Identifying Dynamic Replication Strategies for a High-Performance Data Grid

Kavitha Ranganathan and Ian Foster

Department of Computer Science, The University of Chicago
1100 E 58th Street, Chicago, IL 60637
krangana@cs.uchicago.edu, foster@mcs.anl.gov

Abstract. Dynamic replication can be used to reduce bandwidth consumption and access latency in high performance "data grids" where users require remote access to large files. Different replication strategies can be defined depending on when, where, and how replicas are created and destroyed. We describe a simulation framework that we have developed to enable comparative studies of alternative dynamic replication strategies. We present preliminary results obtained with this simulator, in which we evaluate the performance of five different replication strategies for three different kinds of access patterns. The data in this scenario is read-only and so there are no consistency issues involved. The simulation results show that significant savings in latency and bandwidth can be obtained if the access patterns contain a small degree of geographical locality.

1 Introduction

A data grid connects a collection of geographically distributed computer and storage resources [6] that may be located in different parts of a country or even in different countries, and enables users to share data and other resources. Research projects such as GriPhyN [11], PPDG, and the EU DataGrid aim to build scientific data grids that enable scientists sitting at various universities and research labs to collaborate with one another and share data sets and computational power.

Physics experiments such as CMS, ATLAS, LIGO and SDSS [11] will churn out large amounts of scientific data (in some cases, the scale of petabytes/year). This data needs to be used by thousands of scientists around the world. The sheer volume of the data and computation involved poses new problems that deal with data access, processing and distribution.

There are two aspects to a grid: sharing of data and sharing of resources. A scientist located at a small university may need to run a time consuming processing job on a huge data set. She may choose to get the data from where it exists to the local computing resource and run the job there. Alternatively it may be better to transfer the job to where the data exists or, both the job specification and the data may be sent to a

third location that will perform the computation and return the results to the scientist. We focus here only on the data distribution aspect of a grid.

The data grid envisioned by the GriPhyN project is hierarchical in nature and is organized in tiers. The source where the data is produced is denoted as Tier 0 (e.g. CERN). Next are the Tier 1 national centers, the Tier 2 regional centers (RC), the Tier 3 workgroups and finally Tier 4, which consists of thousands of desktops.

1.1 Replication

When a user generates a request for a file, large amounts of bandwidth could be consumed to transfer the file from the server to the client. Furthermore the latency involved could be significant considering the size of the files involved. Our study investigates the usefulness of creating replicas to distribute data sets among the various scientists in the grid. The main aims of using replication are to reduce access latency and bandwidth consumption. Replication can also help in load balancing and can improve reliability by creating multiple copies of the same data. Static replication can be used to achieve some of the above-mentioned gains but has the drawback that it cannot adapt to changes in user behavior. In our scenario, where the data amounts to petabytes, and the user community is in the order of thousands around the world, static replication does not sound feasible. Such a system needs dynamic replication strategies, where replica creation, deletion and management are done automatically and strategies have the ability to adapt to changes in user behavior. Our study examines different dynamic replication strategies for a grid. There are many related questions of resource discovery (so that the request goes to the nearest replica) and furthermore how to distribute these requests among replicas to archive best results. In this study however we shall concentrate on the replica placement issue.

The three fundamental questions any replica placement strategy has to answer are: When should replicas be created? Which files should be replicated? Where should replicas be placed? The answers to these questions lead us to different replication strategies.

We use simulation to evaluate the performance of each different strategy. Since most datasets in the scientific data grid scenario are read-only, we do not consider the overhead of updates. This paper describes a grid simulator framework and also reports the preliminary results obtained using our simulator.

The rest of the paper is organized as follows. Section 2 describes the specific grid scenario that we use for our simulations. Section 3 discusses the simulator we built to conduct our experiments. Section 4 describes the replication and caching strategies that we evaluate. Section 5 presents the results of the experiments and we interpret the results of the experiments in Section 6. We end the paper with our conclusions and future directions in Section 7.

2 Grid Scenario

The particular data grid setup that we are studying in this paper is described below.

There are four tiers in the grid with all data being produced at the top most tier (the root). Tier 2 consists of the regional centers around the country; in the scenario considered in the experiments reported here, we have four of them. The next tier is composed of work groups at universities or research labs. The final tier (individual workstations) consists of the sources from which the requests arrive. There are a total of 85 nodes in the grid with 64 of them generating requests.

