Proc. 5th International Conference on Algorithms and Architecture for Parallel Processing, ICA3PP'2002,
Bejing, China, October 2002, IEEE Computer Society Press, Los Alamitos, CA, 2002, pp. 378-383

Data Replication Strategies in Grid Environments*

Houda Lamehamedi, Boleslaw Szymanski, and Zujun Shentu
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
lamehh, szymansk, shentu@cs.rpi.edu

Ewa Deelman
Information Sciences Institute
University of Southern California
Los Angeles, CA 90292
deelman@isi.edu

Abstract

Data Grids provide geographically distributed resources for large-scale data-intensive applications that generate large data sets. However, ensuring efficient and fast access to such huge and widely distributed data is hindered by the high latencies of the Internet. To address these problems we introduce a set of replication management services and protocols that offer high data availability, low bandwidth consumption, increased fault tolerance, and improved scalability of the overall system. Replication decisions are made based on a cost model that evaluates data access costs and performance gains of creating each replica. The estimation of costs and gains is based on factors such as run-time accumulated read/write statistics, response time, bandwidth, and replica size. To address scalability, replicas are organized in a combination of hierarchical and flat topologies that represent propagation graphs that minimize inter-replica communication costs. To evaluate our model we use the network simulator NS. Our results prove that replication improves the performance of the data access on Data Grids, and that the gain increases with the size of the datasets used.

1. Introduction

The term grid computing refers to the emerging computational and networking infrastructure that is designed to provide pervasive and reliable access to data and computational resources over wide area network, across organizational domains.

A Data Grid connects a collection of hundreds of geographically distributed computers and storage resources located in different parts of the world to facilitate sharing of data and resources [2, 6, 20]. The size of the data that needs to be accessed on the Data Grid is on the order of Terabytes today and is soon expected to reach Petabytes. Ensuring efficient access to such huge and widely distributed data is a serious challenge to network and Grid designers. The major barrier to supporting fast data access in a Grid are the high latencies of Wide Area Networks (WANs) and the Internet, which impact the scalability and fault tolerance of the total Grid system. In this paper we investigate the use of replication on the Data Grid to improve its ability to access data efficiently. Experience from distributed system design shows that replication promotes high data availability, low bandwidth consumption, increased fault tolerance, and improved scalability [3, 21, 24, 15, 16]. We introduce high level replica protocols on top of a scalable replica management system combined with dynamic data distribution adapting to changing user needs. We use a runtime system to evaluate the access cost and performance gains of replication before moving or copying any data.

The performance of replication-based systems depends on a variety of factors, such as data placement and the protocol used to maintain consistency among object replicas. Our replication cost model is formulated as an optimization problem that minimizes the sum of the access costs to data in a Grid and the replica maintenance costs such as update propagation and cost of storage used by replicas.

To address the scalability, we replicate data in two alternative topologies: a ring and a fat-tree, which can also be organized on top of each other. The replica distribution topology is chosen according to the application's design and sharing patterns. To evaluate our model we use the network simulator *NS* [12] to generate different network topologies, and we use different dataset sizes to study the impact of replication on the data access cost on the overall grid. The paper is organized as follows. Section 2 gives an overview of previous work on grid replication. In Section 3, we introduce our replication topologies and algorithms. Section 4 describes the simulation framework and the results. Finally, we present brief conclusions and future directions in

*This work was partially supported by the NSF Grant KDI-9873138.The content of this paper does not necessarily reflect the position or policy of the U.S. Government—no official endorsement should be inferred or implied.

Section 5.

2. Replication in Data Grids

Globus is a community-based, open-architecture, open-source set of services and software libraries that support Grids and Grid applications [1, 2]. The Globus Toolkit [5] provides middleware services for grid computing environments. There are four main components of Globus: the Grid Security Infrastructure, the Globus Resource Management architecture, the Globus Information Management, and the Data Management architecture. This latter component provides the fundamental Data Grid tools [2]: a universal data transfer protocol for grid computing environments called GridFTP and the Replica Management infrastructure which includes the Replica Catalog and the Replica Management services for managing multiple copies of shared data sets. The Replica Catalog allows users to register files as logical collections. It also provides mappings from logical names of files and collections to the storage system locations of one or more replicas of these objects. The management service however, does not implement the full replica management functionality and it does not enforce any replication semantics. The system only provides the users with tools to replicate data at different locations under user-specific definitions without enforcing the user's assertions [2].

