Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities

Kavitha Ranganathan, Adriana Iamnitchi, Ian Foster
Department of Computer Science,
The University of Chicago, Chicago, IL 60637
{krangana, anda, foster}@cs.uchicago.edu

Abstract

Efficient data sharing in global peer-to-peer systems is complicated by erratic node failure, unreliable network connectivity and limited bandwidth. Replicating data on multiple nodes can improve availability and response time. Yet determining when and where to replicate data in order to meet performance goals in large-scale systems with many users and files, dynamic network characteristics, and changing user behavior is difficult. We propose an approach in which peers create replicas automatically in a decentralized fashion, as required to meet availability goals. The aim of our framework is to maintain a threshold level of availability at all times.

We identify a set of factors that hinder data availability and propose a model that decides when more replication is necessary. We evaluate the accuracy and performance of the proposed model using simulations. Our preliminary results show that the model is effective in predicting the required number of replicas in the system.

1. Introduction

We are interested in using the aggregate storage capacity of large numbers of geographically distributed personal computers to store large scientific data sets. Such a peer-to-peer (P2P) storage system can, in principle, offer low cost, large capacity (a million PCs can provide 1-10 petabytes of storage today), access to significant collocated computation power, and high aggregate access bandwidth.

Yet the design and implementation of a P2P storage system also raises significant challenges. The average reliability of any single storage element in such a system will be low, due to unreliable networks and the possible departure of its associated node. Similarly, the data access performance offered to clients will be variable, depending on often limited bandwidth and the popularity of the data item in question. Yet the scientific applications that are of interest to us want guarantees that data will be available when they need it—at least with high probability.

A well-known technique for improving availability in unreliable systems is replication. If multiple copies of data exist on independent nodes, then the chances of at least one copy being accessible are increased. Aggregate data access performance will also tend to increase, and total network load will tend to decrease, if replicas and requests are reasonably distributed.

Yet while replication has advantages, it also has significant costs. We need a mechanism for creating replicas that allows us to meet availability and performance goals without consuming undue amounts of storage and bandwidth. We would like this mechanism to function entirely automatically. And it must function effectively in a dynamic, decentralized P2P environment.

We describe here a mechanism with these properties. In our approach, each peer in the system possesses a (necessarily highly approximate) model of the P2P storage system that it can use to determine how many replicas of any file are needed to maintain desired availability. Each peer applies this model to the (necessarily incomplete and/or inaccurate) information it has about system state and replication status of its files to determine if, when, and where new replicas should be created. The result is a completely decentralized system that can maintain performance guarantees.

We evaluated our approach with three different simulation experiments to (1) check the accuracy of our model; (2) compare our dynamic model approach to a static replication scheme; and (3) gauge the effects of the decentralized decision making process.

The use of a system model to guide replica creation clearly distinguishes our work from Web caching systems [1], [2], [3] and from file sharing systems such as Freenet [4], in which data is replicated entirely on the basis of popularity. Those systems do not address availability issues. The OceanStore system [5] is concerned with availability, but does not model system behavior. Our cost models of replica creation and placement can be used by such a system to ensure data availability.

The rest of this paper is as follows. Section 2 presents a number of data intensive applications that can benefit from dynamic data management. We present in Section 3 our dynamic, adaptive replica management approach, and describe in Section 4 the analytical mechanism that supports our solution. Our solution relies on two basic services: resource discovery and replica location. Section 5 presents some of the current designs for both. In Section 6 we describe our simulations results and conclude with future directions in Section 7.

2. Target Applications

A typical example of a data sharing collaboration is a community of scientists who want to perform computationally demanding analyses on large amounts of data. The output of their analyses creates new data that they then want to circulate among their colleagues across the world. Both the experimental data and the new derived data are read-only, as is, for example, the case of the Compact Muon Solenoid [6] experiments that will start at CERN in 2006.

Many applications in the scientific community, spanning biology, astrophysics, astronomy, and genetics, fit nicely into target applications that deal with huge data sets and need some amount of data management.

