Data Replication in Data Intensive Scientific Applications with Performance Guarantee

Dharma Teja Nukarapu, *Student Member*, *IEEE*, Bin Tang, *Member*, *IEEE*, Liqiang Wang, *Member*, *IEEE*, and Shiyong Lu, *Senior Member*, *IEEE*

Abstract—Data replication has been well adopted in data intensive scientific applications to reduce data file transfer time and bandwidth consumption. However, the problem of data replication in Data Grids, an enabling technology for data intensive applications, has proven to be NP-hard and even non approximable, making this problem difficult to solve. Meanwhile, most of the previous research in this field is either theoretical investigation without practical consideration, or heuristics-based with little or no theoretical performance guarantee. In this paper, we propose a data replication algorithm that not only has a provable theoretical performance guarantee, but also can be implemented in a distributed and practical manner. Specifically, we design a polynomial time centralized replication algorithm that reduces the total data file access delay by at least half of that reduced by the optimal replication solution. Based on this centralized algorithm, we also design a distributed caching algorithm, which can be easily adopted in a distributed environment such as Data Grids. Extensive simulations are performed to validate the efficiency of our proposed algorithms. Using our own simulator, we show that our centralized replication algorithm performs comparably to the optimal algorithm and other intuitive heuristics under different network parameters. Using GridSim, a popular distributed Grid simulator, we demonstrate that the distributed caching technique significantly outperforms an existing popular file caching technique in Data Grids, and it is more scalable and adaptive to the dynamic change of file access patterns in Data Grids.

Index Terms—Data intensive applications, Data Grids, data replication, algorithm design and analysis, simulations.

1 Introduction

Data intensive scientific applications, which mainly aim to answer some of the most fundamental questions facing human beings, are becoming increasingly prevalent in a wide range of scientific and engineering research domains. Examples include human genome mapping [38], high-energy particle physics and astronomy [1], [24], and climate change modeling [30]. In such applications, large amounts of data sets are generated, accessed, and analyzed by scientists worldwide. The Data Grid [46], [4], [20] is an enabling technology for data intensive applications. It is composed of hundreds of geographically distributed computation, storage, and networking resources to facilitate data sharing and management in data intensive applications. One distinct feature of Data Grids is that they produce and manage very large amount of data sets, in the order of terabytes and petabytes. For example, the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN) near Geneva, Switzerland, is the largest scientific instrument on the planet. Since it began operation in August of 2008, it was expected to produce roughly 15 petabytes of data annually, which are accessed and analyzed by thousands of scientists around the world [2].

Replication is an effective mechanism to reduce file transfer time and bandwidth consumption in Data Grids—placing most accessed data at the right locations can greatly improve the performance of data access from a 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 huge amounts of data produced in scientific applications. With each Grid site having limited memory/storage resources,[1] the data replication problem becomes more challenging. We find that most of the previous work of data replication under limited storage capacity in Data Grids mainly occupies two extremes of a wide spectrum: they are either theoretical investigations without practical consideration, or heuristics-based experiments without performance guarantee (please refer to Section 2 for a comprehensive literature review). In this article, we endeavor to bridge this gap by proposing an algorithm that has a provable performance guarantee and can be implemented in a distributed manner as well.

Our model is as follows: Scientific data, in the form of data files, are produced and stored in the Grid sites as the result of scientific experiments, simulations, or computations. Each Grid site executes a sequence of scientific jobs submitted by its users. To execute each job, some scientific data as input files are usually needed. If these files are not in the local storage resource of the Grid site, they will be accessed from other sites, and transferred and replicated in the local storage of the site if necessary. Each Grid site can store such data files

- *D.T. Nukarapu and B. Tang are with the Department of Electrical Engineering and Computer Science, Wichita State University, Wichita, KS 67260. E-mail: dxnukarapu@wichita.edu, bintang@cs.wichita.edu.*
- *L. Wang is with the Computer Science Department, University of Wyoming, Laramie, WY 82071. E-mail: wang@cs.uwyo.edu.*
- *S. Lu is with the Computer Science Department, Wayne State University, Detroit, MI 48202. E-mail: shiyong@wayne.edu.*

Manuscript received 5 Aug. 2009; revised 4 June 2010; accepted 17 Nov. 2010; published online 2 Dec. 2010.
Recommended for acceptance by J. Weissman.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-08-0352.
Digital Object Identifier no. 10.1109/TPDS.2010.207.

1. In this article, we use memory and storage interchangeably.

subject to its storage/memory capacity limitation. We study how to replicate the data files onto Grid sites with limited storage space in order to minimize the overall file access time, for Grid sites to finish executing their jobs.

Specifically, we formulate this problem as a graph-theoretical problem and design a centralized greedy data replication algorithm, which provably gives the total data file access time reduction (compared to no replication) at least half of that obtained from the optimal replication algorithm. We also design a distributed caching technique based on the centralized replication algorithm,[2] and show experimentally that it can be easily adopted in a distributed environment such as Data Grids. The main idea of our distributed algorithm is that when there are multiple replicas of a data file existing in a Data Grid, each Grid site keeps track of (and thus fetches the data file from) its closest replica site. This can dramatically improve Data Grid performance because transferring large-sized files takes tremendous amount of time and bandwidth [16]. The central part of our distributed algorithm is a mechanism for each Grid site to accurately locate and maintain such closest replica site. Our distributed algorithm is also adaptive—each Grid site makes a file caching decision (i.e., replica creation and deletion) by observing the recent data access traffic going through it. Our simulation results show that our caching strategy adapts better to the dynamic change of user access behavior, compared to another existing caching technique in Data Grids [37].

