduce some typical dynamic data replication algorithms.

To ensure scalability of the distributed systems, a multi-tier architecture has been proposed for data grids [26]. Fig. 1 shows its architecture. Each tier represents a different region. For example, Tier-0 is stores the raw data generated by the experiments in CERN [28] and Tier-1 processes the data analysis by several regional centers. Leafs represent the clients, and each client only accesses the replicas from its ancestor. Two dynamic replication strategies, namely, SBU (Simple Bottom Up) and ABU (Aggregate Bottom Up), were developed in [26].

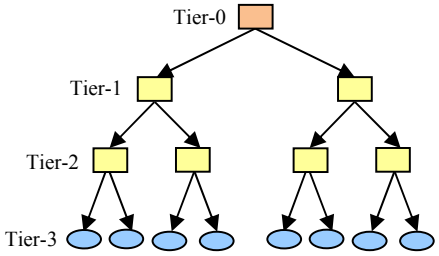


Fig. 1. The multi-tier architecture.

In contrast to traditional data replication where a complete file is replicated, Chang *et al*. proposed a new data replication approach called *fragmented replicas*, which only replicates needed partial contents of a file locally to save storage space [8]. However, the problem of fragmented replica updates is to be solved and it is challenging. Also, their assumption that fragmented replicas are contiguous may not always hold in data grids. Further, they developed a job scheduling policy HCS (Hierarchical Cluster Scheduling) and a dynamic data replication strategy HRS (Hierarchical Replication Strategy) to improve the data access efficiencies in a cluster grid [7]. The rationale behind the two coordinated algorithms is that job scheduling in concert with data replication can significantly reduce data access time and the amount of inter-cluster-communications. In order to design an effective data replication algorithm, file access pattern analysis is frequently employed [6][9][13][24]. Sato *et al*. model the replication problem as a combinatorial optimization problem with constraints from the access time threshold and various system parameters. The objective of their algorithm is to minimize file replication costs by dynamically monitoring and estimating inter-node link throughputs and file access patterns of running applications [24].

A very novel ant-like agent-based data replication mechanism called ARMAP protocol (Ant-based Replication and Mapping) has been developed by Forestiero *et al*. [15]. Agents travel a data grid through P2P interconnections and replicate data whenever necessary. The protocol can enforce the dissemination of descriptors related to high Quality of Service (QoS) resources. One potential problem of this approach is that security could become a concern when agents themselves are infected by computer viruses. Although improving performance in terms of reducing access latency is always the goal of data replication algorithms, data availability is also very important for the users of a data grid. To maximize the data availability, Lei *et al.* devised an on-line data replication optimizer algorithm called MinDmr [19].

A comprehensive investigation on dynamic data replication strategies for high-performance data grids has been conducted by Ranganathan and Foster [22]. They developed a simulation framework, which enables comparative studies of alternative dynamic replication strategies including No Replication or Caching, Best Client, Cascading Replication, Plain Caching, Caching plus Cascading Replication, and Fast Spread. The performance of

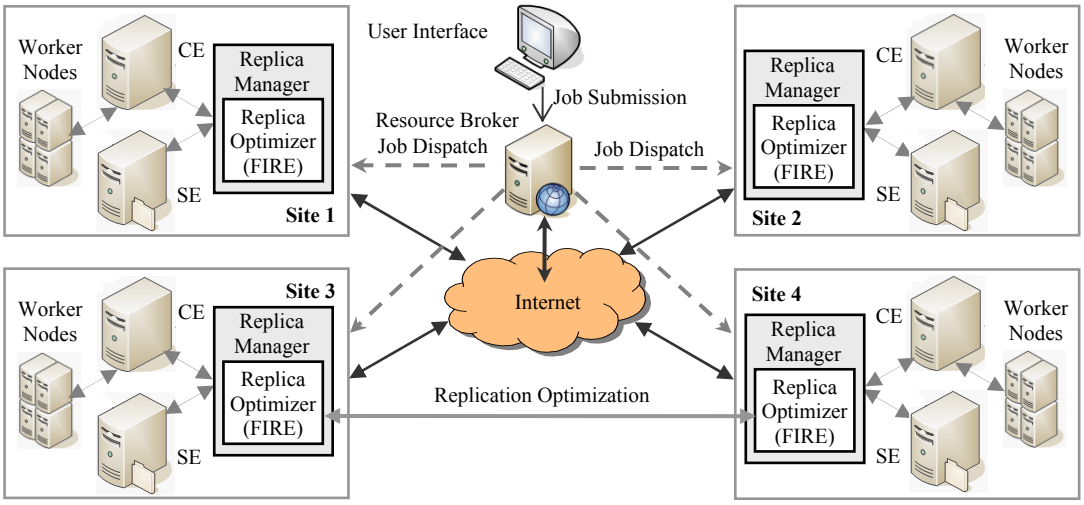


Fig. 2. The data grid architecture.

the six replication strategies was evaluated under three distinct types of access patterns: random access pattern with no locality, a small amount of temporal locality, and a small amount of geographical and temporal locality [22]. Their simulation results show that significant savings in latency and bandwidth can be obtained when the access patterns contain a small degree of geographical locality, which indicates that files recently accessed by a client are likely to be accessed by nearby clients. We view this crucial finding as a strong evidence that a group of geographically close jobs (e.g., staying on one site) tend to access a common set of files, which justifies the fundamental assumption on access patterns used in this research.