The *Human Genome Project [7]* constructs detailed genetic and physical maps of the human genome. The project needs advanced means of making new scientific data widely available to scientists so that the results may be used for public good.

Telescopes like that in the Sloan Digital Sky Survey experiment [8] will scan vast amounts of the sky and generate large amounts of data every night. Data often needs to be processed during the next day, which requires fast dissemination to powerful computational resources.

The *Human Brain Project* [9] consists of collaborations spanning different fields (computer graphics, molecular biology, digital optical microscopy, modeling, and control theory) that require high resolution, multi-dimensional images of the nervous system in several models. This project involves sharing of large datasets among a potentially large set of participants.

A popular application that has boosted interest in the P2P concept is the sharing of mp3 files by geographically distributed users. Music sharing environments are more static than scientific environments: the use of data by scientists often leads to creation of new data. In addition, the number of distinct music files is smaller and there is no attempt to ensure availability of any particular file.

3. Dynamic Replica Management

In the solution we propose, each peer uses a set of tools to obtain a (typically partial and inaccurate) understanding of the state of the system and takes file replication and migration decisions. The system works as follows.

Each node in the network is authorized to create replicas for the files it stores. A node decides where to replicate a file using a performance model that compares the costs and the benefits of creating replicas of a particular file in certain locations. (Section 4.3 discusses the factors that trigger a node to evaluate the opportunity of file replication.) Replicas are deleted according to the local policy of the host node.

Our model-driven approach relies on a resource discovery service to find available storage and on a tool such as the Network Weather Service [10] to provide network availability and prognosis information.

The parameters we consider in our replication decision model are:

1. Single-system stability *p*, which encompasses node failures, communication failures that render the node unreachable, and the departure of the node from the network. In our model, this parameter is expressed as the average probability of a node being up.
2. The transfer time between nodes N_1 and N_2 for the file *F* to be replicated $trans(N_1, N_2, F)$. This parameter is defined as the ratio between file size and available bandwidth.
3. Storage cost of file *F* at a given node *N*: $s(F, N)$. This parameter captures the cost of writing new data to the storage. The storage cost can include local policies, from data replacement mechanisms to acceptance to store specific data. For example, the storage cost published by a node that does not agree to store some data can be infinity. A node that does never replace a replica within an hour after its creation can also advertise a very high storage price, allowing it not to be chosen to host new data. This cost can also depend on the size of the file to store, or incentives (if we consider a market model).
4. The accuracy of the replica location mechanism RL_{acc}. The decision of creating more replicas depends on the number of currently existent replicas. However, this number may not always be accurately determined, given the system's dynamism and inherent communication delays. We hence take into account a less than perfect accuracy of the replica location service. The accuracy is measured through past performance of the mechanism or simulations in a controlled environment.

The model-driven approach we propose aims to help nodes answer two critical questions for replica management.

1. What is the optimal number of replicas for a file?
2. Which is the best node to host a particular replica?

These questions need to be answered for an externally defined availability threshold for each file.

4. Analytical Mechanisms

Our proposed approach has two objectives. First, we want to express the number of replicas per file as a function of the parameters that influence availability. Second, we want to provide a function that evaluates the placement of a replica in a particular location. This function can then be used for choosing the best alternative.

4.1. Computing the number of replicas per file

To determine the number of replicas that guarantees the required availability of a file, we need to consider the system parameters that affect availability and performance.

The availability of a file depends on the failure rate of peers in the network and the accuracy of the replica location service. If a large number of peers are often unreachable, then a large proportion of files may become unavailable. Aggressive replication is one method to maintain the desired level of availability in such environments.

The accuracy of the replica location service determines the percentage of accessible files: if the location service is ineffective, more replicas need to be created to ensure that at least some are retrieved.

We develop a function to calculate the number of replicas needed (*r*) for a certain availability threshold. Since the availability of a file depends on the failure rate of peers in the network and on the efficiency of the Replica Location Service (RLS), a simple function can be developed as follows.