The storage capacity at each tier is given in Table 1. In our experiments, we assume that all network links are 320 Mbytes/sec in bandwidth [13]. In reality network bandwidths vary widely across tiers. The total data generated at the source is assumed to be 2.2 petabytes. The data is stored in files of uniform size of two gigabytes each.

Table 1. System parameters. Network performance and node capacity of a node at each tier in the hierarchy as described in [13].

Tier	Network Bandwidth to Next Tier (MB/s)	Storage Capacity (TB)
1	320	2200
2	320	1000
3	320	120

All requests for files are generated from the leaf nodes. Request patterns for the files can exhibit various locality properties, including:

Temporal Locality: Recently accessed files are likely to be accessed again.

Geographical locality (Client locality): Files recently accessed by a client are likely to be accessed by nearby clients.

Spatial Locality (File Locality): Files near a recently accessed file are likely to be accessed.

In our definition of spatial locality we have to specify what "near" means. This definition involves a study of the nature of the data in the files and how we can relate files to each other. Since this paper deals with a general data grid we defer the study of relationships of data until we model a specific grid.

We yet do not know to what extent file access patterns will exhibit the locality properties described above and whether there will be any locality. We can only make educated guesses at this point. The worst-case scenario is when the access patterns do not exhibit any locality at all, generating random access patterns can simulate this situation.

3 Methodology of Study

To identify a suitable replication strategy for a high performance data grid we decided to use a simulator. Since none of the tools currently available exactly fitted our needs, we built a simulator to model a data grid and data transfers in it. Our simulator uses PARSEC [15], a discrete event simulation tool to model events like file requests and data transfers.

3.1 The Simulator

The simulator consists of three parts. The basic core simulates the various nodes in the different tiers of the data grid, the links between them, and the file transfers from one tier to another. Various replication strategies are built on top of this core and compose the next layer. The final component is the driver entity of the program, which triggers file requests. The driver entity reads an input file, specifying the access patterns to be simulated.

3.2 How the Simulation Works

Topology specification: Starting a simulation involves first specifying the topology of the grid, including the number of nodes at each tier, how they are connected to each other, the bandwidth of each link, and the location of the files across various nodes.

Starting the simulation: Access patterns are read from a file, with each line representing one access and specifying at what time which node needs a particular file. The driver reads the data and triggers the corresponding node. When a node receives a "File needed by me" trigger it needs to locate and request the "nearest" replica of that file.

Locating nearest replica: There are various proposed methods for locating the nearest replica; some of these can involve complex algorithms to identify the closest copy. Location of the best replica is however a related but different topic than what we are trying to answer. This paper concentrates on designing and isolating the best replica placement strategy for a grid. However, to show the effectiveness of any dynamic replication strategy a node needs to be able to identify the nearest replica. We solve this problem by using the 'least number of hops' heuristic. The nearest replica is one, which is the least number of steps away from the node. In the case of a tie between two or more replicas, one of them is selected randomly.

File Transfer: Once the server gets the request for a file, it sends it off to the client. The tree structure of the grid means that there is only one shortest path that the messages and files can travel to get to the destination. When a file is being transferred through a link, the link is busy and cannot transport any other file for the duration of the transfer. The delay incurred in transferring a file depends on the size of the file, the bandwidth of the link and the number of pending requests. A node is busy for the

duration it transfers its file to the network and any incoming data has to wait for the current transaction to finish.

Record Keeping: Each node keeps a record of how much time it took for each file that it requested to be transported to it. This time record forms a basis to compare various replication strategies. The same series of file request are run through different strategies and the one that has a lower average response time is considered better than the others. The various replication strategies are described in the next section.

Scaling: The amount of data in the system is in the order of petabytes. To enable simulation of such large data values, a scale of 1:10,000 was used. That is, the number of files in the system was reduced by a factor of 10,000. Accordingly the storage capacity at each tier was also reduced. Table 2 below illustrates this fact.

