Design and Evaluation of 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

Physics experiments that generate large amounts of data need to be able to share it with researchers around the world. High performance grids facilitate the distribution of such data to geographically remote places. Dynamic replication can be used as a technique to reduce bandwidth consumption and access latency in accessing these huge amounts of data. We describe a simulation framework that we have developed to model a grid scenario, which enables comparative studies of alternative dynamic replication strategies. We present preliminary results obtained with this simulator, in which we evaluate the performance of six different replication strategies for three different kinds of access patterns. The simulation results show that the best strategy has significant savings in latency and bandwidth consumption if the access patterns contain a moderate amount of geographical locality.

1 Introduction

Physics experiments such as CMS, ATLAS, LIGO and SDSS will churn out large amounts of scientific data (in 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. Scientific grids make this kind of large-scale collaboration possible. 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 (CERN in this case). Next are the national centers, which are at Tier1. Below that are the regional centers (RC) called Tier2 and each RC covers a large part of a country. Next in the hierarchy are Tier 3 centers**,** which are workgroups. Finally there is Tier 4, which consists of thousands of desktops.

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 these huge data sets among the various scientists in the grid. The main aims of using replication are to reduce access latency and bandwidth consumption. The other advantages of replication are that it helps in load balancing and improves reliability by creating multiple copies of the same data. Static replication can be used to achieve some of the above-mentioned gains but the drawback with static replication is that it cannot adapt to changes in user behavior. The replicas have to be manually created and managed if one were to use static replication. 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. Dynamic strategies have the ability to adapt to changes in user behavior. Our study examines different dynamic replication strategies for a grid

The three fundamental questions any replica placement strategy has to answer are: When should the replicas be created? Which files should be replicated? Where should the replicas be placed? Depending on the answers, different replication strategies are born. We use simulation to evaluate the performance of each different strategy. Since most of the datasets in the scientific data grid scenario are read-only, we do not consider the overhead of updates in our dynamic replication strategies.

2 The Simulator

To carry out the studies 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, a discrete event simulation tool to model events like file requests and data transfers. The simulator consists of three parts. The basic component is the one that 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 basic component and compose of the next layer. The final component is the driver entity of the program. This is where the triggers for various file requests originate. The driver entity reads an input file, which contains the access patterns to be simulated.

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 the messages and files can travel to get to the destination. When a file is being transferred thru 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.

Access Patterns: The simulation was first run on access patterns that were generated randomly. This being the worst-case scenario, more realistic access patterns that contained varying amounts of temporal and geographical locality were generated.

- P-random: Random access patterns. No locality in patterns.
- P1: Data, which contained a small degree of temporal locality
- P2: Data containing a small degree of geographical and temporal locality

Performance Evaluation: Different replication strategies need to be compared using the simulator. We measuring two parameters, the average response time and the total bandwidth consumed to accomplish this. Response Time 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 includes the bandwidth consumed for both the data transfer that occurs when a node requests a file and when a server creates a replica at another node. As is intuitive, the best replication strategy will have both lower response times and lower bandwidth consumption. All replication strategies need not exhibit an equal amount of decrease in both. In fact there could also be an increase in bandwidth consumption and a decrease in response time or vice versa. Tradeoffs can be made depending on whether bandwidth or response time is the constraint in the particular grid scenario.

3 Replication/Caching Strategies

We implemented and evaluated six different strategies. This helped demonstrate what the simulator is capable of doing, as well as helped us understand the dynamics of a grid system better.

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. This gives us the base performance and any strategy that performs worse than this is not worth considering.

Strategy 2: Best Client- Each node maintains a detailed history for each file that it contains, 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 number of 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 have their replicas created 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.

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 number of requests for the file is exceeded at Level 2 it is then replicated at the next lower tier and so on. A very popular file may ultimately be replicated at the client itself.

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. 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.

4 Results and Conclusions

We compare the results of five of the experiments, 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 plus caching. The experiments were run on the three access patterns: P-random, P1, and P2 and each simulation was run for a thousand requests.

When P-random data was used, all the 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. Again, in terms of bandwidth savings, all the strategies except Best-Client exhibit significant positive savings. 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 data (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 though the latency savings are significant.

We go on to discuss the results got by the other three experiments as Best-Client does not seem a good candidate for a replication strategy for a grid. The two graphs below, contains 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 Figure 1 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. 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.

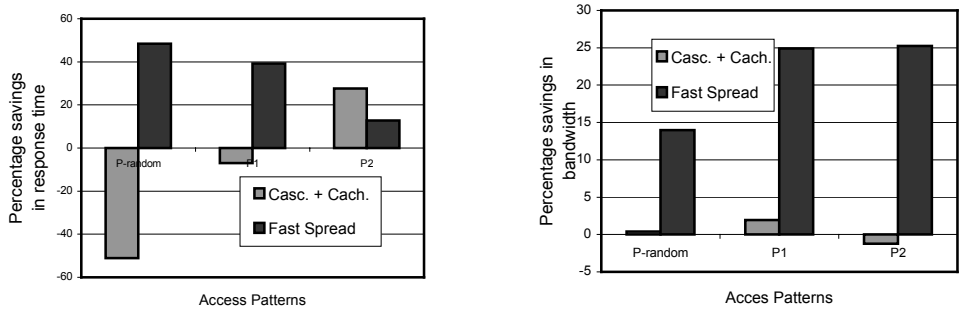

Figure 1: Percentage savings in response time and bandwidth consumption as compared to Plain Caching

In terms of bandwidth consumption Cascading does not differ significantly from caching. 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. These results lead us to conclude that if the grid users exhibit total randomness in accessing data then the strategy that would work best is Fast Spread. If however there were 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, using Cascading results in lowering the 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 were of top priority, Fast Spread would be a better grid replication strategy.

Acknowledgements

This research was supported by the National Science Foundation's "GriPhyN" project.

References

[1] 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," Network Storage Symposium, Seattle 1999.

[2] A. Bestavros and C. Cunha, "Server-initiated document dissemination for the WWW," in IEEE Data Engineering Bulletin, vol. 19, 1996, pp.3 –11.

[3] J. Gwertzman and M. Seltzer, "The case for geographical push-caching," presented at 5th Annual Workshop on Hot Operating Systems, 1995.