Let *r* be the total number of replicas for a file A,
p be the probability of a node to be up,
RL_{acc} be the accuracy of the replica location service,
Avail be the required amount of availability for file A.

Then, the probability of all *r* replicas of file A being unavailable is:

$$(1-p)^r$$

Therefore, the probability of at least one replica of A being available is:

$$1-(1-p)^r$$

and the probability that a replica of A will be found is:

$$RL_{acc}*(1-(1-p)^r)$$

(We assume that the availability of a file and the RLS accuracy are independent).

Therefore, we need

$$RL_{acc}*(1-(1-p)^r) \geq Avail$$

to ensure the amount of availability needed for a given file. The desired value for *r* can be calculated from the above function for any given availability threshold. For example, for a probability *p* of 30% of a node to be up, an availability threshold of 80% and the RLS accuracy of 75%, the model recommends a minimum of 5 replicas. If *p* is 1%, the recommended number of replica increases to 160.

Once a node knows the ideal number of replicas *r* for a file, it employs the replica location service to discover how many actually exist. Let us assume the replica location service returns *M* and *M* is less than *r*, then the node knows that it has to create (*r* – *M*) copies of the file and distribute them to remote locations.

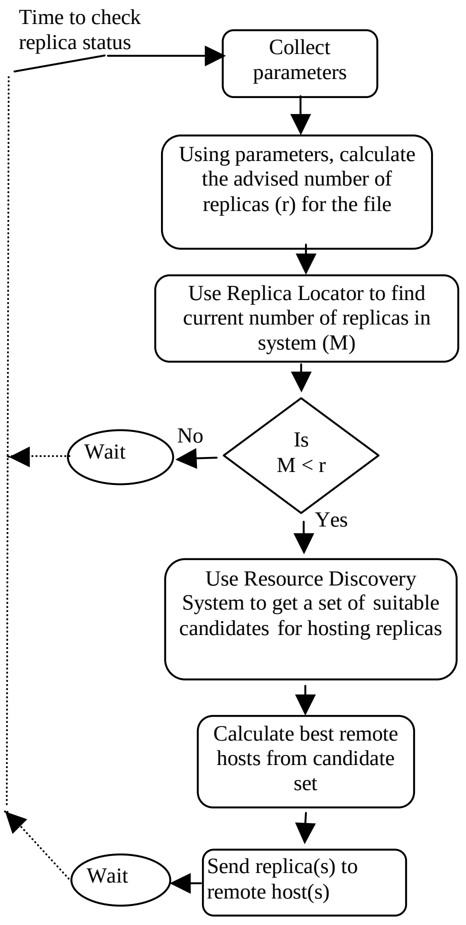
Time to check replica status
Collect parameters
Using parameters, calculate the advised number of replicas (r) for the file
Use Replica Locator to find current number of replicas in system (M)
Is M < r
No
Wait
Yes
Use Resource Discovery System to get a set of suitable candidates for hosting replicas
Calculate best remote hosts from candidate set
Send replica(s) to remote host(s)
Wait

Figure 1: Set of actions taken by a node for a file in the P2P system

If the number of existent replica is larger than needed ($M \geq r$), the node does nothing, for as we mentioned earlier, the extra replicas will eventually be replaced by files more "interesting" to the local host.

Since each peer in the system acts independently, there is considerable chance that two peers simultaneously take the decision of replicating the same file. If storage resources are scarce, then the extra-replicas will soon be replaced with others. If storage resources are abundant, then the excessive replication is not necessarily harmful.

4.2. Determining the location for a new replica

We assume that the resource discovery mechanism provides a set of candidate storage resources located in different domains for a new replica host. A domain is a collection of nodes in a particular geographical area. We assume that any two locations within a particular domain will have the same or similar transfer and storage costs owing to their geographical locality.

The candidates returned by the resource discovery service meet the following criteria: they do not contain a copy of the file to be replicated, they have available storage (or at least replaceable files), and have a reasonable transfer time (below a certain maximum) to the potential users.