Table 2. Grid parameters before and after scaling

	Actual Size	After Scaling
Number of files	1,000,000	100
Storage at Tier 1	2200 Terabytes	220 Gigabytes
Tier 2	1000 Terabytes	100 Gigabytes
Tier 3	120 Terabytes	12 Gigabytes

The link bandwidths and file sizes remained the same. The reason we use a scaling factor is to make the simulation of such a system feasible on a single machine. The meta-data for a million files would be very memory intensive. Since we need to scale the number of files, the storage capacity at each tier needs to be scaled accordingly. This is because the performance of replication strategies is directly dependent on the percentage of files that can be stored at each node. Scaling both the number of files in the system and the capacity at each node achieves this. We do not scale the file sizes also as that would have an increasing effect on the percentage of files that can be stored at each node. Since the file sizes are not scaled, the bandwidths also remain unscaled so that the transport latency is modeled correctly. Individual workstations are assumed to be able to cache one file of size 2 Gigabytes.

Access Patterns: Since the physics data grid is not yet functional there are no actual file access patterns available as of now, and we must work with artificial traces. We derive three such traces. The simulation was first run on random access patterns. This being the worst-case scenario, more realistic access patterns that contained varying amounts of temporal and geographical locality were also generated The three different kinds of traces are described below:

P-random: Random access patterns. No locality in patterns.

P1: Data that contained a small degree of temporal locality

P2: Data containing a small degree of geographical and temporal locality

The index used to measure the amount of locality in the patterns is denoted by ‘q’, where $0<q< 1$. If $q = 0$, it means that requests are completely random and there is no locality. At the other end of the spectrum, when $q =1$ it means all requests are for the

same file. We used q = 0.05 to generate data with a small degree of temporal/geographical locality, using geometric distribution for file popularity.

3.3 Performance Evaluation

We compare different replication strategies by measuring the average response time and the total bandwidth consumed.

Response Time: This is the time that elapses from when a node sends a request for a file until it receives the complete file. If a local copy of the file exists, the response time is assumed to be zero. The average of all response times for the length of the simulation is calculated.

Bandwidth Consumption: This includes the bandwidth consumed for data transfers occurred when a node requests a file and when a server creates a replica at another node.

4 Replication/Caching Strategies

We implemented and evaluated six different strategies. This work and our results both demonstrate what the simulator is capable of doing, and also help us understand the dynamics of a grid system.

In this paper we distinguish between caching and replication. Replication is assumed to be a server side phenomenon. A server decides when and where to create a copy of one of its files. It may do this randomly or by recording client behavior or by some other means. But the decision to make a copy (replica) and send it to some other node is taken solely by the server. Caching is defined as a client side phenomenon. A client requests a file and stores a copy of the file locally for future use. Any other nearby node can also request that cached copy. The different strategies are discussed below.

Strategy 1: No Replication or Caching: The base case against which we compare the various strategies is when no replication takes place. The entire data set is available at the root of the hierarchy when the simulation starts. We then run the set of access patterns and calculate the average response time and bandwidth consumed when there is no replication involved

Strategy 2: Best Client: Each node maintains a detailed history for each file that it contains [12] indicating the number of requests for that file and the nodes that each request came from. The replication strategy then works as follows: At a given time interval each node checks to see if the number of requests for any of its file has exceeded a threshold. If so, the best client for that file is identified. The best client is the one that has generated the most requests for that file. The node then creates a replica of that file at the best client. Thus all files that exceed the threshold of the number of requests are replicated elsewhere. Once a replica is created, the ‘request details’ for the file at the server node are cleared. After this, the recording process begins again

The file replacement algorithm at each node is discussed later and is the same for all of the replication strategies to facilitate a common ground for comparing them.

Strategy 3: Cascading Replication: The best analogy for this strategy is a three-tiered fountain. The water originates at the top. When it fills the top ledge it overflows to the next level. When this level also overflows the water reaches down to the lowest part.

The data in this strategy flows in a similar way. Once the threshold for a file is exceeded at the root, a replica is created at the next level, but on the path to the best client. Hence the new site for the replica is an ancestor of the best client. Once the threshold for the file is exceeded at Level 2 it is then replicated at the next lower tier and so on. A popular file may ultimately be replicated at the client itself.

The advantage of this strategy is that storage space at all tiers is used. Another advantage is that if the access patterns do not exhibit a high degree of temporal locality, geographical locality is exploited by this strategy. By not replicating at the very source of requests but at a higher level the data is brought closer to other nodes in the same sub-tree.

