

# Data Replication and the Storage Capacity of Data Grids

Silvia Figueira<sup>1</sup> and Tan Trieu<sup>2</sup>

<sup>1</sup>Department of Computer Engineering  
Santa Clara University  
Santa Clara, CA 95053-0566, USA  
sfigueira@scu.edu

<sup>2</sup>SGI, 1140 E. Arques Avenue  
Sunnyvale, CA 94085, USA  
tantrieu@sgi.com

**Abstract.** Storage is undoubtedly one of the main resources in data grids, and planning the capacity of storage nodes is an important step in any data-grid design. This paper focuses on storage-capacity planning for data grids. We have developed a tool to calculate, for a specific scenario, the minimum capacity required for each storage node in a grid, and we have used this tool to show that different strategies used for data replication may lead to different storage requirements, affecting the storage-capacity planning.

**Keywords:** Data Grids, Storage Capacity, Capacity Planning, Data Replication

## 1 Introduction

Grid computing addresses the challenge of coordinating resource sharing and problem solving in dynamic, multi-institutional virtual organizations [3]. Though we often think of processor power as a primary motivation for grid computing, access to distributed data is typically as important as access to distributed computational resources. In fields such as high-energy physics, biology and medical image processing, and earth observations, experiments can produce massive amounts of data, on the scale of petabytes per year [4]. The global sharing of such large amounts of data introduces access, processing, and distribution difficulties. With the massive size of the data, the task of managing and distributing the data quickly becomes a problem. Data grids have been developed to facilitate the efficient storage and quick distribution of this data [11].

A data grid connects a collection of geographically distributed computer and storage resources, enabling users to share data and other resources in a seamless fashion. The European Data Grid project is an effort to achieve the vision of uniform and transparent access to data and computing resources [10]. The vision is a computing environment in which a scientist who wants to run a computationally intensive process on a huge data set has several options. If there are sufficient computing resources on the local network, the scientist should be able to download

the data to a local destination and perform the job locally. If the local site lacks the necessary computing resources, the scientist could choose to off-load the processing to the data site. However, if the data site is under heavy load, the scientist could also request that the data be replicated to another site, and run the job on that site.

The design of a data grid requires forethought concerning the capacity of resources in order to avoid over-provisioning and/or under-provisioning at different locations of the grid. Capacity planning is defined as the process of assessing the cost/benefit of a system configuration before actually building it [1, 5, 6]. A useful prediction necessarily considers the evolution of the workload on the data grid. With capacity planning, system administrators can input different network topologies, while varying different resource allocations and other parameters (such as network link capacities, node storage capacities, and data replication strategies) in order to fine-tune the data grid's resource utilization.

Since one of the main resources in data grids is storage, planning the capacity of storage nodes is a key part of a data-grid design. This paper focuses on storage-capacity planning for data grids. We have developed a tool to calculate, for a specific scenario, the minimum capacity required for each storage node in a grid. The scenarios are defined by the network topology and the workload characteristics. The tool combines the topology with the load characteristics to produce guidelines on the minimum storage capacity required in each storage node to provide enough space to the workload considered. We have used this tool to compare data-replication strategies and show that they affect the storage required.

This paper is organized as follows. Section 2 discusses data replication. Section 3 presents a strategy for calculating storage capacity taking into account data replication. Section 4 shows how data replication affects the need for storage in data grids. Section 5 concludes.

## **2 Data Replication**

Data replication is a technique used in grid computing to both decrease access time to data and increase fault tolerance. Replication is especially important when considering requests for transferring massive amounts of data between geographically distant locations, which consume large amounts of bandwidth and are delayed by possibly high latencies [8, 9].