III. Architecture Overview

The architecture of the data grids simulated by the OptorSim simulator [2][3][4] is shown in Fig. 2. We adopt this architecture as it has been widely used for studying various file replication algorithms in conjunction with different job scheduling algorithms in data grid environments [1][3][9]. OptorSim provides a framework to simulate real-world data grids by considering an array of parameters and scenarios found in reality. It was developed by the European Data Grid Project [28]. One data grid consists of several sites. Each site has either a computing element (CE) or a storage element (SE) or both (Fig. 2). Geographically distributed sites are connected by the internet. End users use a machine running the user interface (UI) software to interact with the grid system (Fig. 2). More precisely, end-user uses the machine to submit jobs to the grid system and retrieve output of the completed jobs. The user interface may be also used to monitor the execution of jobs after submission. All submitted jobs will then be forwarded to a resource broker (RB), which matches the jobs' requirements to the available resources at the various sites within the grid and dispatches them [4].

A CE is composed by a gatekeeper machine and a number of worker nodes [4] (Fig. 2). A gatekeeper is the front-end of a computing element. It accepts jobs and then dispatches them for execution on the worker nodes. Finally, it returns the results to RB. This way the gatekeeper provides a uniform interface to the computational resources that it controls. The SE provides uniform access to large storage spaces [2]. The storage element may control large disk arrays or mass storage systems. SE hides the details of the backend storage systems and provides a uniform interface to the grid users. At each site, there is a software component called replica manager (RM), which manages the data flow between sites. Within a replica manager, there is a subcomponent called replica optimizer (RO), whose major duty is to create file replicas and to destroy them whenever necessary [1]. The replica manager possesses the knowledge of the local file access patterns as well as the access history of files that were requested by remote jobs. Various dynamic data replication algorithms like FIRE can be integrated into the replica manager as a replica optimizer. Therefore, their performance can be studied and compared in a fair manner.

A data grid is a highly dynamic environment for the status of its resources can change during the execution of its jobs [29]. Hence, optimizations are needed in every stage from job submission to data replication [13]. The first optimization phase occurs when the RB determines the CE on which a job should run. This process is called scheduling optimization as the RB has to select a CE from many available ones. The scheduling algorithm mainly considers parameters like a job's access cost, which is the sum of the costs that are incurred in bringing all the files required by the job. Besides, the scheduling algorithm also takes the job's execution time on a CE into account. Moreover, it needs to consider the existing load on a site to avoid load imbalance. Only after the CE finishes its ongoing jobs will

Site 1 File Access Table		
File	Job	Accesses
A	J_1,J_2,J_4	3,3,4
B	J_1,J_2,J_4	3,4,4
C	J_1,J_2	2,2
D	J_5(R),J_6 (R)	1,2
E	J_5(R),J_3 (R)	2,1
F	J_7 (R)	1
G	J_8(R)	1
K(R)	J_1,J_2	3,2
L(R)	J_2,J_4	2,2

Site 2 File Access Table		
File	Job	Accesses
H	J_3,J_5,J_6	4,4,3
I	J_3,J_5,J_6	3,4,4
C	J_3,J_5	2,2
K	J_1(R),J_2 (R)	3,2
L	J_2(R),J_4(R)	2,2
M	J_9(R)	1
N	J_{10}(R)	1
D(R)	J_5,J_6	1,2
E(R)	J_5,J_3	2,1

Files on Site 1
A
B
C
K
L
D (D)
E (D)

Files on Site 2
H
I
C
D
E
K (D)
L (D)

(a) Access histories before optimization (b) After optimization

Fig. 3. An example of the FIRE strategy.

it be able to process newly arrived jobs. The waiting cost is called queuing time. In addition to scheduling optimization, data optimization is needed as well. This optimization task is accomplished by the RO where a dynamic data replication strategy like FIRE is implemented (Fig. 2). For example, when a RM selects a replica from multiple available ones, it should bring the one that will incur the least cost in terms of latency and network bandwidth. Thus, the replica optimizer has to weigh the cost of each replica and informs the RM the best replica to fetch. Note that the RO is the "brain" of the RM, which is an interface to send/receive replicas. Further data replication optimization can be done by the RO. For instance, FIRE within the RO can reunite a set of distributed files onto one site after a strong correlation between the set of files and a group of jobs on the site exhibits. FIRE achieves optimization based on file access tables.

