The Stanford Archival Repository Project:

Preserving our digital past

Brian F. Cooper, Arturo Crespo and Hector Garcia-Molina

Department of Computer Science
Stanford University
Stanford, CA 94305
{cooperb,crespo,hector}@db.stanford.edu

Abstract

The Stanford Archival Repository Project aims to build a robust archiving system that can protect digital objects from failures over very long time spans. Objects are replicated among cooperating digital archives, so that if any archive fails its objects survive. We have designed an architecture for digital archives, and developed techniques for efficiently replicating objects to remote sites. We have also built simulation software that allows an archivist to design the most effective archiving system possible despite limited resources.

1 Introduction

The advancement of human knowledge critically depends on the preservation of knowledge gained in the past. Currently, more and more of that knowledge is stored electronically, in the form of files, databases, web pages, and programs. Unfortunately, most of the systems that store this information are not designed for preserving information over very long periods of time. Digital information can be lost not just through magnetic decay in storage devices, but also because of format and device obsolescence, human errors, and the disappearance of organizations. This problem will only get worse as more and more information is provided only in digital form.

Our approach to dealing with this issue is to build a reliable archival repository that protects digital information from failures. Users who create digital documents would deposit these documents in their local

repository. This repository would then take whatever actions were necessary to protect the documents. Our vision of an archiving system is built on several principles:

- *Write-once archiving*: Documents, once archived, are never modified or deleted. Updates to documents are represented as version chains.
- *Community-based replication*: A community of archives work together, storing copies of each others' data so that if a site fails, no documents are permanently lost.
- *Scientific design*: The policies for constructing and operating an archiving system should be studied scientifically, to find the most reliable and resource efficient techniques.
- *Robust operation*: The system should be robust in the face of failures. This means both that documents are preserved, and also that the system operations (document replication, content searches, and so on) continue to function.

Based on these principles, we have followed several research avenues. First, we have laid out the fundamental design of our archiving system, including the data model, digital object naming scheme, and basic components. Using this basic design, we have constructed a prototype archiving system, which we call the *Stanford Archival Vault* (SAV). While building and evaluating this system, we realized that there were many parameters that could impact the reliability of the system. For example, how many copies should we make of each document? How frequently should we check these copies for corruption? Should these copies be stored on a few expensive disks or many cheap disks? To answer these questions, we constructed a simulation system called *ArchSim*. This simulator allows us to try many different configuration options and find the parameters that give us the highest reliability.

Experience with SAV also convinced us that it was necessary to implement a robust mechanism for making remote copies of documents. Although multiple sites may be willing to cooperate and store each others' data, each site has limited resources and wants to make its own decisions about how to use these resources. We have developed a set of techniques, which we call *Data Trading*, which allow sites to negotiate with each other to determine how resources will be allocated for making copies. Once these negotiations are complete, copies of digital documents can be distributed to multiple sites.

In this paper, we describe the work we have done in investigating techniques for digital preservation. Our goal is to preserve the "bits" of digital documents from microscopic and macroscopic failures.

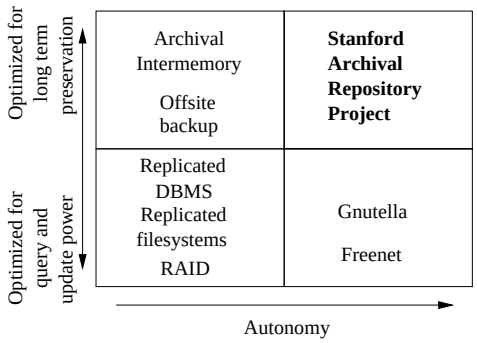

Figure 1: Classification of replicated data management

Our work complements other investigators' work in areas such as preserving digital formats, ensuring the security of digital objects against malicious users, encoding semantic meaning in digital documents, and so on. Moreover, much work has been done in the areas of distributed data mangement, data replication and fault tolerance. Figure 1 shows a schematic representation of how our work fits into the larger picture of replicated data management. The figure shows systems categorized by their focus on reliability and preservation versus efficiency (horizontal axis) and the degree of autonomy permitted to participating nodes (vertical axis). As the figure shows, our objective is to provide preservation, even if some efficiency is lost, while ensuring the autonomy of individual archives. Our focus in this paper is a survey of work done at Stanford. More related work is discussed in Section 5.

This paper is organized as follows. First, in Section 2, we present our design and prototype implementation for an archival repository. Next, in Section 3, we discuss ArchSim, the tool we have built to evaluate archive designs. Then, in Section 4 we discuss the techniques we have developed for effective negotiations between archives that store copies of each others' data. In Section 5 we examine related work, and in Section 6 we present our conclusions.

2 Designing and implementing an archival repository

2.1 A write-once digital archive

An archival repository assures long-term archival storage of digital objects. Digital objects are managed by *archive sites* on behalf of users who create and publish the objects. A federation of independent sites

collaborate to provide preservation by storing copies of each others' digital objects. If a site experiences a failure, the digital objects it is responsible for will not be permanently lost, because good copies will still exist at other sites.

There are several key components in this architecture.