With the goal of reducing access latency and bandwidth consumption, recent research has addressed the usefulness of creating replicas to distribute data among scientists within a grid environment. Data replication can be managed statically or dynamically. Though static replication plans do improve load balancing and reliability, it does not adapt to changes in load behavior. Since data grids are intended for a global computing environment, where variable data-access patterns are expected, dynamic replication is preferred. In dynamic replication, the replication strategies adapt to changes in user behavior, making heavy use of locality in determining where files should be replicated. Decisions are made based on the notion that recently accessed files are more likely to be accessed again (temporal locality), files recently accessed by a node are likely to be accessed by nearby nodes

(geographical locality), and files near recently accessed files are likely to be accessed (spatial locality).

In [8], the authors discussed and evaluated six replication/caching strategies:

- No replication or caching.
- Best client: Nodes maintain an access history for each file, and when a threshold number of accesses is reached, review the history to determine the replica destination.
- Cascading replication: Take advantage of the hierarchical layout of a grid by replicating the requested file along the path from server to requester. A replica is made to the next node on the path to the requester only after a threshold number of requests have been recorded at the data site.
- Plain caching: Simply store a local copy on the requesting node.
- Caching plus cascading replication: The requester caches files locally and the server periodically identifies popular files to propagate down the network hierarchy.
- Fast spread: Requested files are stored at each node along the path from server to client.

The results of the study indicate that among the six replication/caching strategies, cascading and fast spread provide the best overall performance, where performance is measured as savings in latency and bandwidth consumption. The distinction between the two strategies is that fast spread works well in a network where users exhibit total randomness in accessing data. Cascading is the best option when access patterns exhibit geographical locality.

### 3 Calculating Storage Capacity

The data-grid capacity planner will take two main inputs: (1) the network topology, which specifies how the storage nodes are connected, and (2) a synthetic trace, which simulates requests for data. Based on these two inputs, it simulates the requests on the topology, tracking the storage usage at each node, according to a replication strategy. The main goal of the tool is to calculate the maximum storage capacity needed at each storage node after all the requests were processed. The output is a table detailing storage consumption for the nodes.

The requests will determine the data to be copied from the source to one or more nodes, according to the replication strategy. We propose the following scheme to characterize the requests:

- The source nodes, in which data is generated and from which it is disseminated, are determined initially. This information is provided by the user, who also determines the size of the data initially placed in each node and a number of tags to identify portions of each data. Each tag is associated with a size as well.
- Another parameter provided by the user is the lifetime of each request, which represents the time interval during which data should be useful for the destination

node. This information could be a constant value or generated uniformly within a range provided by the user.

- Each request will have a source node, a destination node, a tag, and a lifetime.

The algorithm for calculating the capacity is shown below. After the initial data is assigned to each source node, as determined by the user, and each source node has its current capacity updated, the algorithm starts processing the requests. A request will identify a source, a tag, a destination, and a lifetime. According to the replication strategy, the algorithm will decide which nodes need to receive the data. Note that, together, the source and the tag identify a piece of data, which should not be replicated in a node. In each node, a piece of data is identified by the source node, the tag, and the lifetime of the request. If a piece of data is supposed to be copied to a node, which already has that particular piece of data, the copying is avoided, and the lifetime of the data is updated to reflect the latest request.

```
Calculate_Capacity ( )
begin
  for each node i
    max_capacity of i = curr_capacity of i = initial capacity of i
  for each request for data <s, tag> from s to d, with lifetime l and timestamp ts
    eliminate old data from nodes, according to ts
    update curr_capacity for the nodes which had data eliminated
    for each node k receiving a copy of the data
      if node k already has data <s, tag>
        update the lifetime for <s, tag> in node k
      else
        update curr_capacity of k
        if (curr_capacity of k > max_capacity of k)
          max_capacity of k = curr_capacity of k
        end if
      end for
    end for
  end
end
```

The input and output values for the algorithm are described in the sub-sections below.

### 3.1 Inputs

#### Network Topology

The network topology is read from a file. Each line in the file represents a node in the network, and is described by the node's neighbors and the associated cost between the node and each neighbor. Data centers can be identified by specifying, in a separate file, the tags and those tags' corresponding sizes.