Currently, for most Data Grid scientific applications, the massive amount of data is generated from a few centralized sites (such as CERN for high energy physics experiments) and accessed by all other participating institutions. We envision that, in the future, 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, generating data as a result of scientific experiments or simulations it performs, and meanwhile accessing data from other sites 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 an arbitrary pair of Grid sites. It is a great challenge in such an environment to efficiently share all the widely distributed and dynamically generated data files in terms of computing resources, bandwidth usage, and file transfer time. Our network model reflects such a 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 investigation without practical consideration, or heuristics-based implementation without a provable performance guarantee.
2. To the best of our knowledge, we are the first to propose data replication algorithm in Data Grid, which not only has a provable theoretical performance guarantee, but can be implemented in a distributed manner as well.
3. Via simulations, we show that our proposed replication strategies perform comparably with the optimal algorithm and significantly outperform an existing popular replication technique [37].
4. Via simulations, we show that our replication strategy adapts well to the dynamic access pattern change in Data Grids.

Paper Organization. The rest of the paper is organized as follows: We discuss the related work in Section 2. In Section 3, we present our Data Grid model and formulate the data replication problem. Section 4 and Section 5 propose the centralized and distributed algorithms, respectively. We present and analyze the simulation results in Section 6. Section 7 concludes the paper and points out some future work.

2 Related Work

Replication has been an active research topic for many years in World Wide Web [33], peer-to-peer networks [3], ad hoc and sensor networking [23], [41], and mesh networks [26]. In Data Grids, enormous scientific data and complex scientific applications call for new replication algorithms, which have attracted much research recently.

The most closely related work to ours is by Čibej et al. [44]. The authors study data replication on Data Grids as a static optimization problem. They show that this problem is NP-hard and nonapproximable, which means that 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. Moreover, Baev et al. [5], [6] show that if all the data have uniform size, then this problem is indeed approximable. And they find 20.5-approximation and 10-approximation algorithms. However, their approach, which is based on rounding an optimal solution to the linear programming relaxation of the problem, cannot be easily implemented in a distributed way. In this work, we follow the same direction (i.e., uniform data size), but design a polynomial time approximation algorithm, which can also be easily implemented in a distributed environment like Data Grids.

Raicu et al. [34], [35] study both theoretically and empirically the resource allocation in data intensive applications. They propose a "data diffusion" approach that acquires computing and storage resources dynamically, replicates data in response to demand, and schedules computations close to the data. They give a $O(NM)$ competitive ratio online algorithm, where N is the number of stores, each of which can store M objects of uniform size. However, their model does not allow for keeping multiple

2. We emphasize the difference between replication and caching in this article. Replication is a proactive and centralized approach wherein the server replicates files into the network to achieve certain global performance optimum, whereas caching is reactive and takes place on the client side to achieve certain local optimum, and it depends on the locally observed network traffic, job and file distribution, etc.

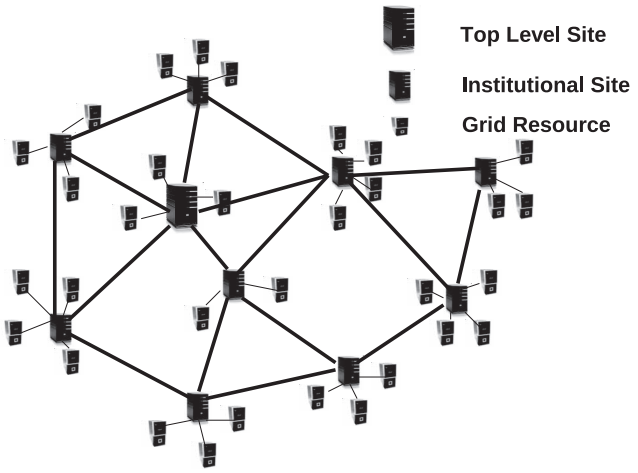


Fig. 1. Data Grid model.

copies of an object simultaneously in different stores. In our model, we assume each object can have multiple copies, each on a different site.

Some economical model-based replica schemes are proposed. The authors in [9], [7] use an auction protocol to make the replication decision and to trigger long-term optimization by using file access patterns. You et al. [47] propose utility-based replication strategies. Lei et al. [28] and Schintke and Reinefeld [39] address the data replication for availability in the face of unreliable components, which is different from our work.