In this paper we introduce a new set of higher level services and protocols that provide full dynamic replication functionality on top of a highly scalable replica distribution topology. These higher-level tools can use basic replica management services implemented in Globus to automatically create new replicas at desirable locations or select among replicas based on network or storage system performance.

3. Replica Management Services Design

Replication has been studied extensively and different distributed replica management strategies have been proposed in the literature [22, 4, 17, 8, 7, 11, 18, 13]. In the context of data grid technology, replication is mostly used to reduce access latency and bandwidth consumption. In our approach, replication is also used to balance the load of data requests within the system both on the network and host levels, and to improve reliability. Given the size of the data stored on data grids, the placement of data replicas and the selection of consistency algorithms are crucial to the success of replication. Based on access cost and replication gains, the replica management system decides when to create a replica and where to place it. These decisions are made based on a cost model that evaluates the maintenance cost and access performance gains of creating each replica. The estimates of costs and gains are based on many factors, such as run-time accumulated read/write statistics, the chosen consistency algorithm, run-time measured network latency, response time, bandwidth, and replica size. These parameters are changing during the program execution, so they need to be measured at runtime and fed to an optimization procedure that minimizes data access costs by dynamically changing the replicas number and placement. In the following subsections we outline the design of the replication management system, and present our replication algorithm.

3.1. Replica Distribution Topologies

To ensure scalability, we use both hierarchical and flat propagation graphs spanning the overall set of replicas to overlay replicas on the data grid and minimize inter-replica communication costs. For the hierarchical topology, we introduce a modified fat-tree structure with redundant interconnections connecting its nodes; closer the node is to the root, more interconnections it has. The fat-tree was originally introduced by Leiserson [10] to improve the performance of interconnection networks in parallel computing systems. For the flat topology we use the ring topology.

The hierarchical distribution is well suited for multi-tier applications, while the ring topology suits best the multiple server or peer replica applications. In the peer-to-peer model, any replica can synchronize with any other replica, and any update can be applied at any accessible replica. The peer model has been implemented in many systems such as Locus [14], Bayou [23], Ficus [7], Roam [17], and Rumor [8]. In the hierarchical model, the replicas are placed at different levels, and communicate with each other in a client-server like scheme. This model has been implemented in many replication systems such as Coda [19]. To further exploit the properties of both topologies, we use a hybrid topology in which both the ring and fat-tree replica organizations can be combined into multi level hierarchies. In such a topology, the replica servers communicate with each other through a ring spanning graph, while the client replicas communicate with the servers and other replicas through the fat-tree connection graph. This approach improves both the data availability and the reliability of the ring topology and allows for a scalable expansion of the hierarchical distribution. Both the ring and fat-tree connection graphs represent virtual connections between the grid nodes that hold replicas of the same object. Depending on the topology, each node is aware of its neighbors or direct ancestors and children. Figure 1 shows a graph representing a hybrid three level replica topology and connection graph.

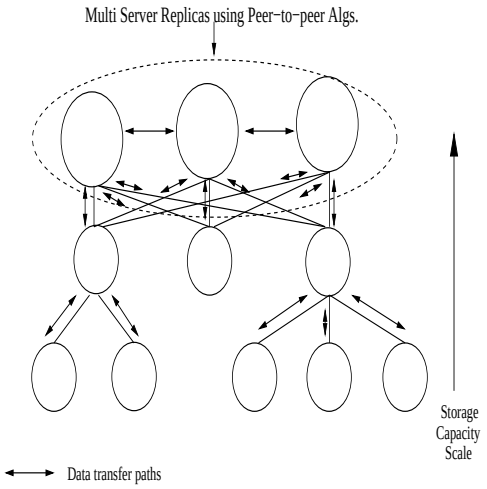

Figure 1. Hybrid Replica Connection Graph.

