Performance of File Replication Policies for Real-time File Access in Data Grids

Atakan Doğan
Anadolu University
Dept. of Electrical and Electronics
Engineering, 26470 Eskisehir Turkey
+ (90) (222) 335 0580
atdogan@anadolu.edu.tr

ABSTRACT
A variety of file replication and replacement algorithms were proposed for the Data Grids in the literature. On the other hand, real-time Grid applications are emerging in many disciplines of science and engineering. Thus, it is natural to ask about the real-time performance of these file replication and replacement algorithms. Based on this motivation, in this study, the performance of three replication algorithms, namely fast spread, cascading, and caching, are evaluated together with three different file replacement policies, including least recently used, least frequently used, and least urgent first, under various Grid settings. For this evaluation, a process oriented and discrete-event driven simulator is developed. A detailed set of simulation studies are conducted using the simulator and the results obtained are presented to elaborate on the real-time performance of the aforementioned replication algorithms.

Keywords
Data Grids, File Replication, File Replacement, Simulation

1. INTRODUCTION
Data Grids, which are composed of geographically distributed storage, computing, and networking resources, are envisaged to run a variety of scientific simulation and experiment applications [1]. Common to such applications is that they involve a large number of data-intensive jobs and require the efficient management and transfer of terabytes and petabytes of information [1], [2]. A major problem identified for the peta-scale data intensive computing is how to schedule jobs and their related data in an effort to minimize jobs' completion times and bandwidth/storage space consumed due to the file transfers [3]-[7]. In order to alleviate these problems, data are dynamically replicated on multiple storage systems guided by a replication algorithm [3]-[11].

Conference name: GridNets 2007, October 17-19, 2007, Lyon, France

DOI 10.4108/gridnets.2007.2235

Data Grids are also considered for running applications with real-time requirements. A few examples of real-time Grid applications can be found in [12]-[16]. A common characteristic of such applications is that data produced or stored in one component of the system need to be transferred across a network of limited resources to another component (or components) while respecting associated real-time attributes. Achieving such data transfers in a timely manner and servicing as many data transfer requests as possible in a distributed environment is a nontrivial problem known as the *real-time data dissemination problem* [17]-[19]. The real-time data dissemination algorithms in [17]-[19] can be grouped into two categories based on how many data transfer requests are scheduled in each iteration. In the first category, *Partial Path Heuristic* (PPH), *Full Path Heuristic* (FPH), *Full Path All Destinations Heuristic* (FPA) [17] and *Extended Partial Path Heuristic* (EPP) [18] schedule only one request per iteration. In the second one, *Concurrent Scheduling Heuristic* (CSH) [18] and *Blocking Analysis Concurrent Scheduling* (BACS) [19] try to transfer data related to a set of requests in parallel. However, the studies in [17]-[19] assumed a specific file replication algorithm and they did not study the performance of these real-time data scheduling algorithms under different data (file) replication algorithms. Furthermore, they evaluated the algorithms' performances under a naïve (static) data replacement policy and uniformly distributed data access pattern only. Recently, in [20], the performance of both FPH and EPP were investigated for a dynamic data replacement policy and a variety of data access patterns.

The motivation of this study stems from the following observation. The studies in [3]-[11] assessed the value of a variety of file replication/replacement strategies under different data access patterns. Because the scope of these studies did not include real-time issues they did not evaluate the performance of the proposed algorithms therein under time constraints. However, the real-time performance of Grid could be essential for some Grid applications as exemplified above. Thus, this preliminary study tries to elaborate on the real-time performance of the file replication algorithms, namely fast spread, cascading, and caching, together with three file replacement heuristics, which are least recently used, least frequently used, and least urgent first, under different Grid conditions.