### **Traffic Trace File**

A synthetic trace for data requests represents the workload of a given topology. We use a traffic simulation tool, Flexible Optical Network Traffic Simulator (FONTS) [7], to generate a trace of data requests that simulates on-demand and advance-reservation requests with different stochastic characteristics. FONTS is based on a stochastic model that incorporates a variety of variables to model data transfer behavior of applications requiring sustained bandwidth. FONTS can be used to model bulk data transfer within the grid infrastructure, and thus is suitable for our use. Though the program allows the user to fine-tune numerous simulation parameters, we are interested in configuring the request type (advanced-reservation or on-demand), source node, and destination node. The user simply generates a trace with FONTS, and that trace can be passed directly to the capacity planner for interpretation.

### **Replication Strategy**

As the results in [8] show that only two of the six proposed replication/caching strategies demonstrate practical viability, for the moment, the capacity planner just considers these two replicating options: cascading with caching and fast spread.

## **3.2 Outputs**

The storage-capacity planner produces as output a table summarizing storage consumption in the network. Specifically, the report shows the number of requests serviced and the initial, maximum, and current capacity at each node.

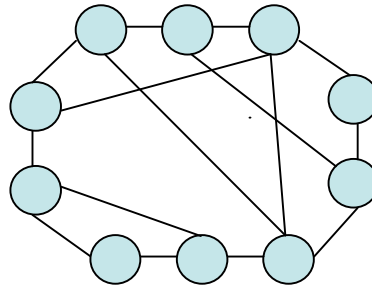
## **4 Experiments**

Our storage-capacity planner enables a study of the effect of different data replication strategies on storage consumption. It is hypothesized that data networks replicating with fast-spread will consume more resources than networks using cascading with caching. In particular, as network size increases, data in fast-spread networks distribute more quickly and therefore the overall capacity to accommodate the rapid distribution will quickly outgrow that of cascading networks.

An experiment was designed to test the hypothesis. The experiment uses a topology typical of grids—the ring with chords. The topological variables in the experiment include: the size of the network, measured by the total number of nodes, and the degree of interconnectedness, measured by the number of chords. The number of nodes in the networks studied ranged from 2 to 20 nodes. The networks were also determined by the interconnectivity parameter, where the 2- to 10-chord networks were simulated. The chords were randomly placed. A sample 10-node ring with 5 chords as used in the experiment is shown in Fig. 1.

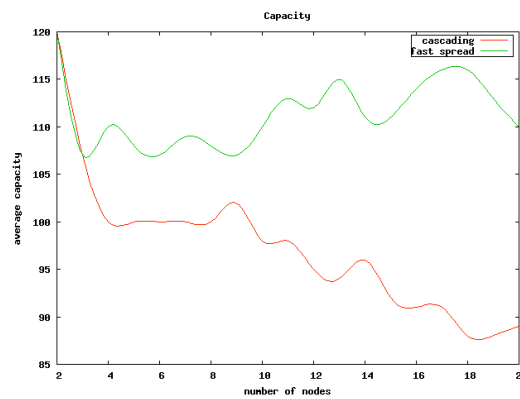
As for the input traffic traces, advanced reservation requests can be generated with FONTS. Relevant configurable variables for the FONTS traces include a uniform distribution for the source and destination of a request and the number of switching nodes (to match the topology under study). The final and most important variable is

the replication strategy used, either fast-spread or cascading with caching. Results from the simulations can be used to quantify either the substantiation or contradiction of the hypothesis. Graphs should also be plotted to visualize potential trends from the collected data that otherwise might have been overlooked in the hypothesis.