3.2. Replica Placement Algorithm

Each entity in the data grid maintains a replica set, which is initially empty. Replicas are created at nodes that receive high number of requests. Each node in the replica set maintains an index list of all entities that it contains, plus a list of the locations of these entities that it is aware of. In the hierarchical topology, each node maintains a list of its parents and children locations, while in the flat topology, each replica keeps a list of the locations of its neighbors. The index list entries contain file information such as size, name, type, and other attributes. Such index lists are called *local replica indices*. The replica set root(s) maintain(s) a list of all replica locations. This index list is called a *global replica index*. Used within a replica cataloging service, these indices identify the closest replica to a given site. A request is then transparently serviced by the nearest replica. This is especially important in the grid environment since the replica set changes dynamically as new replicas are being added and old ones deleted at runtime.

A replica is removed from a site when the user chooses to do so or when the system deletes it. The latter happens if the data is not used anymore locally, or if no requests have been made to access it remotely from other sites after a certain amount of time or when space is needed for more frequently used data. The runtime system is then responsible for updating the replica connection graph and making the necessary changes to replica indices. The connection graph should then be reorganized while maintaining its topology. In the Data Grid environment updates to single files happen very infrequently. Most modifications take a form of additions of files to a collection of files. In such scenarios not all replicas need to be aware of the updates. For that purpose, we intend to use optimistic replication to handle updates on the grid. Such an approach allows any machine storing a replica to perform an update locally. Hence, it minimizes the bandwidth and connectivity requirements for performing updates, and takes advantage of the replica connection topology to scale over distributed networks. It has also been shown to scale well in distributed environments [7, 8, 17, 18, 9, 25].

3.3. Runtime System

The replica management service includes a runtime component that dynamically evaluates the application and user's needs and accordingly adapts the replicas distribution. The runtime system monitors the users behavior and the network status, collecting accumulated read/write statistics and measuring response time, bandwidth, and replica size. The access cost of the replication scheme is calculated based on these parameters using an optimization procedure that is based on the following mathematical model.

Each node v in the overall system and a data object i are associated with a nonnegative read rate $\lambda_{v,i}$ and a nonnegative write rate $\mu_{v,i}$. If W is the write cost for a given object i and R is the read cost for the same object, then $W/R = \alpha_i$ is the ratio of the write cost for each node. If there are no replicas for object i in the system then the total data transfer cost for this object at node v is:

$$cost_{v,i} = (\lambda_{v,i} + \alpha_i \mu_{v,i}) size(i) d(v, r) \quad (1)$$

where r is the node containing the object i, $d(v, r)$ is the cost of sending a unit of data along the path from v to r, so $d(v, r) = 1/bandwidth(v, r)$.

Let N represent the set of all nodes in the system, R_i be the replica set of object i, and $c(v, R_i)$ denote the replica of object i closest to node v. Let T_v be the partition of nodes that would be serviced by v for future access requests to object i, assuming that v is added to R_i. Let $\lambda^t_{v,i}$ represent the total read rate of all nodes at partition T_v, and $\mu^t_{v,i}$ represent the total write rate of the partition.

The incremental cost of data transfer needed for placing a replica at v can be expressed in terms of the change in cost of traffic generated by the read/write requests $RW^i(R_i, v) = -\lambda^t_{v,i} + \alpha_i(\mu^t_{r,i} - \mu^t_{v,i})$, and the size of data:

$$cost^i(N, R_i, v) = RW^i(R_i, v) size(i) d(v, c(v, R_i)) \quad (2)$$

Indeed, adding v to R_i decreases the read cost of each node in T_v by $size(i)d(v, c(v, R_i))$ and increases the write cost of each node in the $N - T_v$ by $size(i)d(v, c(v, R_i))$, but it does not change other costs. Thus the total data transfer cost for object i with a replica set R_i is given by:

$$Cost(N, R_i) = cost(N, r) + \sum_{v \in R_i - \{r\}} cost^i(N, R_i, v) \quad (3)$$

Given the structure of the data grid and the associated read/write patterns for object i, we only need to consider the problem of minimizing the following cost:

$$Cost'(N, R_i) = \sum_{v \in R_i - \{r\}} cost^i(N, R_i, v) \qquad (4)$$

The above formula expresses the data transfer cost for object i improved thanks to the placement of a set of replicas $R_i - \{r\}$. Using access cost statistics, the runtime system compares the replications gains to replication costs (update cost) and informs the replica management service whether to place a replica on node v or not.