2. A MULTI-TIER DATA GRID MODEL

A *Data Grid* is composed of a set of machines **M**= {M_1, M_2, ..., M_m} and links **L**= {L_1, L_2, …, L_n} each of which provides a bidirectional connection between two machines. Each machine M_i has a limited storage capacity C_i and each link L_j is associated with a specific bandwidth of B_j. The machines in the Data Grid are assumed to be organized in tiers, which results in a *multi-tier computing model* inspired by the Large Hadron Collider (LHC) computing model [21], [22]. In this study, the Data Grid is modeled to have four tiers. Tier 0 is the source where all data files are produced and initially stored. Connected to the Tier 0 machine is the Tier 1 machines which correspond to a few national centers around a country. Each Tier 1 machine has a number of related Tier 2 machines each of which models a workgroup at a university or research laboratory. Finally, each Tier 2 machine is associated with a set of Tier 3 workstations.

In the tiered computing model adopted, file transfer requests can come from only the Tier 3 machines which provide the computing power of the Grid. On the other hand, the Tier 1 and Tier 2 machines play the role of the intermediate replica servers on which the files can be replicated during their transfer to the Tier 3 machines. For the tiered computing model, the *store-and-forward* is adopted as the data transfer scheme, which requires that a machine receive the file completely before it starts sending it to the next machine along the path to the destination. As a result, when a file is transferred from the Tier 0 to a Tier 3 machine, both Tier 1 and Tier 2 machines along the path also receive the file in addition to the destination machine.

The Data Grid is used for running real-time distributed applications which generate a set of requests **R**= {R_1, R_2, …, R_r} for data files **X**, where **X**= {X_1, X_2, …, X_q} denotes the set of q unique data files. Each request R_k is associated with one of q files X_k to be transferred from a source machine to a destination machine M_k, a deadline value D_k by which file X_k must be delivered to its destination M_k. Thus, request R_k is summarized by the following tuple: {X_k, M_k, D_k}.

In order to facilitate dynamic file replication in the multi-tier Data Grid, following structures are assumed to be in place in the system: *Replica Catalog (RC)*, *Replica Manager (RM)*, and *Local Replica Manager (LRM)*. The RC is a centralized database which stores the mappings from the logical file names to the physical file names in the Grid. The logical file names (**X**) are unique identifiers for the data files in the Grid, whereas the physical file names represent their physical locations. When the RC is queried by the RM using a logical file name (X_k) the RC should respond to the RM with one or multiple available physical file names corresponding to the logical file name. In addition to responding to the RM's queries, the RC also receives the logical file name update messages from the LRMs and consolidates them into the database.

The RM is a centralized service with the following functions. The RM accepts the file transfer requests from the LRMs of the Tier 3 machines. Upon receiving a request R_k, it should query the RC and obtain a set of physical file names related to the logical file name X_k. For each physical file name returned by the RC, the RM evaluates the transmission time of the replica to destination M_k using the predicted performance of the network. Then, it find outs the replica with the minimum transmission time and the computes the request's urgency, denoted by U_k. The urgency can be computed as $U_k = D_k - t_k^{RM} - t_{Xfer}$, where t_k^{RM} is the RM's current wall-clock time and t_{Xfer} is the estimated file transfer time. If the request's urgency is equal to or less than zero, the RM drops the request since it is impossible to meet its deadline under the current network conditions. Otherwise, the RM notifies the corresponding LRM about the file transfer request (R_k) via a request message. In order to support the store-and-forward model for the real-time file transfers, the RM includes the urgency of the request (U_k) and the time of the urgency computation (t_k^{RM}) into the request message.