Jiang and Zhang [25] propose a technique in Data Grids to measure how soon a file is to be reaccessed before being evicted compared with other files. They consider both the consumed disk space and disk cache size. Their approach can accurately rank the value of each file to be cached. Tang et al. [42] present a dynamic replication algorithm for multitier 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 multitier Data Grid. Chang and Chang [11] propose a dynamic data replication mechanism called Latest Access Largest Weight (LALW). 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 determine a popular file for replication. Park et al. [31] propose a dynamic replica replication strategy, called BHR, which benefits from "network-level locality" to reduce data access time by avoiding networking congestion in a Data Grid. Lamehamedi et al. [27] propose a lightweight Data Grid middleware, at the core of which are some dynamic data and replica location and placement techniques.

Ranganathan and Foster [37] present six different replication strategies: No Replication or Caching, Best Client, Cascading Replication, Plain caching, Caching plus Cascading Replication, and Fast Spread. All of these strategies are evaluated with three user access patterns (Random Access, Temporal Locality, and Geographical plus Temporal Locality). Via the simulation, the authors find that 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.

Systemwise, there are several real testbed and system implementations utilizing data replication. One system is the Data Replication Service (DRS) designed by Chervenak et al. [15]. 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. Another system is Physics Experimental Data Export (PheDEx) system [19]. 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 computing resource, schedules the job at sites where the needed input data files are stored, or transfers both the data and the 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 [45], which studies how to map jobs into Grid resources for execution, and it's coupling with data replication, as our future research. We are aware of very active research studying the relationship between these two [10], [36], [14], [21], [12], [43], [17], [8]. The focus of our paper, however, is on the data replication strategy with a provable performance guarantee.

As demonstrated by the experiments of Chervenak et al. [15], the time to execute a scientific job is mainly the time it takes to transfer the needed input files from server sites to local sites. Similar to other work in replica management for Data Grids [28], [39], [8], we only consider the file transfer time (access time), not the job execution time in the processor or any other internal storage processing or I/O time. Since the data are read only for many Data Grid applications [36], we do not consider consistency maintenance between the master file and the replica files. For readers who are interested in the consistency maintenance in Data Grids, please refer to [18], [40], [32].

3 Data Grid Model and Problem Formulation

We consider a Data Grid model as shown in Fig. 1. A Data Grid consists of a set of sites. There are *institutional sites*,

which correspond to different scientific institutions participating in the scientific project. There is one *top level site*, which is the centralized management entity in the entire Data Grid environment, and its major role is to mange the *Centralized Replica Catalogue* (CRC). CRC provides location information about each data file and its replicas, 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 computing resource, which allows users to submit and execute jobs, or a storage resource, which allows users to store data files.[3] We assume that each site has both computing and storage capacities, and that within each site, the bandwidth is high enough that the communication delay inside the site is negligible.

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

Data Grid model. A Data Grid can be represented as an undirected graph $G(V, E)$, where a set of vertices $V = \{1, 2, \ldots, n\}$ represents the sites in the Grid, and E is a set of weighted edges in the graph. The edge weight may represent a link metric such as loss rate, distance, delay, or transmission bandwidth. In this paper, the edge weight represents the bandwidth and we assume all edges have the same bandwidth B (in Section 6, we study heterogeneous environment where different edges have different bandwidths). There are p data files $D = \{D_1, D_2, \ldots, D_p\}$ in the Data Grid, D_j is originally produced and stored in the *source site* $S_j \in V$. Note that a site can be the original source site of multiple data files. The size of data file D_j is s_j. Each site i has a storage capacity of m_i (for a source site i, m_i is the available storage space after storing its original data).

Users of the Data Grid submit jobs to their own sites, and the jobs are executed in the FIFO order. Assume that the Grid site i has n_i submitted jobs $\{t_{i1}, t_{i2}, \ldots, t_{in_i}\}$, and each job t_{ik} $(1 \leq k \leq n_i)$ needs a subset F_{ik} of D as its input files for execution. If we use w_{ij} to denote the number of times that site i needs D_j as an input file, then $w_{ij} = \sum_{k=1}^{n_i} c_k$, where $c_k = 1$ if $D_j \in F_{ik}$ and $c_k = 0$ otherwise. The transmission time of sending data file D_j along any edge is s_j/B. We use d_{ij} to denote the number of transmissions to transmit a data file from site i and j (which is equal to the number of edges between these two sites). We do not consider the file propagation time along the edge, since compared with transmission time, it is quite small and thus negligible. The *total data file access cost* in Data Grid before replication is the total transmission time spent to get all needed data files for each site:

$$\sum_{i=1}^{n} \sum_{j=1}^{p} w_{ij} \times d_{iS_j} \times s_j/B.$$

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

Problem Formulation. The *data replication problem in the Data Grid* is to select a *set of sets* $M = \{A_1, A_2, \ldots, A_p\}$, where $A_j \subset V$ is a set of Grid sites that store a replica copy of D_j, to minimize the *total access cost* in the Data Grid:

$$\tau(G, M) = \sum_{i=1}^{n} \sum_{j=1}^{p} w_{ij} \times min_{k \in (\{S_j\} \cup A_j)} d_{ik} \times s_j/B,$$

under the storage capacity constraint that $|\{A_j | i \in A_j\}| \leq m_i$ for all $i \in V$, which means that each Grid site i appears in, at most, m_i sets of M. Here, each site accesses the data file from its closest site with a replica copy. Čibej, Slivnik, and Robič show that this problem (with arbitrary data file size) is NP-hard and even nonapproximable [44]. However, Baev et al. [5], [6] have shown that when data is uniform size (i.e., $s_i = s_j$ for any $D_i, D_j \in D$), this problem is approximable. Below we present a centralized greedy algorithm with provable performance guarantee for uniform data size, and a distributed algorithm respectively.