- No deletions of digital objects

Once an object is archived it is never deleted. This is useful to prevent accidental erasure by human users. Moreover, if an object is missing, it can only be because of a failure, and should always be restored. If an object must be updated, the update is reflected as a new version in a chain, while the original version is preserved.

- Signatures as object handles

Digital objects are identified by *object handles*. In our architecture, object handles are constructed by computing a signature function over the content and metadata of the digital objects. This offers several advantages. First, copies of the same object always have the same name automatically. Then, we can efficiently determine if two objects are copies of each other merely by comparing their handles. At the same time, if an object becomes corrupted due to a failure, we can easily detect the corruption becaue the object will no longer match its own handle. Finally, a site can create an object and give it a globally unique name without having to consult other sites. This property requires that objects with different content have different handles with high probability, and for this reason we use a digest functions such as CRC or SHA-1 for the signature function.

- Disposable auxiliary structures

Data management requires auxiliary structures, such as indexes and mappings, to efficiently process objects. In our architecture, all such structures are disposable, and can be reconstructed at any time from the objects themselves. Because auxiliary structures may change often, it would be difficult to keep all replicated copies synchronized. If the structures are disposable, it is only necessary to replicate the digital objects, which never change.

This architecture is focused on prexserving the "bits" of digital objects. Other types of preservation (such as preserving formats or the semantic meaning of information) are also important, and can be provided by services layered on top of our architecture. For more details, see [9].

2.2 SAV: The Stanford Archival Vault

The Stanford Archival Vault (SAV) is an implementation of our Archival Repository design summarized in the previous section. In addition to demonstrating the feasibility of the Archival Repository architecture, it allows us to examine implementation issues such as efficiency and ease of use.

For example, we can most effectively ensure that documents are preserved if little or no effort is required on the part of users in order to add documents to an archive. Ideally, users would create information, and this information would be automatically archived. The InfoMonitor component of the SAV system provides such automatic archiving. Users create digital objects using their favorite applications, and save those objects to a standard filesystem. The InfoMonitor detects that an object has been created or modified, and in response creates a new archive object to preserve this information. The system has a filtering mechanism to allow an administrator to configure which files are archived. For example, it may be desirable to archive software sources but not personal email.

Once digital objects have been added to the archive, it is necessary to replicate them to other sites. A key component of this process is determining which objects should be replicated to which sites. A replication agreement governs which sites are involved (see Section 4.1) but it is also necessary to define the set of objects that will be replicated. If a new object is archived, it should be automatically replicated as part of an appropriate existing set. One inefficient way to achieve this would be to have an administrator manually assign objects to sets. We have designed a more flexible mechanism: the application archiving the object creates pointers to other related objects, and a set of related objects is replicated together. For example, an HTML file might link to an image, and this link is represented as an archive pointer when the image is archived. Then, whenever the HTML file is replicated to a remote site, the image will automatically be replicated as well. This allows sets of replicated objects to grow without human intervention whenever new objects are created.

We have conducted experiments to measure the performance of the SAV archive. Most of the time, the archive simply checks existing objects for corruption by comparing the object signatures to signatures calculated by other archives that have the same objects. Occasionally, a new object will be added, and the archive must add the object to its indexes and replicate the object to remote archives. We have tested these operations on an archive of the Stanford Database Group website, with up to ten gigabytes of data. Our experiments indicate that in our SAV implementation, all of these operations scale well as the number of

objects in the archive grows. For example, on an archive containing 270,000 digital objects representing 10 gigabytes of data, our system requires 21 seconds to construct a list of uncorrupted objects at an archive, transfer that list to a remote archive, and compare the list with the uncorrupted objects at the remote archive.

For more details about SAV, see [4].

3 ArchSim: A simulator for reliable archive design

ArchSim is a powerful simulation tool that aids in the design of archival repositories. ArchSim can model important configuration options, such as multiple formats, preventive maintenance, and failure distribution functions. By using specialized techniques, ArchSim is able to provide cost and reliability information for a configuration in a time frame that allows the exploration and testing of different policies. Moreover, ArchSim can efficiently perform large simulations involving many components and very long simulated periods. We believe that ArchSim can help librarians and computer scientists make rational and economical decisions about preservation, and help achieve better archival repositories.

3.1 Challenges in simulating an archival repository

Archsim builds upon existing simulation techniques for fault-tolerant systems. However, the unique characteristics of archival repositories make their simulation especially challenging:

- *Time Span:* The life of an archival system is measured in hundreds, perhaps thousands of years. This means that simulation runs will be extremely long, so special precautions must be taken to make the simulation very efficient. Furthermore, given these long periods, failure distributions must take into account component "wear-out." (A component is more likely to fail after 50 years that it is when new.) Simple failure distributions (e.g., exponentially distributed time between failures) are frequently used in fault-tolerance studies, but they cannot be used here since they do not capture wear out.

- *Repairs:* In an archival system we cannot in general assume that damaged components can always be replaced by new identical components (another common assumption when studying fault-tolerant systems). For example, after say 100 years, it may be impossible or undesirable to replace a disk with one having the same failure characteristics.