IV. THE FIRE STRATEGY

In this section, we first present an example to show the basic idea of FIRE, which is followed by a brief introduction of the auction protocol, as well as a detailed analysis of the FIRE strategy.

A. An Illustrative Example

An example to show the basic idea of the FIRE strategy is illustrated in Fig. 3. Assume that there is a set of files on Site 1 and Site 2 as given in Fig. 3a. For illustration purpose, suppose that all files are of the same size (e.g., 1 GB) and the SE of a site can only accommodate 7 files. Initially, there is just one set of master files available in a data grid. Assume that the master file set is present on Site 0, which only has a SE. When jobs are submitted to the data grid, they run at different sites. Replication takes place when a job on a site requires a file that is not locally available and the site has enough space to accommodate the replica. When a site's SE has no free storage space, all jobs on it have to remotely access files if they are not locally available. After a period of time, files (i.e., replicas) are spread out randomly across the sites in the data grid. These files are copies of the master files, and thus, can be deleted. Every site individually keeps track of its access histories for all local files and remote files accessed by the jobs on it. The replica manager of a site maintains a File Access Table as shown in Fig. 3a. While site 1 has files *A, B, C, D, E, F*, and *G*, Site 2 has files *H, I, C, K, L, M,* and *N*. On Site 1, job J_1 accessed files *A, B*, and *C* 3, 3, and 2 times, respectively. Job J_2 accessed files *A, B*, and C 3, 4, and 2 times, respectively. File *D* was remotely accessed by J_5 and J_6 1 and 2 times, respectively. File *E* was remotely accessed by J_3 and J_5, which are running on Site 2. Files *F* and *G* have no local access but each of them has one remote access. Note that J_1 and J_2 also accessed file *K* remotely. Similarly, J_2 and J_4 accessed *L* remotely (Fig. 3a). The correlation degree between a file f_i and a job j_k is defined as the number of times j_k accessed f_i and it is denoted as c_i^k. The correlation degree between a file f_i and a group of local jobs $J = \{j_a, j_b, j_k, \ldots, j_q\}$ that have accessed it is defined as group correlation degree, which is the total number of times that f_i was accessed by jobs in *J*. Group correlation degree of a file f_i is denoted as $G_i = \sum_{j_k \subset J} c_i^k$. At the first time a job needs to access a remote file, the file is not immediately replicated onto the job's site. Until FIRE makes sure that there exists a benefit in replicating the file, the job will continue to access the file remotely. The cost of replication of a file includes access cost and the queuing time at the site where the wanted file stays. The access cost is determined by network bandwidth available between the two sites and the size of the file. For a file to be replicated onto a site, the site should also delete a file if it has no free storage space and each file has the same size. If there is a file whose group correlation degree is lower than the number of times the remote file is being accessed, the file at the requesting site is deleted and the remote file is bought in. Similarly, the access history of files on Site 2 is given in Fig. 3a. We can see that files *H, I, C* are accessed frequently by J_3, J_5, and J_6. Also, these jobs access files *D* and *E* on Site 1 remotely.

After a certain period of time, FIRE analyzes the file access tables on both Site 1 and Site 2. It detects that there exists a strong correlation between a local job group (J_1, J_2, J_4) and a file set (*A, B, C*) on Site 1. Meanwhile, FIRE found that jobs in the group of (J_1, J_2, J_4) were correlated with files *K* and *L* on Site 2 as they remotely accessed them multiple times. Thus, FIRE decided to replicate *K* and *L* on Site 1 so that the file set (*A, B, C, K, L*) and their correlated job group (J_1, J_2, J_4) can stay together (Fig. 3b). In order to accommodate *K* and *L*, FIRE has to delete two files on Site 1 due to the limited storage space. Since *F* and *G* were only accessed once remotely so far, they have no correlation with local jobs. Therefore, FIRE deleted them to accept *K* and *L* on Site 1 (Fig. 3b). Similarly, on Site 2 files *H, I*, and *C* were

kept there while files *D* and *E* were replicated from Site 1 because they were frequently accessed by jobs J_3, J_5, and J_6. Files *M* and *N* were deleted from Site 2 because they have no local access (Fig. 3b). After data replication optimization, we see that file *C* is present on both sites as it has a strong correlation with both job group (J_1, J_2, J_4) on Site 1 and job group (J_3, J_5, J_6) on Site 2 (Fig. 3b). On Site 1, since files *D* and *E* have no local accesses, these files are likely to be deleted if more replication has to occur in the future. Likewise, files *K* and *L* will be deleted if any replication has to happen on site 2 in the future.