4 Centralized Data Replication Algorithm in Data Grids

Our centralized data replication algorithm is a greedy algorithm. First, all Grid sites have all empty storage space (except for sites that originally produce and store some files). Then, at each step, it places one data file into the storage space of one site such that the reduction of total access cost in the Data Grid is maximized at that step. The algorithm terminates when all storage space of the sites has been replicated with data files, or the total access cost cannot be reduced further. Below is the algorithm.

Algorithm 1. Centralized Data Replication Algorithm
BEGIN
  $M = A_1 = A_2 = \cdots = A_p = \emptyset$ (empty set);
  While (the total access cost can still be reduced by replicating data files in Data Grid)
    Among all sites with available storage capacity and all data files, let replicating data file D_i on site m gives the maximum $\tau(G, \{A_1, A_2, \ldots, A_i, \ldots, A_p\})$ $-\tau(G, \{A_1, A_2, \ldots, A_i \cup \{m\}, \ldots, A_p\})$;
    $A_i = A_i \cup \{m\}$;
  end while;
  RETURN $M = \{A_1, A_2, \ldots, A_p\}$;
END.

The total running time of the greedy algorithm of data replication is $O(p^2 n^3 \overline{m})$, where n is the number of sites in the Data Grid, $\overline{m}$ 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 $n\overline{m}$, and at each stage, we need to compute at most pn benefit values, where each benefit value computation may take $O(pn)$ time. Below we present two lemmas showing some properties of above greedy algorithm. They are also necessary for us to prove the theorem below, which shows that the greedy algorithm returns a solution with a near optimal total access cost reduction.

3. We differentiate site and node in this paper. We refer to each site in the Grid level and each node in the cluster level, where each node is a computing resource or storage resource or combination of both.

Lemma 1. *Total access cost reduction is independent of the order of the data file replication.*

Proof. This is because the reduction of total access cost depends only on the initial and final storage configuration of each site, which does not depend on the exact order of the data replication. □

Lemma 2. *Let M be a set of cache sites in an arbitrary stage. Let A_1 and A_2 be two distinct sets of sites not included in M. Then the access cost reduction due to selection of A_2 based upon cache site set $M \cup A_1$, denoted as a, is less than or equal to the access cost reduction due to selection of A_2 based upon M, denoted as b.*

Proof. Let $ClientSet(A_2, M)$ be the set of sites that have their closest cache sites in A_2 when A_2 are selected as cache sites upon M. With the selection of A_1 before A_2, some sites in $ClientSet(A_2, M)$ may find that they are closer to some cache sites in A_1 and then "deflect" to become elements of $ClientSet(A_1, M)$. Therefore, $ClientSet(A_2, M \cup A_1) \subseteq ClientSet(A_2, M)$.

Since the selection of A_1 before A_2 does not change the closest sites in M for all sites in $ClientSet(A_2, M \cup A_1)$, if $ClientSet(A_2, M \cup A_1) = ClientSet(A_2, M)$, $a = b$; if $ClientSet(A_2, M \cup A_1) \subset ClientSet(A_2, M)$, $a < b$. □

Above lemma says that the total access cost reduction due to an arbitrary cache site never increases with the selection of other cache site before it.

We present below a theorem that shows the performance guarantee of our centralized greedy algorithm. In the theorem and the following proof, we assume that all data files are of the same uniform-size (occupying unit memory). The proof technique used here is similar to that used in [41] for a closely related problem of data caching in ad hoc networks.

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.*

Proof. Let L be the total number of memory pages in the Data Grid. WLOG, we assume L is the total number of iterations of the greedy algorithm (note it is possible that the total access cost cannot be reduced further even though sites still have available memory space). We assume the sequence of selections in greedy algorithm is $\{n_1^g f_1^g, n_2^g f_2^g, \ldots, n_L^g f_L^g\}$, where $n_i^g f_i^g$ indicating that at iteration i, data file f_i^g is replicated at site n_i^g. We also assume the sequence of selections in optimal algorithm is $\{n_1^o f_1^o, n_2^o f_2^o, \ldots, n_L^o f_L^o\}$, where $n_i^o f_i^o$ indicating that at iteration i, data file f_i^o is replicated at site n_i^o. Let O and C be the total access cost reduction from optimal algorithm and greedy algorithm, respectively.

Next, we consider a modified data grid G', wherein each site i doubles its memory capacity (now $2m_i$). We construct a cache placement solution for G' such that for site i, the first m_i memories store the data files obtained in greedy algorithm and the second m_i memories store the data files selected in the optimal algorithm, as shown in Fig. 2. Now we calculate the total access cost reduction O' for G'. Obviously, $O' \geq O$, simply because each site in G' caches extra data files beyond the data files cached in the same site in G.