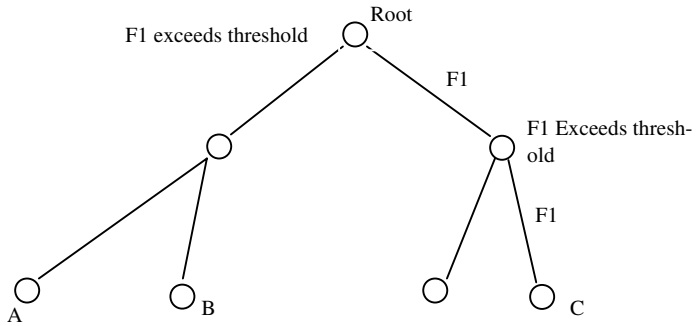

Fig.1.*Cascading Replication.* At the root the number of requests for F1 exceed the threshold and a copy is sent to the next layer. Eventually the threshold is exceeded at the next layer and a copy is sent to Client C.

Strategy 4: Plain Caching: The client that requests a file stores a copy locally. Since these files are large (2 Gigabytes each) and a client has enough space to store only one file at a time, the files get replaced quickly.

Strategy 5: Caching plus Cascading Replication: This combines strategy three and four. The client caches files locally. The server periodically identifies the popular files and propagates them down the hierarchy. Note that the clients are always located at the leaves of the tree but any node in the hierarchy can be a server. Specifically, a Client can act as a Server to its siblings. (Siblings are nodes that have the same parent).

Strategy 6: Fast Spread: In this method a replica of the file is stored at each node along its path to the client [14]. That is, when a client requests a file, a copy is stored

at each tier on the way. This leads to a faster spread of data. The generic file replacement strategy used for all the cases is discussed below.

File Replacement Strategy: The storage spaces at all levels eventually fill up. An efficient file replacement strategy is needed so that popular files are retained and not displaced as and when new files arrive. Initially we decided to expunge the least popular file from the list. But this might delete a relatively new file that has just come in and not yet been requested that might become popular in the future. Thus there needs to be a measure of time and hence the age of each file in that cache. The replacement strategy we employed takes care of both these aspects and is a combination of least popular and the age of the file. If more than one file are equally unpopular, the oldest file is deleted

We clear the popularity logs at a given time interval in order to capture the dynamics of the access patterns. Over time users may shift from using one group of files to another group etc. We can thus expect that the effectiveness of the strategy will depend on how well the time interval is tuned to the access behavior. Another parameter that has to be tuned for each scenario is the threshold. Only if the number of requests exceeds this threshold is the file replicated. We can imagine refining the algorithm so that the "time_to_check" interval and threshold automatically change according to user behavior. However, this is left for future work.

5 Experimental Results

We present the results for five strategies, no replication, plain Caching, Best Client, Caching+Cascading, and Fast Spread. We do not discuss pure Cascading as its results can be found in strategy 5, Cascading+Caching. The experiments were run on the three access patterns defined earlier: P-random, P1, and P2 and each simulation was run for a thousand requests.

Random patterns are the worst-case scenario and it seems sensible to assume the patterns will exhibit some amount of geographical and temporal locality, as scientists tend to work in groups on projects. That said we proceed to discuss the results obtained from the experiments.

When P-random data was used, all strategies except for Best-Client and Cascading show significant improvement in the access latency as compared to the case of no replication. Best-Client does not seem to work well for random access patterns, as the average response time is four times more than when no replication/caching policy is used. Again, in terms of bandwidth savings, Best Client utilizes almost the same amount of bandwidth as the base case of no replication. In the case of P1 access patterns (patterns with a small amount of temporal locality) all strategies except for Best-Client yield positive savings in both access latency and bandwidth consumption. Only in the case of P2 (patterns with both temporal and geographical locality) does Best-Client show any savings. Even in this case the bandwidth savings by using Best-Client are marginal (10% savings as compared to the base case of no replication/caching) although the latency savings are significant (40% when compared to the

base case). However Best Client consistently performs much worse than Plain Caching.

We next discuss the results obtained by the other three strategies as Best-Client does not seem a good candidate for a replication strategy for a grid. The two graphs below contain results for Cascading+Caching, Fast Spread, and Plain Caching. The first two strategies are compared, with Plain Caching being the standard of comparison. Thus the graphs illustrate the savings achieved by Fast Spread and Cascading, beyond those achieved by only caching the files.