4. Simulation Framework and Results

Essentially, the simulated events represent read and write requests, although the presented results were obtained with simulation of only read requests. The simulation generates random background traffic in the network and the stream of requests for the grid data. In the future, agents will be placed in specific network nodes to monitor the read/write frequencies, network bandwidth and other variables to evaluate the replica placement policy. These agents will gather network statistics and help to optimally place file replicas. The current simulation is described in the next section.

4.1. Simulation Framework

To evaluate our model we are using *NS* [12], the network simulator which enables us to generate different network topologies. However, to be able to simulate the specifics of the replication in the grid, we introduced a separate grid simulator on top of *NS*. This additional simulator extends the original semantics of a node object in *NS*. Each node in the grid is able to specify its storage capacity, organization of its local data files, the relative processor performance, and maintain a list of its neighbors and peer replica nodes. We are using three different node models: data server nodes (storage site), user nodes, and cache or intermediate nodes. The cache nodes represent nodes in the grid hierarchy that have higher storage capacities than user nodes, but smaller than that of a storage site. These nodes may contain parts of the data stored at the main storage sites. Placing data in the intermediate nodes makes it closer and more accessible to some users. The user nodes represent the network nodes where the access requests are generated. The network interface model is specified in the link object that is provided in *NS*. The link object is used to model the physical interconnections in the simulated network, such as link bandwidth and latency. Figure 2 shows the simulation model used in our experiments.

To collect statistics data like access frequencies and storing patterns, we are attaching to each node a monitoring

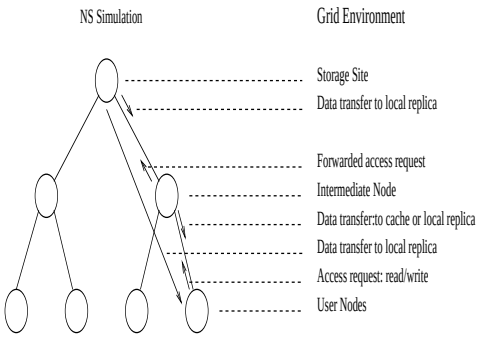

Figure 2. Simulation Model

agent that is responsible for computing the number of data requests generated at each node, as well as the number of requests received from other nodes. The data collected is then evaluated by the replica placement protocol using the cost function. Currently, we are placing replicas ourselves, creating three different scenarios to measure the data access costs with the number of replicas varying from none to a few to many.

Packets representing grid user requests as well as the start and the end of grid data transmission in *NS* transfer control of the simulation to the grid node simulator. The latter simulates the replication decision at each node and generates new *NS* traffic (sending requests from the user or cache nodes or sending the requested data down the path to the replica site or the user). This separation of the network simulation strata, the grid node and replication enables us to use the existing package, *NS*, and add only the grid specific elements to the simulation. The *NS* simulator allowed us to compare the performances of three different replication scenarios. In the future, it will be used to select the optimal replica placement policy.

4.2. Simulation Experiments and Results

In the absence of real trace data, we are simulating a two-tier Data Grid topology using eight different file sizes ranging from 100MB to 1GB. Initially only one replica per dataset exists in the system. Users are mapped evenly across sites and submit a number of sequential data access requests (only read requests are used). Requests are processed sequentially and each user accesses only one file. However, collectively, users utilize some files more than others.