The LRM runs on all machines in all tiers and manages the storage resource of the machine on which it is deployed. The LRM is responsible for the following activities: (1) honoring a file transfer request, (2) forwarding a file to the next machine along the path, (3) keeping its *Local Replica Catalog (LRC)* up-to-date and publishing the LRC's content to the RC, (4) executing a particular file replacement algorithm. The way an LRM honors a file transfer request depends on where it is deployed. If the LRM is deployed on a Tier 3 machine, upon receiving a request, it checks its *Local Replica Catalog (LRC)* to see if it owns the requested file, where the LRC holds the logical file names on this machine. If the LRM owns it, it provides the file to the requestor job on the same machine. Otherwise, the Tier 3 LRM sends a request to the RM for the remote file access. If the LRM runs on a Tier 0, Tier 1, or Tier 2 machine, it will receive the requests from the RM, not from a job on the same machine. After receiving the request message from the RM, the LRM starts the transmission of the requested file to the destination machine only if the urgency of the request is equal to or greater than zero. The LRM can compute the current urgency of the request as $U_k - (t_k^{LRM} - t_k^{RM})$, where t_k^{LRM} is the LRM's current wall-clock time.

Another responsibility of a Tier 1 and Tier 2 LRM is to forward a file to the next machine along the request's path after the file is completely written into its local storage. Similar to honoring a request, the LRM must compute the request's urgency and process the request accordingly. The third function of the LRM is that it must keep its LRC up-to-date by updating the LRC whenever a new file is written into the local storage. Following the update, the LRM must publish the content of its LRC to the RC. Finally, the LRM must implement a file replacement policy to choose one or more files to delete when a new file needs to be copied into the local storage facility without enough available storage space.

3. FILE REPLICATION ALGORITHMS

In the literature, a variety of dynamic file replication algorithms has been proposed for replicating files without real-time requirements in Data Grids. In this preliminary study, among these replication algorithms, three basic algorithms are chosen for the evaluation of the impact of the file replication on the real-time data distribution in a multi-tier Grid environment. All three replication algorithms are taken from [8].

Caching: The replica manager forwards all file transfer requests to the LRM of the Tier 0 machine. As a result, the Tier 1 and Tier 2 machines do not function as replica servers and their LRMs are not required to publish their LRCs. Once a Tier 3 machine receives the file it has requested, it keeps the file in its local storage for a possible future reference until it gets replaced.

Cascading: In order to support the cascading replication algorithm, LRMs also need to keep a history for each file they have, where the history information includes the number of requests for that file and the machines each request came from. After an LRM honors a request from the RM it updates the file history information. During the history update, if the LRM finds out that the number of requests for the file has reached to a preset threshold value, it determines the best client for the file and forwards it to the next machine (one tier below) along the path to the best client machine. The best client is a Tier 3 machine which has generated the most requests for that file. With the completion of the forwarding operation, the LRM clears the history information of the file and starts over logging again. Furthermore, the LRM of the next machine one tier below updates its LRC and publishes the new content, which will make this machine an alternative source for the file after the RC is updated accordingly. Once the requestor Tier 3 machine receives the file, it caches the file until it gets replaced. As a result, the cascading replication is implemented in a distributed fashion by the LRMs deployed on the Grid machines. Note that the replica manager initially sends all file transfer requests to the LRM of the Tier 0 machine. Furthermore, any regular forwarding of a file due to the store-and-forward scheme requires neither an intermediate source LRM to update its file history nor an intermediate destination LRM publish its LRC after the completion of the forwarding.

Fast Spread: This replication algorithm is naturally supported by the adopted data transfer scheme. During the transmission of a file, any LRM on a Tier 1 or Tier 2 machine along the path to the destination machine needs to update its LRC and to publish it to the RC after the file is written into the local storage. As a result, the Tier 1 or Tier 2 machine will be considered as possible source machine for that file once the RC receives the LRC update message.

In order to complement these file replication algorithms, a particular file replacement policy needs to be enforced by the LRMs. In this study, three different file replacement policies are considered. These policies are *Least Recently Used* (LRU), *Least Frequently Used* (LFU), and *Least Urgent First* (LUF). The first two policies are commonly used in the literature and they are shown to be as effective as more advanced economic model based file replacement policies [5]. The latter is proposed in this study.