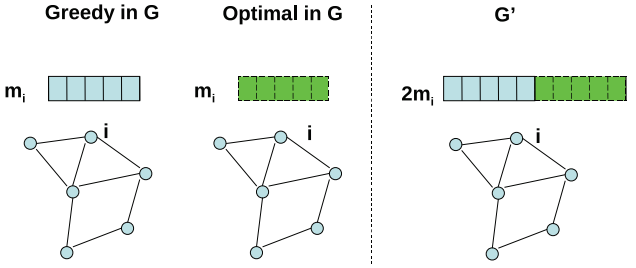


Fig. 2. Each site i original graph G has m_i memory space, each site in modified graph G' has $2m_i$ memory space.

According to Lemma 1, since the total access cost reduction is independent of the order of the each individual selection, we consider the sequence of the cache section in G' as $\{n_1^g f_1^g, n_2^g f_2^g, \ldots, n_L^g f_L^g, n_1^o f_1^o, n_2^o f_2^o, \ldots, n_L^o f_L^o\}$, that is, the greedy selection followed by optimal selection. Therefore, in G', the total access cost reduction is due to the greedy selection sequence plus the optimal selection sequence. For the greedy selection sequence, after the selection of $n_L^g f_L^g$, the access cost reduction is C. For the optimal selection sequence, we need to calculate the access cost reduction due to the addition of each $n_i^o f_i^o$ $(1 \leq i \leq L)$ based on the already added selections of $\{n_1^g f_1^g, n_2^g f_2^g, \ldots, n_L^g f_L^g, n_1^o f_1^o, n_2^o f_2^o, \ldots, n_{i-1}^o f_{i-1}^o\}$, and from Lemma 2, we know that it is less than the access cost reduction due to the addition of $n_i^o f_i^o$ based on already added sequence of $\{n_1^g f_1^g, n_2^g f_2^g, \ldots, n_{i-1}^g f_{i-1}^g\}$. Note that the latter is less than the access cost reduction due to the addition of $n_i^g f_i^g$, based on the same sequence of $\{n_1^g f_1^g, n_2^g f_2^g, \ldots, n_{i-1}^g f_{i-1}^g\}$. Thus, the sum of the access cost reduction due to selection of $\{n_1^o f_1^o, n_2^o f_2^o, \ldots, n_L^o f_L^o\}$ is less than or equal to C too.

Thus, we come to the conclusion that O is less than or equal to O', which is less than or equal to 2 times of C. □

5 Distributed Data Caching Algorithm in Data Grids

In this section, we design a localized distributed caching algorithm based on the centralized algorithm. In the distributed algorithm, each Grid site observes the local Data Grid traffic to make an intelligent caching decision. Our distributed caching algorithm is advantageous since it does not need global information such as the network topology of the Grid, and it is more reactive to network states such as the file distribution, user access pattern, and job distribution in the Data Grids. Therefore, our distributed algorithm can adopt well to such dynamic changes in the Data Grids.

The distributed algorithm is composed of two important components: nearest replica catalog (NRC) maintained at each site and a localized data caching algorithm running at each site. Again, as stated in Section 3, the top level site maintains a Centralized Replica Catalogue (CRC), which is essentially a list of *replica site list* C_j for each data file D_j. The replica site list C_j contains the set of sites (including source site S_j) that has a copy of D_j.

Nearest Replica Catalog (NRC). Each site i in the Grid maintains an NRC, and each entry in the NRC is of the form

(D_j, N_j), where N_j is the nearest site that has a replica of D_j. When a site executes a job, from its NRC, it determines the nearest replicate site for each of its input data files and goes to it directly to fetch the file (provided the input file is not in its local storage). As the initialization stage, the source sites send messages to the top level site informing it about their original data files. Thus, the centralized replica catalog initially records each data file and its source site. The top level site then broadcasts the replica catalogue to the entire Data Grid. Each Grid site 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 the *second nearest replica site*, i.e., the closest site (other than i itself) that has a copy of D_j. The second nearest replica site information is helpful when site i decides to remove the cached file D_j. If there is a cache miss, the request is redirected to the top level site, which sends the site replica site list for that data file. After receiving such information, the site will update correctly its NRC table and sends the request to the site's nearest cache site for that data file. Therefore, a cache miss takes much longer time. The above information is in addition to any information (such as routing tables) maintained by the underlying routing protocol in the Data Grids.

Maintenance of NRC. As stated above, when a site i caches a data file D_j, N_j is no longer the nearest replica site, but rather the second-nearest replica site. The site i sends a message to the top level site with information that it is a new replica site of D_j. When the top level site receives the message, it updates its CRC by adding site i to data file D_j's replica site list C_j. Then it broadcasts a message to the entire Data Grid containing the tuple (i, D_j) indicating the ID of the new replica site and the ID of the newly made replica file. Consider a site l that receives such message (i, D_j). Let (D_j, N_j) be the NRC entry at site l signifying that N_j is the replica site for D_j currently closest to l. If $d_{li} < d_{lN_j}$, then the NRC entry (D_j, N_j) is updated to (D_j, i). Note the distance values, in terms of number of hops, are available from the routing protocol.