Once the node has the possible set of host candidates, it uses a heuristic to select the best candidate(s). The best candidate maximizes the difference between replication benefits and replication costs. The benefit is the reduction in transfer time to the potential users. The replication costs are the storage cost at the remote site and the transfer time from the current location to the new location.

The cost of creating a replica at a location N_2 for a file F stored at the location N_1 is: $s(F, N_2) + trans(F, N_1, N_2))$, where,

N_1 = Node that currently has the file
N_2 = Candidate node for new replica
$s(F, N)$ = Storage cost for file F at node N
$trans(F,a,b)$=transfer cost between locations a and b

The benefit of creating a replica at N_2 is:
$trans(F, N_1, User) – trans(F, N_2, User)$, where *User* is the location from which we expect the most number of future requests. Therefore, the net benefits of replicating at N_2 can be calculated and the best candidate identified.

4.3. When to Check

An important problem to be addressed is what triggers a node to check if more replicas are needed for a file.

A possible solution is to compare periodically the number of existent replicas with the required number of replicas. The periodicity of these checks can be altered to suit network volatility and usage patterns. If, for example, during the last three checks there was no action needed on the part of a node, it can increase the time interval of its checks (and hence use fewer system resources). If, on the other hand, consecutive checks show that more replicas are needed, the node may want to increase its frequency of checks to adapt to the changing surroundings. One disadvantage of this approach is the overhead created by frequently using the replica location service: for each file a node has, the location service is called, independent of the file's popularity. An alternative is to check the level of replication of a file only when that file is requested.

However, in both scenarios, multiple nodes could simultaneously create replicas for the same file. Though the extra replicas will eventually be overwritten, there still remains the cost of wasted bandwidth and the problem of efficient resource utilization. Since we cannot rely on a central decision-making authority, such conditions are hard to eliminate. Section 6.2 elaborates on this.

5. Resource discovery and replica location

The mechanisms presented in the previous sections rely on *resource location services* for locating replicas and *resource discovery services* for locating available storage and network resources. We briefly present the state of the art for these two mechanisms.

A basic service in many wide-area sharing environments is resource discovery: given a description of resources desired, a resource discovery mechanism returns a set of (contact addresses of) resources that match the description. Resource discovery can be challenging because of system scale, heterogeneity, and dynamism. These characteristics create significant difficulties for traditional centralized and hierarchical resource discovery services. [11] evaluates some simple, fully decentralized resource discovery mechanisms. Search-efficient, hierarchical solutions were also proposed ([12])

There are already many solutions for locating files in P2P environments: CAN [13], Chord [14], Tapestry [15], Gnutella [16], Freenet [4], and Napster.

CAN [13], Chord [14], and Tapestry [15] build search-efficient indexing structures that provide good scalability and search performance at the increased cost of file insertion and removal. Gnutella does not use indexing mechanisms; its relatively good search performance (as measured in number of hops) is due to intensive network usage. Napster uses a centralized approach: a file index is maintained at a central location, while real data (files) are widely distributed. Freenet includes, in our terminology, both replica management and replica location mechanisms: popular files are replicated closer to users, while the least popular files eventually disappear. Freenet's file location mechanism is also built based on usage patterns, using dynamic routing tables. However, the Freenet approach assumes that non-popular data is unimportant data (and removes it), which is not a valid assumption for many scientific applications. We assume that files always exist at their source (since we are not concerned with anonymity of data).

6. Simulation Results

It is by no means clear that our proposed approach to replica management will work well in practice. Too many replicas could be created due to partial or incorrect information and due to multiple nodes acting simultaneously. The overhead of the state monitoring required to guide replica creation might be excessive. Or, replicas might not be created often enough, with the result that availability would fall below desired goals.

We simulated a 100-node P2P network with unlimited storage space. While the number of nodes does not influence the model's answer, it somewhat reflects the scale of the network. Nodes join and leave the network with a specified probability. We assume that nodes that fail or leave the network lose all the replicas they store. Each simulation is run for 100 steps. At each step:

- A certain number of nodes go down (depending on the probability of nodes being up)
- A percentage of the nodes that are up check for available replicas for the files they contain. If needed, they create more replicas.