Least Recently Used: LRM keeps a history for each file in its LRC indicating the latest time at which that file is accessed. When the storage space available is not enough to hold a new file, LRM sorts the files in the increasing order of their latest access times into a list and then deletes the files starting from the file at the top of the list until it frees up large enough storage space for the new file.

Least Frequently Used: In this algorithm, the history information for each file is the number of requests for that file. Thus, LRM sorts the files in the increasing order of their number of requests into a list and then deletes the files starting from the file at the top of the list until large enough storage space for the new file is obtained.

Least Urgent First: The history information needed to realize this algorithm is the average urgency of the file when LRM receives the request message from the RM. As a result, LRM finds out the least urgent files on average and removes them from the storage to open the necessary space for the new file.

4. SIMULATIONS

A simulator was developed to investigate the performance of the file replication strategies, Caching, Cascading, and Fast Spread together with the file replacement policies, LRU, LFU, and LUF. The simulator was written in C programming language using the CSIM 19 library [23]. The CSIM library allows the development of process-oriented discrete-event simulation programs. A CSIM program models a system as a collection of CSIM processes which interact with each other by using the CSIM structures. Following the CSIM programming style, the simulator developed models the file transfer of each request as a process and uses events to signal the end of file transfers. Furthermore, the network links and storage devices are modeled as facilities, a CSIM structure, which can be used and later released by the processes.

The simulator consists of three parts. The network component is used to create LHC-like tiered computing systems. The next component of the simulator is the request generator which produces real-time file transfer requests. The third component is the heuristic which handles the simulation of the file replication algorithms and file replacement policies.

4.1 Network

With the start of the simulation, a LHC-like tiered computing system is created. As in [8] and [10], there are four tiers. The Tier 0, 1, 2, and 3 are assumed to include 1, 5, 25, and 125 machines, respectively, which is a total of 156 machines (85 in [8] and 156 in [10]). These machines are interconnected by a randomly generated tree topology.

In the tree topology generated, each machine except those in the Tier 3 has at least one child machine in the lower tier. The bandwidth of a link connecting two machines in the different tiers is assumed as follows: 2.5 Gbit/s for a link between Tier 0 and Tier 1, 1 Gbit/s between Tier 1 and Tier 2, and 622 Mbit/s between Tier 2 and Tier 3.

In the simulated Grid, the storage capacity of each machine is determined based on the *relative capacity* (R) index, which is the ratio between the total storage capacity of Tier 1 and Tier 2 machines and the total size of all data files. It should be noted that a higher value of R will have an effect of allowing more file replicas in the Grid. Thus, the performance of a particular file replication algorithm depends on the value of R. In the simulations, three different values for the relative capacity, which are 76.8%, 38.4%, and 19.2%, are assumed and the corresponding storage capacities are determined as shown Table 1.

Table 1. Storage capacity configurations for Tier 1 and Tier 2 machines.

Case	Tier 1		Tier 2		Overall
	T_1 (TB)	R_1 (%)	T_2 (TB)	R_2 (%)	R (%)
1	1×5 = 5	51.2	0.1×25 = 2.5	25.6	76.8
2	0.5×5 = 2.5	25.6	0.05×25 = 1.25	12.8	38.4
3	0.25×5 = 1.25	12.8	0.025×25 = 0.625	6.4	19.2

In Table 1, the relative capacity of the simulated Grid is computed as follows. The number of data files in a Grid will be in the order of millions. However, it is not feasible to simulate the transfer of such a large number of files on a single computer. Thus, the number of files is scaled down and it is assumed to be 5000 during the simulations, each of which is 2 GB. Fixed file size is also common in other studies, e.g., [4], [8], [10]. As a result, the total size of all files is about 9.75 TB. At the beginning of the simulation, all original data files are stored in the Tier 0 machine with unlimited storage capacity and all other machines do not store any files. Once the requests start coming into the system, Tier 1 and Tier 2 machines are used as the replica servers on which the original files can be replicated following a replication algorithm. Taking case 2 as an example, the total storage capacity of Tier 1 machines, T_1, is 2.5 TB, where Tier 1 has 5 machines each of which has a storage capacity of 0.5 TB; the total storage capacity of Tier 2 machines, T_2, is 1.25 TB, where Tier 2 includes 25 machines, each with a storage capacity of 0.05 TB. Thus, the total replication capacity, T, is $T_1 + T_2 = 3.75$ TB and the relative capacity R becomes 3.75/9.75= 38.4%. Finally, each Tier 3 machine has 2 GB of storage capacity which is enough to hold only one file.