B. The Auction Protocol

When FIRE needs to make a replication of a file and there are multiple copies of the file in a data grid, FIRE has to decide which site to choose to make the replication. Obviously, FIRE prefers to make a replication from a site that will incur the least cost in terms of access time and the site's job queue cost. The job queue cost of a site is defined as the earliest time that a job can start its execution if it is assigned on the site. FIRE employs an auction protocol named *Vickery Auction* proposed in [1] to accomplish the best replica selection process. Whenever FIRE finds multiple copies of a file that it wants to make a copy, it starts an auction process by sending a bidding request to all sites that have the file. Each site that received the request will then send a bid back to FIRE. After examining all the bids, FIRE selects the site that offered the lowest bid (i.e., the least cost in access time and job queue cost) as the winner, who only has to pay the price of the second lowest bid. This way FIRE replicates the file from the winner of the auction. If there is only one replica existed in the data grid, FIRE directly copies the file from that site. More detailed information about the Auction Protocol can be found in [1].

C. Strategy Description

The Fig. 4 outlines the FIRE strategy. Assume that a data grid consists of m sites and it is represented by the *set* $S = \{s_1, ..., s_j, ..., s_n\}$. Also, assume that there are master files denoted by $F = \{f_1, ..., f_i, ..., f_m\}$ in the data grid. When a job j is submitted to a data grid, the resource broker receives the job and then makes a decision about where to dispatch it. This decision is made based on the scheduling algorithm utilized at the resource broker. Here, we use the scheduling algorithm that takes the file access cost and job queue cost into consideration. The sum of these two costs is calculated for all *n* sites (Step 2 – Step 4). The least-total-cost site *s* is chosen to execute the job (Step 5). It is highly possible that site s_l has all or part of the files required by *j*. Assume that files required by *j* constitute a set $D = \{d_1, ..., d_k, ..., d_w\}$ where $w <= m$ (Step 6). For each file d_k in D, FIRE checks if it is locally available. If not, FIRE tests whether or not local SE has space to accommodate d_k. If yes, FIRE replicates d_k from a site with least cost based on AP (the Auction Protocol described in Section 4.2) (Step 9 – Step 11). If not, FIRE compares the number of remote accesses on d_k issued from site s_l against the number of local accesses

```
1. for each job j submitted to the RB of a data grid do
2.   for each site in the data grid do
3.     Calculate its file access cost and job queue cost
4.   end for
5.   Dispatch j to the least-total-cost site s_l
6.   Files in D = {d_1, …, d_k, …, d_w} are required by j
7.   for each file d_k in D do
8.     if d_k is not locally present on s_l
9.       if d_k can be accommodated by local SE
10.        Use the Auction Protocol to replicate d_k on s_l
11.        Space of local SE is reduced by the size of d_k
12.      else
13.        RA = number of remote accesses on d_k from s_l
14.        LAF = the local file received least local access
15.        if RA >= the number of local accesses of LAF
16.          Delete LAF
17.          Use the Auction Protocol to replicate d_k on s_l
18.        else
19.          Access d_k remotely
20.          RM of the remote site updates its access table
21.        end if
22.      end if
23.    end if
24.  end for
25.  Update the file access table on s_l
26.end for
```

Fig. 4. The FIRE strategy.

of the least locally accessed file on s_l. If the former is no less than the latter, the least locally accessed file will be deleted and d_k will be replicated onto s_l using the Auction Protocol (Step 13 – Step 17).Otherwise, job j has to access d_k remotely (Step 19). In this case, the replica manager of the remote site that has file d_k needs to update its file access table to log this remote access event (Step 20). Finally, the replica manager of site s_l updates its file access table to log local accesses and remote accesses caused by job *j* (Step 25).

V. Performance Evaluation

A. Simulation Setups

In this section, we evaluate the performance of our FIRE strategy along with two existing algorithms using a set of high energy physics analysis jobs generated from the CDF experiment in the European Organization for Nuclear Research project [14]. In this real-world based workload, each file has a size of 1 GB and the total size of file set is 97 GB. The simulated grid used in our experiments has 27 sites and the experimental data came from a world wide data

TABLE I. System Parameters

Parameter	Value
Number of sites	27
Storage space at each site (GB)	50
Inter-Cluster bandwidth (Mbps)	10
Intra-Cluster bandwidth (Mbps)	100
Single file size (GB)	1
Total file size (GB)	97
Job inter arrival time (second)	2.5

production generated in [28]. To simplify the simulations, we assume that there is no contending network traffic. To test FIRE under a real world scenario, we adopt this set of jobs from a practical data grid application. Among the 27 sites, 18 of them have a computing element and 20 of them have a storage element. The capacity of each SE is 50 GB. We tested two initial replica distribution cases: *random replicas* and *no replica*. In the first case, the copies of the master files are distributed randomly over various sites before any job is submitted. In the second case, the set of master files stay on a particular site and all other sites have no replica at all. The sites are connected to each other by networking links with a constant bandwidth. The configuration above is stored in OptorSim as a parameter file [4]. Not all sites are directly connected to each other. Since most replication algorithms assume that data is read-only in data grid environments, we adopt this assumption as well. The parameters used are shown in Table 1.