As Fig.3 indicates, Cascading does not work well when the access patterns contain no locality. For random data, the response time is far better when Plain Caching is used rather than Cascading. Fast spread works much better than Plain Caching for random data. There is almost a 50% reduction in response times in the case of Fast Spread.

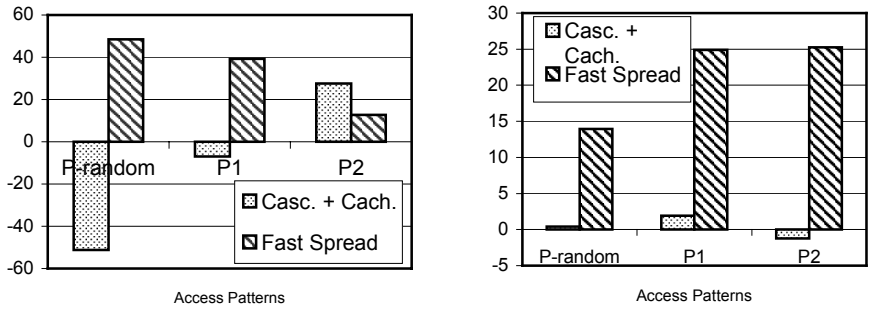

Fig.3. Percentage savings in response time (*left*) and bandwidth consumption (*right*) as compared to Plain Caching for all three kinds of access patterns

In the case of P1 patterns, the advantage of Fast-Spread over Caching decreases and Cascading works almost as well as Caching. Once the data contains more locality (as is the case with P2) Cascading has a significant improvement in performance, its average response time is almost 30% less than that for Plain Caching. Fast Spread however has less than 15% improvement over Caching for patterns that contain geographical locality. These results are interpreted further in Section 6.

We now discuss the amount of bandwidth savings for the different cases. As shown in Fig.3, Cascading does not differ significantly from Caching in terms of bandwidth consumption. The difference between the two strategies falls with the range of plus or minus 2% for all the access patterns. Fast Spread on the other hand leads to large savings in bandwidth usage, up to 25% when the access patterns contain locality.

6 Discussions

Among the methods of replication that we consider in the paper, Best-Client performs the worst. In many cases the overheads it creates are more than the advantages of the strategy and it performs worse than the base case of no replication.

Considering the remaining candidates Plain Caching, Cascading + Caching and Fast Spread there is no sure best strategy for all scenarios. Fast Spread consistently performs better than Caching both in terms of response time and bandwidth savings. In spite of the overhead Fast Spread has, in terms of excessive creation of replicas its advantages over Caching are plainly evident. The bandwidth savings of Fast Spread are up to 25% more than that of Caching [refer to Fig.3]. The disadvantage is that it has high storage requirements. The entire storage space at each tier is fully utilized by Fast-Spread.

Cascading on the other hand utilizes less then 50% of the storage space at each tier since it involves a judicious creation of replicas. The bandwidth requirements of Cascading however are greater than that for Fast Spread. This is because every replica that is created has to be sent separately to the new location as opposed to Fast Spread where a copy is created in the process of transferring the requested file.

Cascading, moreover does not work well when the access patterns are totally random. In fact it does not even work as well as Caching for random user patterns. This can be attributed to the fact that the overhead in creating these extra copies of files is not offset by the advantage of moving them closer to the users. The copied files are not asked for often enough to justify the increased data movement. However, when the patterns contain a small amount of locality, the performance of Cascading improves significantly. It even performs better than Fast Spread for P2 patterns, with an average response time almost 18% better than that for Fast-Spread. There are however no significant bandwidth savings in using Cascading over Caching when we assume only a small amount of geographical locality.

These results lead us to conclude that if grid users exhibit total randomness in accessing data then the strategy that would work best is Fast Spread. If however there are sufficient amount of geographical locality in the access patterns, Cascading as a replication policy would work better than the others. With more or less the same amount of bandwidth utilization as Caching, Cascading lowers response times significantly, while judiciously using storage space.

The above results also indicate that depending on what is more important in the grid scenario, lower response times or lesser bandwidth consumption, a tradeoff between Cascading and Fast Spread can be made. If the chief aim is to elicit faster responses from the system, Cascading might work better. On the other hand if conserving bandwidth is of top priority, Fast Spread is a better grid replication strategy.