We built our simulations with three objectives in mind: (1) evaluate the validity of our model; (2) evaluate the effect of partial information on decision accuracy; and (3) compare our dynamic replication strategy with a static strategy. We assume a replica location accuracy of 0.8 for all experiments. In Figures 2, 3 and 4 each plot corresponds to a different average node reliability.

6.1. Model Validation

Our model computes the minimum number of replicas that are necessary to achieve a certain availability threshold in the presence of node failures. One way to check the accuracy of our model's results is therefore to fix the number of replicas existent in the system at any time and measure data availability. Ideally, for the number of replicas equal to that returned by our model for a required availability X, we should measure an availability close to X. Figure 2 compares the numbers of replicas corresponding to matching availability values. For example, for the probability of 0.4 of nodes being up and a required availability threshold of 0.6, the model requires 3 replicas. When we maintain the number of replicas per file at 3, we measure the average availability per file to be around 0.65. Though the model predictions are not this accurate at all points in the graph, the simulations validate the general trend pointed to by the model. We note that further fine-tuning of the model is needed for better accuracy.

6.2. Effect of decentralized decision making

A potential drawback of the P2P mechanism we describe is that each node makes its own replication decisions. Extra replicas may get created in the event of more than one node replicating the same file simultaneously. To evaluate this effect, we record the total number of replicas per file in the system (after system warm-up). At each step, 25% of the nodes that are up check the system and create replicas if needed, simultaneously. The results of these runs along with the ideal number of replicas as calculated by the model are plotted in Figure 3.

When the probability of a node being up is moderately high (≥ 0.4), the negative effect of decentralized replication is not substantial: e.g., for probability = 0.4 and availability of 0.8, the model suggests 5 replicas/file, but we find an average of 7 replicas/file. When nodes are highly unreliable (e.g., probability=0.2), the number of replicas in the system is significantly larger than the number required by the model.

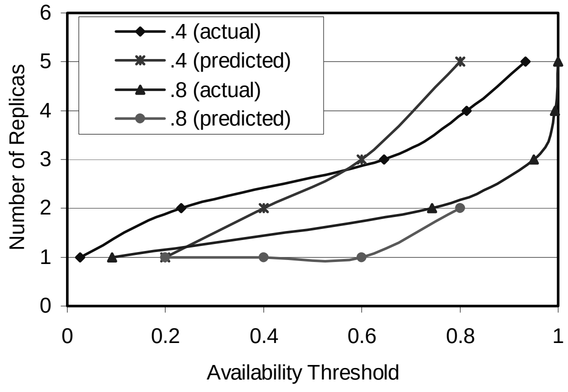

Figure 2: Model prediction versus actual behavior for different values of node reliability

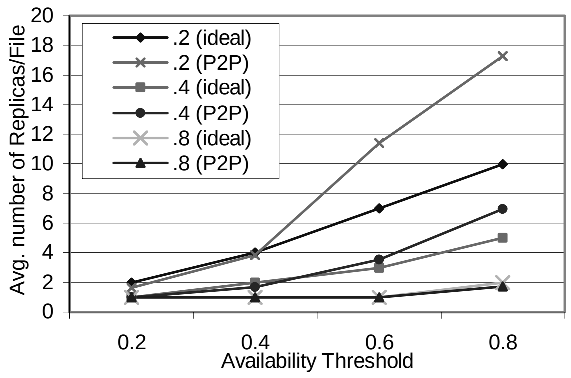

Figure 3: The effect of independent, decentralized replication decisions

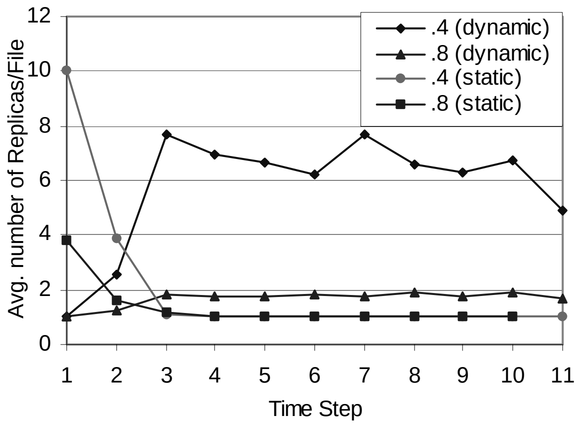

Figure 4: Static versus dynamic replication for different probabilities of a node being up and availability threshold of 0.7