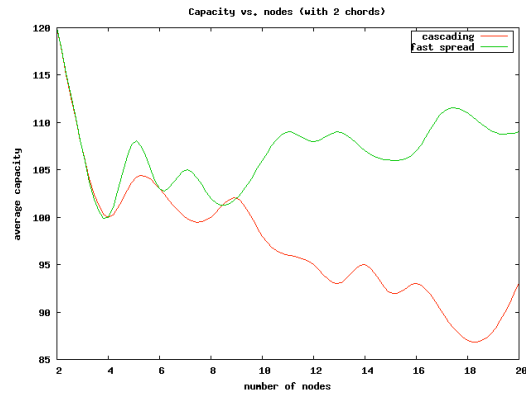


**Fig. 1** – 10-node ring with 5 chords.

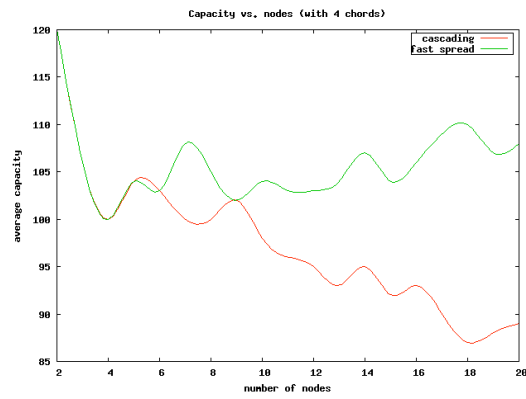
The storage-capacity planner output is a table detailing the initial, current, and maximum storage resources consumed at each node in the network during the simulation. We aggregate the results into a succinct and meaningful mathematical value so that the simulated network configurations can be easily compared to each other. That mathematical value is obtained by averaging the maximum storage consumed, which provides a useful index for evaluating the overall distribution of maximum resources utilized. The calculated indices are plotted as average capacity versus the number of nodes in the simulated network. The number of chords is also varied and plotted. A representative set of these experiments is shown in Fig. 2 to Fig. 6.



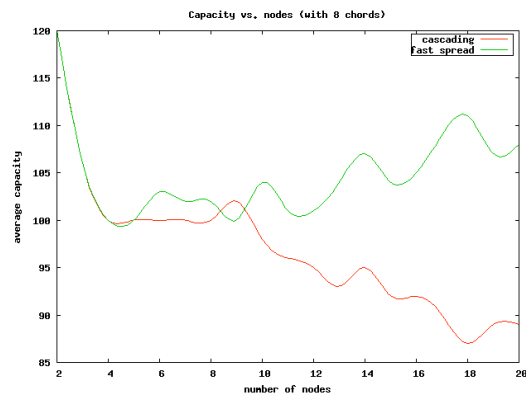
**Fig. 2** – Ring with no chords.



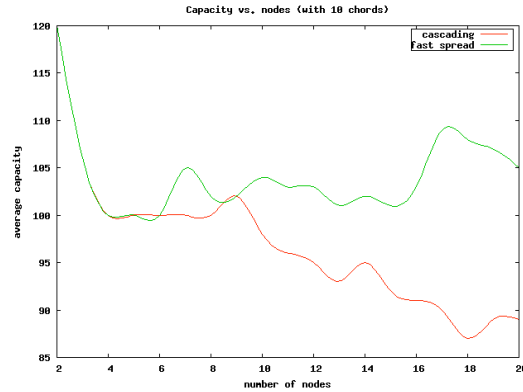
**Fig. 3** – Ring with 2 chords.



**Fig. 4** – Ring with 4 chords.



**Fig. 5** – Ring with 8 chords.



**Fig. 6** – Ring with 10 chords.

The trends observed in the graphs confirm the hypothesis—that is, fast-spread replication on average consumes more storage resources than cascading with caching. With increasing network size, the storage utilization gap between the two replication strategies widens. The results correspond to the intuition that, if we more readily replicate, then the overall storage spending increases accordingly. When there are more nodes in the network, since the fast-spread replication algorithm distributes data to nearly all the nodes, the average storage usage shows a relatively uniform distribution. However, the more conservative cascading algorithm replicates only after a predetermined threshold number of requests have been reached. As network size increases, a smaller fraction of the nodes in the network will contain replicated data. Therefore, with an increasing network size, the average storage utilization shows a decreasing (somewhat linear) trend. The fast-spread’s somewhat constant average capacity, together with cascading’s decreasing average capacity, explains the widening gap in resource utilization between the two replication strategies.