7 Conclusions and Future Work

We have presented and evaluated dynamic replication strategies for managing large data sets in a high performance data grid. Replication enables faster access to files, decreases bandwidth consumption, and distributes server load. In contrast to static replication, dynamic replication automatically creates and deletes replicas according to changing access patterns, and thus ensures that the benefits of replication continue even if user behavior changes.

We discussed the components of the simulator we built and explained how we used the simulator to study the performance of different replication strategies within a grid environment We generated three different kinds of access patterns, random, temporal, and geographical and showed how the bandwidth savings and latency differ with access patterns. Two strategies performed the best in our tests: Cascading and Fast Spread. While Fast Spread worked well for random request patterns, Cascading worked better when there was a small amount of locality. We analyzed why we thought these were the best strategies and the pros and cons of each method.

In future work, we want to use the simulator to test the performance of some more advanced replication strategies. We also have plans to extend the simulator so that we can plug in different algorithms for selecting the best replica.

In our work so far, the replication strategies we discussed exploit both temporal and geographical locality of the request patterns. What we put off for later is considering the spatial locality of the requests. Once we better understand the relationship among various files in a scientific data set, some amount of pre-fetching is possible.

Another area for further research is to study the movement of code towards data. We have assumed here that clients ask for files and locally run the data through their code to analyze the data. Thus we have only considered moving data towards code. Another option is to move code towards where data resides and communicate only the result of the computation back to the client. This is a feasible option considering that in the data grid scenario the data may be tens of thousands of times larger than either the code or the result.

A data grid enables thousands of scientists sitting at various universities and research centers to collaborate and share their data and resources. The sheer volume of the data and computation calls for sophisticated data management and resource allocation. This paper is one step into better understanding the dynamics of such a system and the issues involved in increasing the overall efficiency of a grid by intelligent replica creation and movement.

Acknowledgements

This research was supported by the National Science Foundation's "GriPhyN" project under contract ITR-0086044.

References

1. Acharya, S., Zdonik, S.B.: An efficient scheme for dynamic data replication: Technical report CS-93-94-43, Brown University
2. Baentsc, M., et al.: Quantifying the overall impact of caching and replication in the web. University of Kaiserslautern February (1997)
3. Bestavros, A., Cunha, C.: Server-initiated document dissemination for the WWW. IEEE Data Engineering Bulletin, Vol. 19 (1996) 3 –11
4. Bestavros, A.: Demand-based document dissemination to reduce traffic and balance load in distributed information systems. IEEE symposium on Parallel and Distributed Processing, San Antonio, TX (1995)
5. Bhatacharjee, S., Calvert, K.L., Zegura, E.: Self-organizing wide-area network caches. Georgia Institute of Technology GIT-CC-97/31(1997)
6. 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 Data Sets. J. Network and Computer Applications (2000)
7. Chuang, J.C.I., Sirbu, M.A.: Distributed Network Storage with Quality-of-Service Guarantees. Proc. INET'99 (1999)
8. Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: a scalable wide-area web cache sharing protocol. ACM SIGCOMM (1998)
9. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann (1999)
10. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Intl. J. Supercomputer Applications, (to appear).
11. GriPhyN: The Grid Physics Network Project http://www.griphyn.org
12. Gwertzman, J., Seltzer, M.: The case for geographical push-caching. 5th Annual Workshop on Hot Operating Systems (1995)
13. Holtman, K.: HEPGRID2001: A Benchmark for Virtual Data Grid Schedulers. http://kholtman.home.cern.ch/kholtman/tmp/benchv3.ps
14. Michel, S., Nguyen, K., Rosenstein, A., Zhang, L., Floyd, L., Jacobson, V.: Adaptive Web Caching: Towards a New Global Caching Architecture. Proceedings of the 3rd International WWW Caching Workshop (1998)
15. Parsec home page http://pcl.cs.ucla.edu/projects/parsec
16. Rabinovich, M., Aggarwal, A.: RaDaR: A scalable architecture for a global Web hosting service. The 8th Int. World Wide Web Conf, May (1999)
17. Samar, A., Stockinger, H.: Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication. IASTED International Conference on Applied Informatics, Innsbruck, Austria (2001)
18. Wolfson, O., Jajodia, S., Huang, Y.: An Adaptive Data Replication algorithm. ACM transactions on Database Systems (1997)