The grid topology consists of a 15 nodes binary tree shown in Figure 3, in which users reside at leaf nodes. The connectivity bandwidths of user nodes to the higher level nodes is of 10MB/sec, whereas the bandwidth of intermediate-to-intermediate and intermediate-to-root nodes is 100MB/sec. To simulate a real world grid environment, background traffic is introduced by using extra

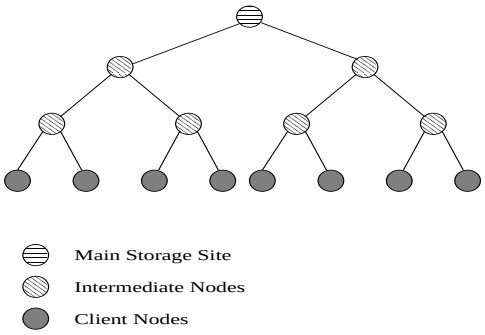

Figure 3. Simulated Grid Topology

nodes in the simulation that generate random traffic in the grid. We ran a total of 30 simulations using three different scenarios. In the first scenario there are no replicas. In the second scenario replicas are placed at the second level of the tree, i.e. the first intermediate nodes. In the third scenario replicas are placed at the lower level intermediate nodes, hence located closer to the users. We use response time as our performance metric as it represents both the data transfer costs and gains in different replica placement scenarios. Figure 4 shows a graph representing the average response time per file size for the three different scenarios.

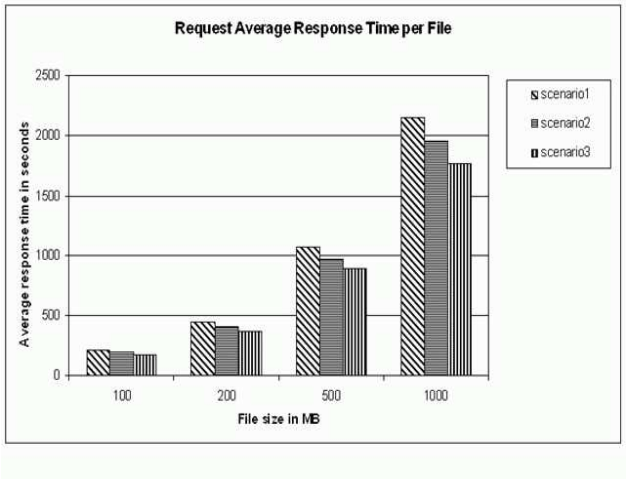

Figure 4. Simulation Results: Access Time in Different Scenarios

The results show that better performance is achieved when replicas are placed closer to the users. The gains are more considerable for larger file sizes for which replication offers about 12% improvement. We expect that in the real Data Grid environments these gains will be even more considerable as dataset sizes are expected to reach Terabytes.

5. Conclusions and Future Work

We have addressed the problem of replication in Data Grid environments by investigating the use of a new set of highly decentralized dynamic replication services that can be used to improve data access time, reliability, data availability, bandwidth consumption, fault tolerance, and scalability of the overall system. We have also used a cost function that dynamically evaluates the replica placement policy by comparing the replica maintenance costs and data access gains of creating a replica at any given location.

Our results show that using replication improves the data access performance of the overall system as measured by the response time (scenario 1 versus scenarios 2 and 3) and that the placement of replicas matters as well (scenario 2 versus scenario 3). With the introduction of additional traffic on the grid, we created an environment in which the connection network is not solely dedicated to the grid data transfer. This additional traffic uses part of the bandwidth and introduces extra delays. Our results also show that performance gains increase with the size of data used. As dataset sizes in Data Grid environments are reported to reach the Terabyte scale, we expect that a larger scale use or our model will yield better performance results.

These results are very promising, but they are based on synthetic workloads and simplified grid scenarios. In future work, we will investigate more realistic scenarios, and real user access patterns. We will also consider the effect of different replica consistency algorithms on the overall performance of the Grid. We are also interested in exploring different adaptive replication algorithms that select replication algorithms dynamically depending on current conditions.

References

[1] W. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Lee, A. Sim, A. Shoshani, B. Drach, D. Williams. ”High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies.” *Proceedings of SC 2001*, Denver, CO, November 2001.