When a site i removes a data file D_j from its local storage, its second-nearest replica site N_j becomes its nearest replica site (note the corresponding NRC entry does not change). In addition, site i sends a message to the top level site with information that it is no longer a replica site of D_j. The top level site updates its replica catalog by deleting i from D_j's replica site list. And then it broadcasts a message with the information (i, D_j, C_j) to the Data Grid, where C_j is the replica site list for D_j. Consider a site l that receives such a message, and let (D_j, N_j) be its NRC entry. If $N_j = i$, then site l updates its nearest replica site entry using C_j (with the help of the routing table).

Localized data caching algorithm. Since each site has limited storage capacity, a good data caching algorithm that runs distributedly on each site is needed. To do this, each site observes the data access traffic locally for a sufficiently long time. The *local access traffic* observed by site i includes its own local data requests, nonlocal data requests to data files cached at i, and the data request traffic that the site i is forwarding to other sites in the network.

Before we present the data caching algorithm, we give the following two definitions:

- **Reduction in access cost of caching a data file.** Reduction in access cost as the result of caching a data file at a site is the reduction in access cost given by the following: access frequency in local access traffic observed by the site $\times$ distance to the nearest replica site.
- **Increase in access cost of deleting a data file.** Increase in access cost as the result of deleting a data file at a site is the increase in access cost given by the following: access frequency in local access traffic observed by the site $\times$ distance to the second-nearest replica site.

For each data file D_j *not* cached at site i, site i calculates the reduction in access cost by caching the file D_j, while for each data file D_j cached at site i, site i computes the increase in access cost of deleting the file. With the help of the NRC, each site can compute such a reduction or increase of access cost in a distributed manner using only local information. Thus our algorithm is adaptive: each site makes a data caching or deletion decision by observing locally the most recent data access traffic in the Data Grid.

Cache replacement policy. With the above knowledge, a site always tries to cache data files that can fit in its local storage and that can give the most reduction in access cost. When the local storage capacity of a site 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 access cost reduction of caching a newly available data file D_j is higher than the access cost increase of some set D of cached data files where $|D| > |D_j|$, then the set D is replaced by D_j.

Data file consistency maintenance. In this work, we assume that all the data are read-only, and thus no data consistency mechanism is needed. However, if we do consider the consistency issue, two mechanisms can be considered. First, when the file in a site is modified, it can be considered as a data file removal in our distributed algorithm discussed above with some necessary deviation. That is, the site sends a message to the top level site with information that it is no longer a replica site of the file. The top level site broadcasts a message to the Data Grid so that each site can delete the corresponding nearest replica site entry. The second mechanism is Time to Live (TTL), which is the time until the replica copies are considered valid. Therefore, the master copy and its replica are considered outdated at the end of the TTL time value and will not be used for the job execution.

6 Performance Evaluation

Please refer to the Supplemental File, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2010.207, for the details of the performance evaluations.

7 Conclusions and Future Work

In this article, we study how to replicate data files in data intensive scientific applications, to reduce the file access time with the consideration of limited storage space of Grid sites. Our goal is to effectively reduce the access time of data files needed for job executions at Grid sites. We propose a centralized greedy algorithm with performance guarantee,

and show that it performs comparably with the optimal algorithm. We also propose a distributed algorithm wherein Grids sites react closely to the Grid status and make intelligent caching decisions. Using GridSim, a distributed Grid simulator, we demonstrate that the distributed replication technique significantly outperforms a popular existing replication technique, and it is more adaptive to the dynamic change of file access patterns in Data Grids.

We plan to further develop our work in the following directions:

- As ongoing and future work, we are exploiting the synergies between data replication and job scheduling to achieve better system performance. Data replication and job scheduling are two different but complementary functions in Data Grids: one to minimize the total file access cost (thus total job execution time of all sites), and the other to minimize the makespan (the maximum job completion time among all sites). Optimizing both objective functions in the same framework is a very difficult (if not unfeasible) task. There are two main challenges: first, how to formulate a problem that incorporates not only data replication but also job scheduling, and which addresses both total access cost and maximum access cost; and second, how to find an efficient algorithm that, if it cannot find optimal solutions of minimizing total/maximum access cost, gives near-optimal solution for both objectives. These two challenges remain largely unanswered in the current literature.
- We plan to design and develop data replication strategies in the scientific workflow [29] and large-scale cloud computing environments [22]. We will also pursue how provenance information [13], the derivation history of data files, can be exploited to improve the intelligence of data replication decision making.
- A more robust dynamic model and replication algorithm will be developed. Right now, each site observes the data access traffic for a sufficiently long time window, which is set in an ad hoc manner. In the future, a site should dynamically decide such an "observing window," depending on the traffic it is observing.

ACKNOWLEDGMENTS

Bin Tang's research has been supported in part by Kansas NSF EPSCoR Grant EPS-0903806.

REFERENCES

[1] The Large Hadron Collider, http://public.web.cern.ch/Public/en/LHC/LHC-en.html, 2011.

[2] Worldwide Lhc Computing Grid, http://lcg.web.cern.ch/LCG/, 2011.

[3] A. Aazami, S. Ghandeharizadeh, and T. Helmi, "Near Optimal Number of Replicas for Continuous Media in Ad-Hoc Networks of Wireless Devices," *Proc. Int'l Workshop Multimedia Information Systems,* 2004.