4.2 Request Generator

The request generator component is implemented as a CSIM process. During the simulations, the requests are assumed to come in to the Grid according to a Poisson process with a specific arrival rate. Furthermore, they are submitted only from the Tier 3 machines [4], [8], [10]. For each submitted request, three different parameters are associated with it. First, the request is associated with a data file available in the Grid according to a particular file access pattern. However, as of now, no actual file access patterns for the Grid applications are known. Thus, three commonly used file access patterns [3]-[11], namely random, geometric, and Zipf, are implemented in the simulator. Specifically, the geometric distribution with the file popularity parameter of 0.05 and the Zipf distribution with the file popularity parameter of 0.8 are assumed. Second, the request is accompanied with a randomly chosen Tier 3 destination machine. Finally, a deadline value is assigned to the request using the formula: $D_k = A_k + \text{rand}(0, \lambda) \times |X_k|/B_{2\text{-}3}$, where A_k is the arrival time of the request, rand (0, λ) is a function which returns a uniformly distributed real number between 0 and λ, λ denotes the deadline factor, $|X_k|$ is the size of file X_k, $B_{2\text{-}3}$ represents the link bandwidth between Tier 2 and Tier 3 machines.

4.3 Heuristics

Three file replication algorithms and three file replacement policies are implemented in another CSIM process, called as file_transfer. For each file request submitted, one file_transfer process is created to take care of the transfer of the file. During the transfer, the process implements the store-and-forward style of communication between source and destination as the core component. On top of this core, the file replication and replacement algorithms are developed.

In the implementation of the store-and-forward by file_transfer, it is assumed that the network links can be used to transfer only one file at a time. That is, if more than one file needs to be transferred on the same link, these files are serviced in FIFO order on the link. An immediate result of this assumption is that the number of simultaneous file transfer operations (read or write) for a machine cannot be greater than the number of point-to-point connections that machine has. In the simulator developed, each machine is allowed to have simultaneous file transfer operations as many as its point-to-point connections.

4.4 Performance Results

Using the simulator developed, a set of simulation studies were conducted. First, a base set of results was established. In the base set, the fast spread and cascading with LRU, LFU, and LUF and caching replication algorithms are evaluated in terms of the percentage of the satisfied requests for the random, geometric, and Zipf file access patterns under the following simulation parameters: number of data files = 5000, number of requests= 50000, deadline factor= 10, and arrival rate of requests= 1 request/second. Later, each of these simulation parameters is individually varied to study the impact of the parameter on the performance of the algorithms. The results of the simulation studies are presented in Tables 2-6, where each data shown is the average of 10 simulation runs. Note that each iteration of the simulation creates a different Grid topology and request set under the given simulation parameters.