[2] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, ”Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing,” *IEEE Mass Storage Conference*, 2001.

[3] W. Allcock, A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, ”The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets,” *Journal of Network and Computer Applications*, 23:187-200, 2001.

[4] G. Coulouris, J. Dollimore, T. Kindberg, "Distributed Systems, concepts and designs," third Edition, Addisson Wesley, 2001.

[5] I. Foster, C. Kesselman, S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations." International J. Supercomputer Applications, 15(3), 2001.

[6] Foster, I. "The Grid: A New Infrastructure for 21st Century Science", *Physics Today*, 54 (2). 2002.

[7] R. Guy, J. Heidmenn, W. Mak, T. Page Jr., G. Popek, and D. Rothmeier, "Implementation of the Ficus Replicated File system," *Proceedings of the summer Usenix Conference*, 1990.

[8] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek, "Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication," *Workshop on Mobile Data Access*, November 1998.

[9] D. Hagimont, D. Louvegnies, "Javanaise: Distributed Shared Objects for Internet Cooperative Applications," IFTP International Conference on Distributed Systems, Platforms, and Open Distributed Processing Middleware98, the Lake District, England, 1998.

[10] C. H. Leiserson, "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," IEEE Transactions on Computers, vol. C-34, no. 10, pp. 892-901, October 1985.

[11] N. Narasimhan, "Transparent Fault Tolerance For Java Remote Method Invocation," Ph.D Dissertation, Department of Electrical and Computer Engineering, University of California, Santa Barbara, June 2001.

[12] NS network simulator. *http://www-mash.cs.berkeley.edu/ns*.

[13] T. Page, R. Guy, J. Heidemann, D. Ratner, P. Reiher, A. Goel, G. Kuenning, and G. Popek, "Perspectives on Optimistically Replicated, Peer-To-Peer Filing," Software - Practice and Experience, Dec. 1997.

[14] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G. Thiel, "Locus: A Network Transparent High Reliability Distributed System," Proceedings of the Eighth Symposium on Operating Systems Principles, pp 169-177 ACM, December 1981.

[15] K. Ranganathan and I. Foster, "Identifying Dynamic Replication Strategies For a High performance Data Grid," Proceedings of the International Grid Computing Workshop, Denver, November 2001.

[16] K. Ranganathan and I. Foster, "Design and Evaluation of Replication Strategies for a High Performance Data Grid," International Conference on Computing in High Energy and Nuclear Physics, Beijing, September 2001.

[17] D. H. Ratner, "Roam: A Scalable Replication System for Mobile and Disconnected Computing," PhDThesis, University of California, Los Angeles, Los Angeles CA, January 1998.

[18] Y. Saito and H. Levy, "Optimistic Replication for Internet Data Services", In Proc. of the 14th Intl. Conf.on Distributed Computing, p. 297-314, October 2000.

[19] M. Satyanarayanan, J. Kister, P. Kumar, M. Okasaki, E. Siegel, and D. Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment," IEEE Transactions on Computers, 39(4):447-459, April 1990.

[20] Stevens, R., Woodward, P., DeFanti, T. and Catlett, C., "From the I-WAY to the National Technology Grid", *Communications of the ACM*, 40 (11). 50-61. 1997.

[21] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, B. Tierney , "File and Object Replication in Data Grids," *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*, IEEE Press, August 2001.

[22] A. S. Tanenbaum and M. van Steen, "Distributed Systems, Principles and Paradigms," first edition, Prentice Hall, 2002.

[23] D. Terry, M. Theimer, K. Peterson, A. Demers, M. Spreitzer, and C. Hausen, "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System," *Proceedings of the fifteenth Symposium on Operating systems Principles*, pp 49-70 ACM, October 1983.

[24] S. Vazhkudai, S. Tuecke, I. Foster, "Replica Selection in the Globus Data Grid," *Proceedings of the First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001)*, pp. 106-113, IEEE Computer Society Press, May 2001.

[25] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso, "Understanding Replication in Databases and Distributed Systems," *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS'2000)*.