[4] 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," *Proc. IEEE Symp. Mass Storage Systems and Technologies,* 2001.

[5] I. Baev and R. Rajaraman, "Approximation Algorithms for Data Placement in Arbitrary Networks," *Proc. ACM-SIAM Symp. Discrete Algorithms (SODA),* 2001.

[6] I. Baev, R. Rajaraman, and C. Swamy, "Approximation Algorithms for Data Placement Problems," *SIAM J. Computing,* vol. 38, no. 4, pp. 1411-1429, 2008.

[7] 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," *Proc. Int'l Workshop Agent Based Cluster Computing and Grid (CCGrid),* 2003.

[8] D.G. Cameron, A.P. Millar, C. Nicholson, R. Carvajal-Schiaffino, K. Stockinger, and F. Zini, "Analysis of Scheduling and Replica Optimisation Strategies for Data Grids Using Optorsim," *J. Grid Computing,* vol. 2, no. 1, pp. 57-69, 2004.

[9] M. Carman, F. Zini, L. Serafini, and K. Stockinger, "Towards an Economy-Based Optimization of File Access and Replication on a Data Grid," *Proc. Int'l Workshop Agent Based Cluster Computing and Grid (CCGrid),* 2002.

[10] A. Chakrabarti and S. Sengupta, "Scalable and Distributed Mechanisms for Integrated Scheduling and Replication in Data Grids," *Proc. 10th Int'l Conf. Distributed Computing and Networking (ICDCN),* 2008.

[11] R.-S. Chang and H.-P. Chang, "A Dynamic Data Replication Strategy Using Access-Weight in Data Grids," *J. Supercomputing,* vol. 45, pp. 277-295, 2008.

[12] R.-S. Chang, J.-S. Chang, and S.-Y. Lin, "Job Scheduling and Data Replication on Data Grids," *Future Generation Computer Systems,* vol. 23, no. 7, pp. 846-860, Aug. 2007.

[13] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi, "Storing and Querying Scientific Workflow Provenance Metadata Using an Rdbms," *Proc. IEEE Int'l Conf. e-Science and Grid Computing,* 2007.

[14] 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," *Proc. IEEE/ACM Int'l Conf. Grid Computing,* 2007.

[15] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, and B. Moe, "Wide Area Data Replication for Scientific Collaboration," *Proc. IEEE/ACM Int'l Workshop Grid Computing,* 2005.

[16] A. Chervenak, R. Schuler, M. Ripeanu, M.A. Amer, S. Bharathi, I. Foster, and C. Kesselman, "The Globus Replica Location Service: Design and Experience," *IEEE Trans. Parallel and Distributed Systems,* vol. 20, no. 9, pp. 1260-1272, Sept. 2009.

[17] N.N. Dang and S.B. Lim, "Combination of Replication and Scheduling in Data Grids," *Int'l J. Computer Science and Network Security,* vol. 7, no. 3, pp. 304-308, Mar. 2007.

[18] D. Düllmann and B. Segal, "Models for Replica Synchronisation and Consistency in a Data Grid," *Proc. 10th IEEE Int'l Symp. High Performance Distributed Computing (HPDC),* 2001.

[19] J. Rehn et al., "Phedex: High-Throughput Data Transfer Management System," *Proc. Computing in High Energy and Nuclear Physics (CHEP),* 2006.

[20] I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today,* vol. 55, pp. 42-47, 2002.

[21] I. Foster and K. Ranganathan, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications," *Proc. 11th IEEE Int'l Symp. High Performance Distributed Computing (HPDC),* 2002.

[22] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degrees Compared," *Proc. Grid Computing Environments Workshop,* pp. 1-10, 2008.

[23] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," *Proc. ACM MobiCom,* 2000.

[24] 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," *Proc. Earth Science Technology Conf.,* 2004.

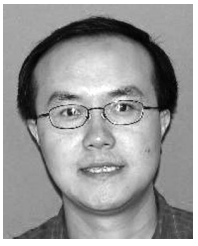
[25] S. Jiang and X. Zhang, "Efficient Distributed Disk Caching in Data Grid Management," *Proc. IEEE Int'l Conf. Cluster Computing,* 2003.

[26] S. Jin and L. Wang, "Content and Service Replication Strategies in Multi-Hop Wireless Mesh Networks," *Proc. ACM Int'l Conf. Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM),* 2005.

[27] H. Lamehamedi, B.K. Szymanski, and B. Conte, "Distributed Data Management Services for Dynamic Data Grids," unpublished.