The next configuration file is called the job parameter. It specifies the file names along with their logical ids. It also lists the number of jobs to be run, which in our case will be 100. Each job requires a set of files for its execution. There are six jobs, which will run for a total number of 100 times. We used a scheduling algorithm that makes scheduling decision based on the sum of the access cost and the job queue cost. The site with the least-total-cost will receive the job. The job parameter file also gives the submission frequency of each job. The job submission frequency is configured as 2.5 seconds in our simulations. The final configuration file is called the bandwidth configuration file. It is used to simulate background traffic in the simulated data grid at different times. We used the default bandwidth configuration file.

In addition to total execution time, we also measured effective network usage (ENU), which was defined in [5]. Assume that $N_{remote\ file\ accesses}$ is the number of times the CE reads a file from a SE on a different site, $N_{file\ replications}$ is the total number of file replications, and $N_{local\ file\ accesses}$ is the number of times a CE reads a file from a SE on the same site. ENU is equal to $(N_{remote\ file\ accesses} + N_{file\ replications})/N_{local\ file\ accesses}$. For a particular network topology, a lower value of ENU indicates a better performance for a data replication strategy for data grids [5].

B. Experimental Results

We conduct a group of simulation experiments to examine the effectiveness of the FIRE strategy. We run the six jobs totally 100 times and measured the total execution times. In the first simulation experiment, we evaluate the impacts of file access pattern on FIRE. We tested FIRE and the two existing algorithms in both sequential file access pattern and random file access pattern. We first set the file access parameter to sequential access, which means that all files of a given file set will be accessed in a sequential fashion. Next, we tested the three algorithms for random a access pattern where each job accesses files in random manner. The number of reads on a file issued from a job was randomly generated. Fig. 5a shows the total execution time of the three algorithms. Under sequential access pattern, FIRE obviously outperforms the two baseline algorithms. More precisely, compared with LRU and LFU, FIRE improves job execution performance by 25% and 14%, respectively (Fig. 5a). This is because FIRE knows that all the files required by a particular job will be accessed sequentially. Hence, these files will not be deleted and shall be accessed later by the same job or related jobs, which results in performance improvement. On the other hand, for LRU and LFU, if the file set of required by a particular job is larger than storage space available in a SE, the existing files in the SE will be deleted. Therefore, the next time the job runs, RO has to do a remote read and replicate the file again, which leads to performance degradation. Under the random access pattern, a job could access any files from its candidate file set during its execution. For example, assume that job *j* has a candidate file set *(A, B, C, D)*. In its first run, *j* may only access file A. However, in its second run, *j* accesses both file *C* and *D*. In short, each time when *j* is running, it could access any number of files in the candidate file set. Clearly, we can see that the performance of FIRE decreases while the performance of LRU and LFU increases. This is because the history access information has little help for FIRE to keep files that will be accessed in the near future as file access is totally random. LRU and LFU can still keep some recently access files on local SE due to the nature of these two mechanisms. This is the reason why the total execution time of LRU and LFU decreases in random pattern. Fig. 4b shows the ENU performance of all three algorithms. While the ENU for FIRE is around 0.37, the