6.3. Static versus Dynamic schemes

Finally, we compare the performance over time of a static replication scheme with our dynamic scheme. For the static scheme, we initially placed a constant number of replicas for each file in the system. For the dynamic model, we started the simulations with only one replica per file. As Figure 4 shows, in the static scheme the replicas in the system eventually decrease to 1 copy (minimum) per file. In the dynamic scheme, the number of replicas is maintained at a constant level, thus ensuring a better, constant data availability over time.

7. Conclusions and Future Directions

We have proposed a decentralized model for dynamic creation of replicas in an unreliable peer-to-peer system. The aim of our model is to ensure a specified degree of data availability. Along with the analytical mechanisms used by our model, we discussed the advantages and disadvantages of our approach and presented some preliminary evaluation results based on simulations.

Our results show that the model we propose is able to predict the required number of replicas in the system with moderate accuracy. We plan to further fine-tune the model based on more simulation studies. The results also show that our adaptive scheme is more adequate to a dynamic environment than a static replication scheme.

Our decentralized, dynamic mechanism has no single point of failure, as it does not rely on a central monitoring agent. Even if a sizable portion of the network is down or the network becomes partitioned, our adaptive mechanism adjusts the number of replicas to the new conditions. Moreover, the overhead associated with the extra monitoring and computations are distributed over the system.

These advantages come at the price of accuracy: nodes take decisions based on partial information, which sometimes lead to unneeded replication. Simulation results show that the redundancy in action associated with distributed authority is more evident when nodes are very unreliable.

Acknowledgements

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

References

[1] S. Acharya and S. B. Zdonik, "An Efficient Scheme for Dynamic Data Replication," Brown University CS-93-43, 1993.

[2] M. Rabinovich and A. Aggarwal, "RaDaR: A Scalable Architecture for a Global Web Hosting Service," presented at The 8th Int. World Wide Web Conference, 1999.

[3] O. Wolfson, S. Jajodia, and Y. Huang, "An adaptive data replication algorithm," *ACM Transactions on Database Systems*, vol. 22, pp. 255--314, 1997.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," presented at ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, California, 2000.

[5] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage," presented at Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), Cambridge, MA, 2000.

[6] CMS: Compact Muon Solenoid: http://cmsinfo.cern.ch/Welcome.html/

[7] Human Genome Project: http://www.nhgri.nih.gov/

[8] Sloan Digital Sky Survey: http://www.sdss.org/sdss.html

[9] Human Brain Project: http://www-hbp.scripps.edu/

[10] R. Wolski, "Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service," in *Proc. 6th IEEE Symp. on High Performance Distributed Computing*. Portland, Oregon, 1997.

[11] A. Iamnitchi and I. Foster, "On Fully Decentralized Resource Discovery in Grid Environments," presented at International Workshop on Grid Computing, Denver, CO, 2001.

[12] T. D. Hodes, S. E. Czerwinski, B. Zhao, A. D. Joseph, and R. H. Katz, "An Architecture for Secure Wide-Area Service Discovery," *Wireless Networks*, 2001.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," presented at SIGCOMM Conference, 2001.

[14] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," presented at SIGCOMM Conference, 2001.

[15] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," UC Berkeley, Technical Report CSD-01-1141, 2001.

[16] Clip2, "The Gnutella Protocol Specifications v0.4."

[17] K. Ranganathan and I. Foster, "Identifying Dynamic Replication Strategies for a High Performance Data Grid," presented at International Workshop on Grid Computing, Denver, CO, 2001.