[28] M. Lei, S.V. Vrbsky, and X. Hong, "An Online Replication Strategy to Increase Availability in Data Grids," *Future Generation Computer Systems,* vol. 24, pp. 85-98, 2008.
[29] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, and J. Hua, "A Reference Architecture for Scientific Workflow Management Systems and the View Soa Solution," *IEEE Trans. Services Computing,* vol. 2, no. 1, pp. 79-92, Jan.-Mar. 2009.
[30] 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,* vol. 18, no. 4, pp. 379-391, 2003.
[31] S.M. Park, J.H. Kim, Y.B. Lo, and W.S. Yoon, "Dynamic Data Grid Replication Strategy Based on Internet Hierarchy," *Proc. Second Int'l Workshop Grid and Cooperative Computing (GCC),* 2003.
[32] J. Pérez, F. García-Carballeira, J. Carretero, A. Calderón, and J. Fernández, "Branch Replication Scheme: A New Model for Data Replication in Large Scale Data Grids," *Future Generation Computer Systems,* vol. 26, no. 1, pp. 12-20, 2010.
[33] L. Qiu, V.N. Padmanabhan, and G.M. Voelker, "On the Placement of Web Server Replicas," *Proc. IEEE INFOCOM,* 2001.
[34] I. Raicu, I. Foster, Y. Zhao, P. Little, C. Moretti, A. Chaudhary, and D. Thain, "The Quest for Scalable Support of Data Intensive Workloads in Distributed Systems," *Proc. ACM Int'l Symp. High Performance Distributed Computing (HPDC),* 2009.
[35] I. Raicu, Y. Zhao, I. Foster, and A. Szalay, "Accelerating Large-Scale Data Exploration through Data Diffusion," *Proc. Int'l Workshop Data-Aware Distributed Computing (DADC),* 2008.
[36] 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," *Proc. Seventh IEEE Int'l Symp. Cluster Computing and the Grid (CCGRID),* 2007.
[37] K. Ranganathan and I.T. Foster, "Identifying Dynamic Replication Strategies for a High-Performance Data Grid," *Proc. Second Int'l Workshop Grid Computing (GRID),* 2001.
[38] A. Rodriguez, D. Sulakhe, E. Marland, N. Nefedova, M. Wilde, and N. Maltsev, "Grid Enabled Server for High-Throughput Analysis of Genomes," *Proc. Workshop Case Studies on Grid Applications,* 2004.
[39] F. Schintke and A. Reinefeld, "Modeling Replica Availability in Large Data Grids," *J. Grid Computing,* vol. 2, no. 1, pp. 219-227, 2003.
[40] 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),* 2001.
[41] B. Tang, S.R. Das, and H. Gupta, "Benefit-Based Data Caching in Ad Hoc Networks," *IEEE Trans. Mobile Computing,* vol. 7, no. 3, pp. 289-304, Mar. 2008.
[42] 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, pp. 775-790, 2005.
[43] 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,* vol. 22, pp. 254-268, 2006.
[44] U. Čibej, B. Slivnik, and B. Robič, "The Complexity of Static Data Replication in Data Grids," *Parallel Computing,* vol. 31, nos. 8/9, pp. 900-912, 2005.
[45] S. Venugopal and R. Buyya, "An Scp-Based Heuristic Approach for Scheduling Distributed Data-Intensive Applications on Global Grids," *J. Parallel and Distributed Computing,* vol. 68, pp. 471-487, 2008.
[46] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing," *ACM Computing Surveys,* vol. 38, no. 1, 2006.
[47] X. You, G. Chang, X. Chen, C. Tian, and C. Zhu, "Utility-Based Replication Strategies in Data Grids," *Proc. Fifth Int'l Conf. Grid and Cooperative Computing (GCC),* 2006.

Dharma Teja Nukarapu received the BE degree in information technology from Jawaharlal Nehru Technological University, India, in 2007, the MS degree from the Department of Electrical Engineering and Computer Science, Wichita State University in 2009. He is currently working toward the PhD degree in the Department of Electrical Engineering and Computer Science at Wichita State University. His research interests include data caching and replication and job scheduling in data intensive scientific applications. He is a student member of the IEEE.



Bin Tang received the BS degree in physics from Peking University, China, in 1997, the MS degrees in materials science and computer science from Stony Brook University in 2000 and 2002, respectively, and the PhD degree in computer science from Stony Brook University in 2007. He is currently an assistant professor in the Department of Electrical Engineering and Computer Science at Wichita State University. His research interests include algorithmic aspect of data intensive sensor networks. He is a member of the IEEE.

Liqiang Wang is currently an assistant professor in the Department of Computer Science at the University of Wyoming. He received the BS degree in mathematics from Hebei Normal University, China, in 1995, the MS degree in computer science from Sichuan University, China, in 1998, and the PhD degree in computer science from Stony Brook University in 2006. His research interests include the design and analysis of parallel computing systems. He is a member of the IEEE.

Shiyong Lu received the PhD degree in computer science from the State University of New York at Stony Brook in 2002, the ME degree from the Institute of Computing Technology of Chinese Academy of Sciences at Beijing in 1996, and the BE degree from the University of Science and Technology of China at Hefei in 1993. He is currently an associate professor in the Department of Computer Science, Wayne State University, and the director of the Scientific Workflow Research Laboratory (SWR Lab). His research interests include scientific workflows and databases. He has published more than 90 papers in refereed international journals and conference proceedings. He is the founder and currently a program cochair of the IEEE International Workshop on Scientific Workflows (2007 2010), an editorial board member for *International Journal of Semantic Web and Information Systems* and *International Journal of Healthcare Information Systems and Informatics*. He is a senior member of the IEEE.