Table 2 shows the base set of results. As far as the performance is concerned, the fast spread is always the best, followed by the cascading, and caching. Note that since the caching algorithm replicates files only on the clients, it is not affected by the changes in the relative capacity. When the fast spread and caching are compared, the caching shows a close performance to the fast spread for the random file access pattern only. The reason for the close performance is due to the fact that accessing files in random fashion decreases the time of stay for files in the replica servers, which makes the file replication less effective for the clients. For the geometric access pattern, the fast spread outperforms the caching in the range of 40% and %54; for the Zipf, the performance difference is between 14% and 29%. When the performance of the cascading and caching are compared, the performance improvement of the cascading is between 35% and 47% for the geometric and between 4% and 12.5% for the Zipf file access patterns, respectively. For the random access pattern, the caching is a bit better than the cascading. This can be attributed to two factors. The first one is as explained above. The second factor stems from the nature of the cascading. That is, the number of file transfers under the cascading is more than the number of file requests (50000). This excess number of the file transfers is used for replicating files to machines on the paths to the best client machines, which increases the network load and makes more requests to be unsatisfied. Finally, when the fast spread and cascading are compared, the fast spread always outperforms the cascading with the following margins: 4%-11% for the random, 4%-%5 for the geometric, and 8%-14% for the Zipf file access pattern. The additional network load created by the cascading as well as the aggressive replications performed by the fast spread have created this performance difference.

According to Table 2, as expected, decreasing the relative capacity has adverse impact on the performance of both fast spread and cascading. In terms of the effect of the file access pattern on the performance, all three algorithms show their best performance for the geometric access pattern, followed by the Zipf and random. Under the geometric and Zipf access patterns, several files are requested more as compared to the others. This makes it more probable to find these popular files on the replica

Table 2. The performance of the fast spread and cascading with LRU, LFU, and LUF and caching under the base simulation parameters.

		Base								
		Random			Geometric			Zipf		
		76.8%	38.4%	19.2%	76.8%	38.4%	19.2%	76.8%	38.4%	19.2%
Fast Spread	LRU	60	57	55	88	86	82	72	69	67
	LFU	58	56	55	87	83	80	69	67	64
	LUF	60	57	55	86	84	81	71	68	66
Cascading	LRU	54	53	53	84	83	79	63	63	62
	LFU	54	53	53	84	80	77	63	62	60
	LUF	54	53	53	84	80	77	63	61	58
Caching		55			57			56		

Table 3. The performance of the fast spread and cascading with LRU, LFU, and LUF and caching for two different values of the number of files.

		No of files= 5000 (Base)			No of files= 10000		
		Random	Geometric	Zipf	Random	Geometric	Zipf
Fast Spread	LRU	55	82	67	56	82	63
	LFU	55	80	64	55	80	61
	LUF	55	81	66	55	80	63
Cascading	LRU	53	79	62	54	79	61
	LFU	53	77	60	54	77	60
	LUF	53	77	58	54	77	58
Caching		55	57	56	54	58	56

Table 4. The performance of the fast spread and cascading with LRU, LFU, and LUF and caching for two different values of the number of requests.

		No of requests= 50000 (Base)			No of requests= 75000		
		Random	Geometric	Zipf	Random	Geometric	Zipf
Fast Spread	LRU	55	82	67	55	82	64
	LFU	55	80	64	54	80	64
	LUF	55	81	66	55	81	66
Cascading	LRU	53	79	62	50	79	64
	LFU	53	77	60	50	75	61
	LUF	53	77	58	50	78	60
Caching		55	57	56	54	57	58

servers, which results in more requests to be satisfied. The worse performance under the Zipf file access pattern can be due to the long-tail of the Zipf distribution. That is, the number of different files requested under the Zipf is more as compared to the geometric distribution and this adversely affects the replication performance. Finally, in terms of the file replacement policy of the choice, all three policies have lead to quite similar performance values. More specifically, the LRU is followed by the LUF and then LFU.

Table 3 compares the performance of the algorithms when the number of files is increased from 5000 to 10000 while keeping the other simulation parameters unaltered and the relative capacity at 19.2%. The only noticeable performance difference as compared to the base case is observed for the fast spread under the Zipf file replacement policy, which is not very significant.

Table 4 presents the performance of the algorithms when the number of requests is increased from 50000 to 75000 while keeping the other simulation parameters fixed. This simulation study is conducted to see how the algorithms react when the request load in the Grid is increased. All three algorithms have performed a stable performance under heavier load conditions. Based on the results in Table 3 and 4, it can be concluded that the algorithms studied show scalable performances.