An unanticipated result is the overlapping capacity indices between the replication strategies when there are approximately five to nine nodes in the network. The observed trend sharpens with an increased interconnectivity. This finding provides valuable insight for data-grid designers, because it mitigates the often complicated tradeoff between time (data access speed) and space (storage resources). If the network is relatively small, such that there are no more than ten nodes, and some interconnectivity exists, then the tradeoff is unnecessary. Fast-spread data replication will consume on average the same amount of capacity as cascading while providing low access latency. However, as the number of nodes increases, the network designer must more carefully consider the time and space tradeoff. If access speed is critical, we must endure the increased storage consumption for low latency access. If the budget for storage resources is restricted, then use cascading replication to conserve capacity. There exists also demand for relatively low latency, but with a limited storage budget. Such a scenario requires an average capacity that lies somewhere between that of fast-spread and cascading, which may be achieved by a combination of the two replication strategies.



## 5 Conclusion

Data replication has been explored as a performance-tuning parameter for data grids, wherein two replication strategies (fast-spread and cascading with caching) have been identified to effectively reduce access latency or bandwidth consumption. A storage-capacity planner was developed to help data-grid designers better gauge the cost (in terms of storage resources) for fast-spread and cascading replication. An experiment was designed to estimate capacity consumption of the two replication strategies on ring networks. The results of the experiment coincided with the intuition that fast-spread networks consume more resources than cascading networks. The time and space tradeoff in network design was discussed, and a hypothesis was made that certain scenarios may require the use of both replication strategies in order to achieve good latency with moderate, or limited, storage resource funds. As future work, this feature can be incorporated into the storage-capacity planner, and a new experiment conducted to verify if a network combining both replication strategies would indeed yield the expected capacity consumption.

## References

1. Andy Hospodor, "Capacity Planning of Commercial and Scientific Grids with GridPlan", *GlobusWORLD*, 2006.
2. S. Figueira, N. Kaushik, S. Naiksatam, S. A. Chiappari, and N. Bhatnagar, "Advance Reservation of Lightpaths in Optical-Network Based Grids," *Proceedings of the ICST/IEEE Gridnets*, 2004.
3. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid - Enabling Scalable Virtual Organizations," *International Journal of High-Performance Computing Applications*, 2001.
4. GriPhyN - Grid Physics Network, <<http://www.griphyn.org/>>, 2005.
5. K Ting, T. Liu, L. Tibbets, and S. Figueira, "CapPlan - A Network Capacity Planning Tool for LambdaGrids," *Proceedings of the IARIA/IEEE ICNS*, 2006.
6. D. A. Menasce and V. A. Almeida, *Capacity Planning for Web Services*, Prentice Hall, 2001.
7. S. Naiksatam, S. Figueira, S. A. Chiappari, and N. Bhatnagar, "Analyzing the Advance Reservation of Lightpaths in Lambda-Grids," *Proceedings of the IEEE/ACM CCGRID*, 2005.
8. K. Ranganathan and I. Foster, "Identifying Dynamic Replication Strategies for a High-Performance Data Grid," *Proceedings of the International Workshop on Grid Computing*, 2002.
9. H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney, "File and Object Replication in Data Grids," *Proceedings of the 10th HPDC - International Symposium on High Performance Distributed Computing*, 2001.
10. The DataGrid Project, European Union, <<http://eu-datagrid.web.cern.ch/eu-datagrid/>>, 2005.
11. The Globus Data Grid Effort, <<http://www.globus.org/toolkit/docs/2.4/datagrid/>>, 2005.