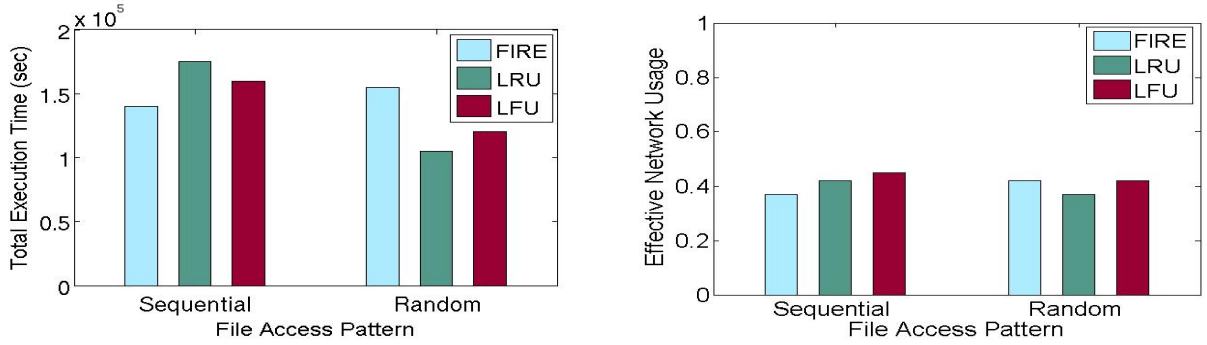
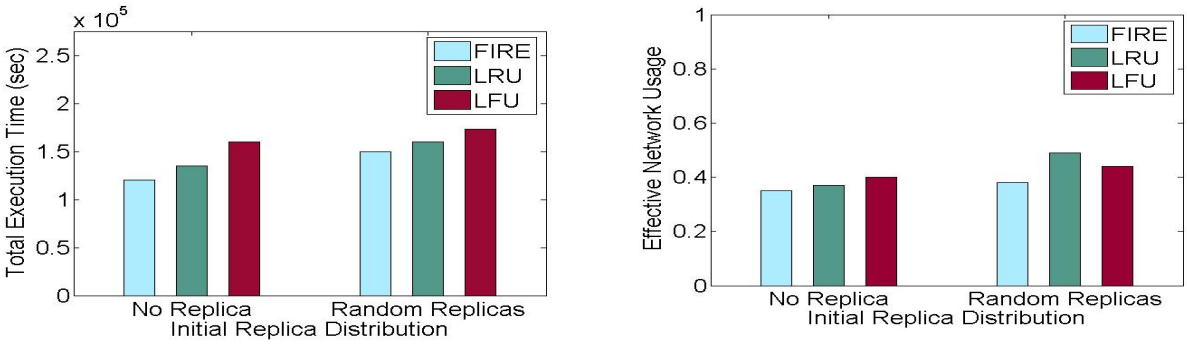


Fig. 5. Performance Impact of file access pattern.



(a) (b)

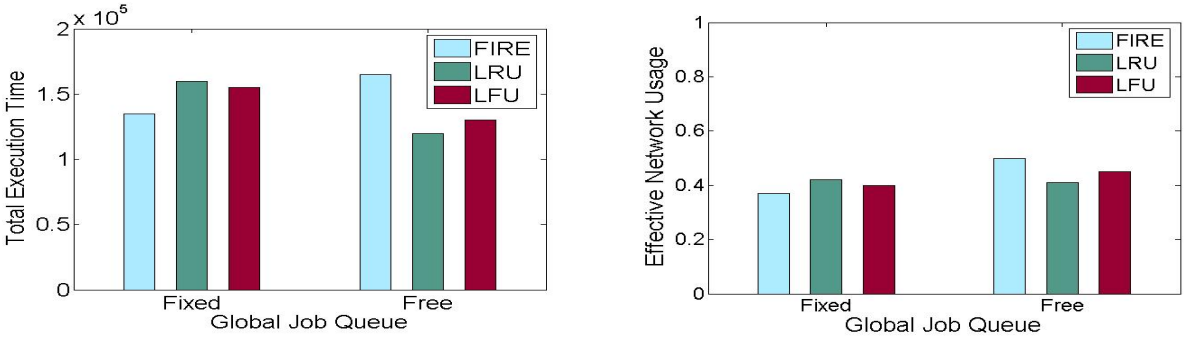
Fig. 6. Performance Impact of initial state.

ENU for LRU and LFU are 0.42 and 0.45, respectively. FIRE gains a better ENU value under sequential access pattern. However, it performs slightly worse than LRU (Fig. 5b) when the access pattern random is chosen. This is because FIRE moves around files more often when files are accessed randomly, and thus, the network usage was increased.

The second group of experiments is to examine the performance of FIRE under two initial replica distribution scenarios: *random replicas* and *no replica*. In the first case, there is just one set of master files on a particular site while all other sites' SEs are empty with no replica at all. Experiments in this scenario help us understand how much time a replication algorithm needs to fill in sites with replicas. In the second scenario, initially all sites are randomly filled with some replicas. In the first scenario as shown in Fig. 6a, compared with LRU and LFU, FIRE reduces total execution time by 7.4% and 19.3%, respectively. The implication is that FIRE creates replicas faster than LRU and LFU. Thus, it leads to a lower total execution time.

In the second scenario, we examine the total execution time of jobs when initially replicas are spread randomly among sites. FIRE still outperforms the two baseline algorithms in terms of total execution time (Fig. 6a). In particular, FIRE achieves 3.2% and 12.8% improvements in total execution time compared with LRU and LFU, respectively. However, we notice that for each algorithm its total execution time increases compared with the *no replica* scenario. This is because initially the sites are randomly filled with replicas, and thus, it takes time for the replicas to get grouped along with similar files in the same set. Due to this fact, the network usage is also increased, which can be observed in Fig. 6b. The value of ENU is also enlarged from the first scenario to the second scenario due to increased network activities. Nevertheless, FIRE consistently performs better in terms of ENU (Fig. 6b), which demonstrates that FIRE is a more efficient replication strategy due to its lower network usage.