Table 5 shows the performance of the algorithms when the deadline factor is increased from 10 to 20 while keeping the other simulation parameters unchanged. Note that increasing the deadline factor leads to relaxing the request deadlines. Thus, it is expected that all three algorithms perform better when $\lambda = 20$, which is really the case in Table 5.

Table 6 shows the performance of the algorithms when the request arrival rate is increased from 1 request/sec to 2 requests/sec while keeping the other simulation parameters fixed. Increasing the request arrival rate is expected to lead to more resource contention among the requests. Consequently, the less number of requests will be satisfied. According to Table 6, all three algorithms have been significantly affected and they have experienced sharp decreases in their performances. For the random and Zipf file access patterns, the decrease in the performance is more profound; for the geometric access pattern, it is more tolerable.

5. CONCLUSIONS

From the results presented in the previous section, it is evident that the file replication algorithm chosen and the file access pattern of the tasks running on the Grid have significant impact on the real-time Grid performance. Among the simulated algorithms, the fast spread has shown superior performance to the cascading and caching and the caching being the worst of the three. In addition to the file replication algorithms, three different file replacement policies are accounted for the fast spread and cascading. Among these policies, the LRU has constantly improved the performance more as compared to the LUF and LFU. Finally, the algorithms have demonstrated better performance under the geometric file access pattern, followed by the Zipf and random. These initial yet detailed results on the impact of the replication algorithms on the real-time Grid performance motivate the development of more sophisticated replication algorithms to better use of the Grid resources, which will be topic of the future research.

6. REFERENCES

[1] Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 23, 3 (2000), 187-200.

[2] Allock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Foster, I., Kesselman, C., Mader, S., Nefedova, V., Quesnel, D., Tuecke S., Data Management and Transfer in High Performance Computational Grid Environments. *Parallel Computing Journal*, 28, 5 (2002), 749-771.

[3] Ranganathan, K., Foster, I. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, 1, 1 (2003), 63-74.

[4] Camaron, D. G., Millar, A. P., Nicholson, C., C.-Schiaffino, R., Zini, F., Stockinger, K. Analysis of Scheduling and Replica Optimisation Strategies for Data Grids using OptorSim. *Journal of Grid Computing*, 2, 1 (2004), 57-69.

[5] Nicholson, C., Camaron, D. G., Doyle, A. T., Millar, A. P., Stockinger, K. Dynamic Data Replication in LCG 2008. *UK e-Science All Hands Conference* (2006).

[6] Tang, M., Lee, B.-S., Tang, X., Yeo, C.-K. The Impact of Data Replication on Job Scheduling Performance in the Data Grid. *Future Generation Computer Systems*, 22 (2006), 254-268.

[7] Desprez F., Vernois, A. Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid. *Journal of Grid Computing*, 4, 1 (2006), 19-31.

[8] Ranganathan, K., Foster, I., Identifying Dynamic Replication Strategies for a High-Performance Data Grid. *Lecture Notes In Computer Science*, 2242 (2001), 75-86.

[9] Lamehamedi, H., Shentu, Z., Szymanski, B., Deelman, E. Simulation of Dynamic Data Replication Strategies in Data Grids. *Heterogeneous Computing Workshop* (2003), 100b.

[10] Tang, M., Lee, B.-S., Yeo, C.-K., Tang, X. Dynamic Replication Algorithms for the Multi-tier Data Grid. *Future Generation Computer Systems*, 21 (2005), 775-790.

[11] Rahman, M. R., Barker, K., Alhaji, R. Replica Placement Design with Static Optimality and Dynamic Maintainability. *IEEE Int'l Symposium on Cluster Computing and the Grid* (2006).