The purpose of the last group of simulations is to evaluate the impacts of global job queue construction on the performance of FIRE. In our simulations, we have 6 distinct jobs (hereafter, candidates) and each of them can run multiple times. Each execution of a job is called a job instance and there are totally 100 job instances. The 100 job instances (hereafter, jobs) stays in the global job queue, which resides on the resource broker (see Fig. 2). There are multiple ways to construct the global job queue. One way we called *free* is that each of the 100 jobs in the global job queue is randomly selected from the 6 candidates. The global job queue built in this way represents a random job submission pattern. Another way we called *fixed* is that each of the 6 candidates has a specific percentage weight in the range [0, 100%] and the sum of these percentage weights is equal to 1. For example, a global job queue constructed in this manner could be $(10\,j_1, 10\,j_2, 10\,j_3, 10\,j_4, 30\,j_5, 30\,j_6)$. In other words,



(a) (b)

Fig. 7. Performance Impact of global job queue pattern.

each of the 100 jobs in the global job queue is selected from the 6 candidates based on their respective percentage weight. As a result, the global queue constructed in this manner represents a job submission pattern with some "space" locality. We tested these two job submission patterns in Fig. 7. When the global job queue was constructed in the *fixed* manner, in terms of total execution time FIRE outperforms LRU and LFU by 18.2% and 12.9%, respectively (Fig 7a). In the *free* situation, FIRE performs worse than LRU and LFU. The reason is that FIRE depends on the close relationship between jobs and files. If this workload assumption does not hold, i.e., similar jobs are not executed together but rather are mixed with other non-related jobs, the performance of FIRE deteriorates. Hence, the performance of FIRE is worse than that of LRU and LFU. This also explains why LRU and LFU outperform FIRE in ENU when the global job queue was built in the *free* manner (Fig. 7b). Still, FIRE performs better in the *fixed* case

VI. Conclusions and Future Work

Data replication is a frequently used technique that can reduce bandwidth consumption and access latency in high performance data grids where end users demand remote accesses to large files. In this paper, we developed a dynamic data replication strategy called FIRE, which was motivated by an important observation from many data-intensive grid applications. Researchers in [16][17] found that a group of jobs in a site tend to demand a common set of files distributed in a data grid. To optimize replication based on the critical access pattern characteristic, FIRE reunites a set of distributed files onto one site after a strong correlation between the set of files and a group of jobs on the site exhibits.

To evaluate the efficiency of our algorithm, we ran extensive simulation experiments using the OptorSim simulator and a real-world data grid test bed from high energy physics [28]. Total job execution time and effective network usage are used as two performance metrics. We compared FIRE with two traditional algorithms LRU and LFU. The experimental results demonstrate that FIRE noticeably improves the total execution time and ENU under sequential access pattern. The performance enhancement is attributed to FIRE's ability of coupling similar jobs that access a common set of files together through replication. As part of our future work, we plan to extend FIRE to parallel applications where jobs may have precedence constraints and communicate with each other during their executions.

Acknowledgment

This work was supported by the US National Science Foundation under grant number CNS (CAREER)-0845105, CNS-0834466, and CCF-0702781.

References

[1] W.H. Bell, D.G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini, "Evaluation of an economy-based file replication strategy for a data grid," *Proc. IEEE Int'l Symp. Cluster Computing and the Grid (CCGrid)*, pp. 661-668, 2003.

[2] W.H. Bell, D.G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini, "Simulation of dynamic grid replication strategies in optorSim," *Proc. Third Int'l Workshop on Grid Computing,* pp. 46-57, 2002.

[3] W.H. Bell, D.G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini, "OptorSim-a grid simulator for studying dynamic data replication strategies," *Int'l Journal High Perform Computer Application*, Vol. 17, No. 4, pp. 403–416, 2003.

[4] D.G. Cameron, A.P. Millar, and C. Nicholson, "OptorSim: a simulation tool for scheduling and replica optimization in data grids," *Proc. Computing in High Energy and Nuclear Physics (CHEP)*, 2004.

[5] D.G. Cameron, R.C. Schiaffino, A.P. Millar, C. Nicholson, K. Stockinger, and F. Zini, "Evaluating scheduling and replica optimisation strategies in optorSim," *Proc. 4th Int'l Workshop on Grid Computing*, pp. 52-59, 2003.

[6] M. Carman, F Zini, L.Serafini, and K.Stockinger, "Towards an Economy-based optimisation of file access and replication on a data grid," *Workshop on Agent-based Cluster and Grid Computing, in conjunction with Cluster Computing and the Grid (CCGrid)*, May 2002.

[7] R.S. Chang, J.S. Chang, and S.Y. Lin, "Job scheduling and data replication on data grids," *Future Generation Computer Systems*, Vol. 23, Issue 7, pp. 846-860, August 2007.

[8] R.S. Chang and P.H. Chen, "Complete and fragmented replica selection and retrieval in data grids," *Future Generation Computer Systems*, Vol. 23, Issue 4, pp. 536-546, May 2007.

[9] R.S. Chang and H.P. Chang, "A dynamic data replication strategy using access-weights in data grids," *Journal of Supercomputing*, Vol. 45, Issue 3, pp. 277 – 295, 2008.

[10] A. Chervenaka, I. Foster, C. Kesselmana, C. Salisbury, and S. Tuecke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," *Journal of Network and Computer Applications*, Vol. 23, No. 3, pp. 187-200, 2000.

[11] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney, "Giggle: a framework for constructing scalable replica location services," *Proc. ACM/IEEE Conference on Supercomputing*, pp. 1-17, 2002.

[12] U. Cibej, B. Slivnik, and B. Robic, "The complexity of static data replication in data grids," *Parallel Computing*, Vol. 31, Issue 8-9, pp. 900-912, 2005.

[13] M.M. Deris, J.H. Abawajy, and A. Mamat, "An efficient replicated data access approach for large-scale distributed systems", *Future Generation Computer Systems*, Vol. 24, Issue 1, pp. 1-9, January 2008.

[14] European Organization for Nuclear Research (CERN),http://public.web.cern.ch/Public/Welcome.html.

[15] A. Forestiero, C. Mastroianni, and G. Spezzano, "QoS-based dissemination of content in grids," *Future Generation Computer Systems*, Vol. 24, Issue 3, pp. 235-244, March 2008.

[16] A. Iamnitchi, S. Doraimani, G. Garzoglio, "Filecules in High-Energy Physics: characteristics and impact on resource management," *Proc. 15th IEEE Int'l Symp. High Performance Distributed Computing (HPDC)*, pp. 69-80, 2006.

[17] S.Y. Ko, R. Morales, and I. Gupta, "New worker-centric scheduling strategies for data-intensive grid applications," *Proc. ACM/IFIP/USENIX Int'l Conference on Middleware*, pp. 121-142, 2007.

[18] H. Lamehamedi, B.Szymanski, and Z. Shentu, "Data replication strategies in grid environments", *Proc. 5th Int'l Conf. On Algorithms and Architecture for Parallel Processing*, pp. 378–383, 2002.

[19] M. Lei, S.V. Vrbsky, X. Hong, "An on-line replication strategy to increase availability in data grids," *Future Generation Computer Systems*, Vol. 24, Issue 2, pp. 85-98, 2008.

[20] Y. Machida, S. Takizawa, H. Nakada, and S. Matsuoka, "Multi-replication with intelligent staging in data-Intensive grid

applications," *Proc. 7th IEEE/ACM Int'l Conference on Grid Computing*, pp. 88-95, 2006.

[21] L. Meyer, J. Annis, M. Wilde, M. Mattoso, and I. Foster, "Planning spatial workflows to optimize grid performance," *Proc. ACM Symp. Applied Computing*, pp. 786 - 790, 2006.

[22] K. Ranganathan and I. Foster, "Identifying dynamic replication strategies for a high-performance data grids," *Proc. 3rd Int'l Workshop on Grid Computing*, *Lecture Notes on Computer Science*, Vol. 2242, pp. 75-86, 2002.

[23] K. Ranganathan and I. Foster, "Computation scheduling and data replication algorithms for data Grids", *Grid resource management: state of the art and future trends*, pp. 359-373, 2004.

[24] H. Sato, S. Matsuoka, T. Endo, and N. Maruyama, "Access-pattern and bandwidth aware file replication algorithm in a grid environment," *Proc. 9th IEEE/ACM International Conference on Grid Computing*, pp. 250-257, 2008.

[25] H. Stockinger, A. Samar, K. Holtman, B. Allcock, I. Foster, and B. Tierney, "File and object replication in data Grids," *Proc. 10th IEEE Int'l Symp. High Performance Distributed Computing (HPDC)*, pp. 76-86, 2001.

[26] M. Tang, B.S. Lee, C.K. Yeo, and X. Tang, "Dynamic replication algorithms for the multi-tier data grid," *Future Generation Computer Systems*, Vol. 21, Issue 5, pp. 775–790, 2005.

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

[28] The European Data Grid Project, http://eu-datagrid.web.cern.ch/eu-datagrid.

[29] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A taxonomy of data grids for distributed data sharing, management, and processing," *ACM Computing Surveys*, Vol. 38, Issue 1, Article 3, 2006.