[12] Keahey, K., Fredian, T., Peng, Q., Schissel, D. P., Thompson, M., Foster, I., Greenwald, M., McCune, D. Computational Grids in Action: The National Fusion Collaboratory. *Future Generation Computer Systems*, 18, 8 (2002), 1005-1015.

[13] Wang, Y., De Carlo, F., Mancini, D., McNulty, I., Tieman, B., Bresnahan, J., Foster, I., Insley, J., Lane, P., von Laszewski, G., Kesselman, C., Su, M.-H., Thiebaux, M. A High-Throughput X-ray Microtomography System at the Advanced Photon Source. *Review of Scientific Instruments*, 72, 4 (2001), 2062-2068.

Table 5. The performance of the fast spread and cascading with LRU, LFU, and LUF and caching for two different values of deadline factor.

		Deadline factor= 10 (Base)			Deadline factor= 20		
		Random	Geometric	Zipf	Random	Geometric	Zipf
Fast Spread	LRU	55	82	67	64	91	76
	LFU	55	80	64	63	90	74
	LUF	55	81	66	64	90	76
Cascading	LRU	53	79	62	61	89	71
	LFU	53	77	60	61	87	69
	LUF	53	77	58	61	88	66
Caching		55	57	56	64	65	64

Table 6. The performance of the fast spread and cascading with LRU, LFU, and LUF and caching for two different values of arrival rate.

		Arrival rate= 1 (Base)			Arrival rate= 2		
		Random	Geometric	Zipf	Random	Geometric	Zipf
Fast Spread	LRU	55	82	67	29	76	46
	LFU	55	80	64	28	73	42
	LUF	55	81	66	28	75	45
Cascading	LRU	53	79	62	24	70	40
	LFU	53	77	60	23	65	37
	LUF	53	77	58	24	66	33
Caching		55	57	56	27	30	29

[14] Gharai, L., Perkins, C., Real-time Collaborative Environments and the Grid. *Workshop on Advanced Collaborative Environments* (2004).

[15] Fox, G., Ho, A., Pallickara, S., Pierce, M., Wu, W. Grids for the GiG and Real Time Simulations. *IEEE International Symposium on Distributed Simulation and Real-Time Applications* (2005).

[16] Kim, K. H. Wide-Area Real-Time Distributed Computing in a Tightly Managed Optical Grid - An Optiputer Vision. *Int'l Conf. on Advanced Information Networking and Applications* (2004).

[17] Theys, M. D., Tan, M., Beck, N., Siegel, H. J., Jurczyk, M. A. Mathematical Model and Scheduling Heuristic for Satisfying Prioritized Data Requests in an Oversubscribed Communication Network. *IEEE Transactions on Parallel and Distributed Systems*, 11, 9 (2000), 969-988.

[18] Eltayeb, M., Doğan, A., Özgüner, F. Concurrent Scheduling: Efficient Heuristics for Online Large-Scale Data Transfers in Distributed Real-Time Environments. *IEEE Transactions on Parallel and Distributed Computing*, 17, 11 (2006), 1348-1359.

[19] Eltayeb, M., Doğan, A., Özgüner, F. A Path Selection-Based Algorithm for Real-time Data Staging in Grid Applications. *Journal of Parallel and Distributed Computing*, 65, 11 (2005), 1318-1328.

[20] Doğan., A. Performance of Real-Time Data Scheduling Heuristics Under Data Replacement Policies and Access Patterns in Data Grids. *Lecture Notes in Computer Science*, 4330 (2006), 884-893.

[21] GridPP Collaboration. GridPP: Development of the UK Computing Grid for Particle Physics. *Journal of Physics G: Nuclear and Particle Physics*, 32 N1-N20 (2006).

[22] Avery, P. Data Grids: A New Computational Infrastructure for Data Intensive Science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical & Engineering Sciences,* 360 (2002), 1191-1209.

[23] User's Guide: CSIM19 Simulation Engine (C Version), http://www.mesquite.com/documentation/index.htm#userc.