- *Component models:* Component models are fairly rich, compounding the number of states that must be considered. For instance, as we have discussed, a file is not simply correct or corrupted. Instead, it can be corrupted but the error undetected, it can be correct but not accessible for reads, and so on. The failure models in each of these states may be different, e.g., a file is more likely to be lost when being read.

- *Sources of failures:* A document can be lost for many reasons, e.g., a disk fails or a format becomes obsolete. Each of these failures has very different models and probability distributions. The approach of finding the "weak link" and assuming that all other factors can be ignored is not appropriate for archival repositories.

- *Number of Components:* An archive needs to deal with a large number of components and materializations.

To evaluate a possible archival repository configuration, we need to predict how well it protects documents and how much it costs. This prediction can sometimes be done analytically, but as the configuration gets more complex, an analytical solution is impractical (and sometimes impossible). Instead, we rely on a specialized simulation engine for archival repositories: ArchSim.

ArchSim receives as input an archival repository model (including costs), a stop condition (e.g., stop when the first document is lost), a simulation time unit (minutes, hours, days, etc.), and the number of repetitions. ArchSim outputs the mean time to failure (mean time to stop condition), a cost metric, and a confidence interval for both the MTTF and the cost metric. We are currently considering other output metrics, e.g., the fraction of the documents that are available after some fixed amount of time.

For defining the archive model, each distribution and policy is implemented as a Java object, so they can be as general as necessary. For example, for component repair, the corresponding Java module can simply use a probability distribution (perhaps one of the library functions described below) to generate the expected repair time. However, that module can easily be replaced by one that first decides if the component can be repaired (using one probability distribution), and then for each cases generates a completion time (when the component is repaired or the repair is declared unsuccessful).

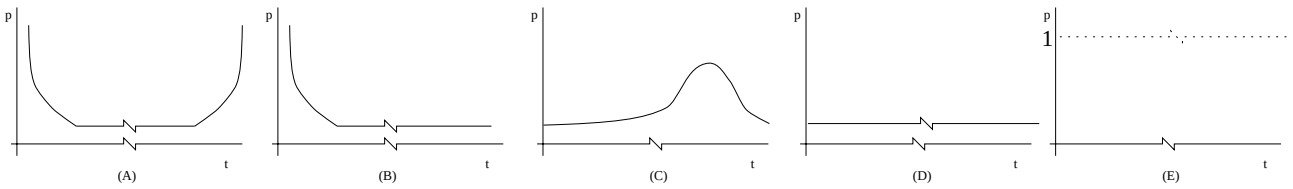

Figure 2: Possible failure functions in ArchSim

3.2 Library of failure distributions

ArchSim makes available a library of pre-defined failure distributions that can be used to describe archive components. The distributions in the library are: bathtub, infant mortality, historical survival, uniform, and deterministic. In Figure 2(a)-(e) we sketch generic versions of these probability distributions. For instance, in the bathtub distribution in (a), the instantaneous probability of failure early in the component's life (left on the horizontal axis) and late in its life (to the right) are higher than during the middle years. With the deterministic distribution (e), the component fails at a fixed time, where a spike is shown. (The area under these curves, from time 0 to t, represents the probability the component will fail by time t.)

3.3 Implementation of ArchSim

ArchSim follows the structure of a traditional simulation tool. Each module of the archival repository model can register future events in a timeline. For example, when a disk is created, the simulation uses the disk failure distribution to compute when the disk will fail; then, it registers the future failure event in the timeline. The simulation engine advances time by calling the module that registered the first event. This module may change the state of the repository and register more events in the timeline. Additionally, the module may contact the Cost Manager and record some cost involved with its operation. After the module returns, the simulation advances to the next event, in chronological order. The user can choose between two end conditions for the simulation: the simulation can stop after the first document is lost or after all the documents are lost. If the stop condition has occurred, the simulation stops and records the point on the timeline when this happened and the total cost incurred up to that time. The engine keeps re-running the simulation until the number of repetitions requested by the user is reached. At that moment, ArchSim computes the MTTF and the cost metric by averaging the recorded time to failure and costs at the end of each repetition.

ArchSim is described in more details, along with case studies illustrating its use, in [10, 11].

4 Replicating information

The process of replicating digital objects at multiple remote sites is key to ensuring that objects survive failures. We have investigated techniques for making replication as robust as possible while reducing the resource requirements on participating sites. In this section we examine *data trading*, where sites negotiate agreements to store copies of data, and *awareness services* to ensure objects are properly propagated after an agreement has been reached.

4.1 Data trading

In community-based replication, multiple archives cooperate to preserve data. Each site contributes storage resources to the system, and in return reserves the right to store copies of its own collection at other sites. A community-based replication system is subject to two constraints:

- Each site is autonomous
- Each site has limited resources

Because each site wants to make its own decisions about how to allocate its sparse resources, it is not feasible to have a central authority dictate which copies will be stored at which sites. Such a central authority is not desirable in any case, since the system is more robust if allocations can be made in a distributed manner.

To overcome these constraints, we have designed a framework for negotiations between sites to allocate resources. The basis of these negotiations is a trade, where one site essentially says to another: "I'll store a copy of your data if you'll store a copy of mine." If both sites agree with this proposition, then they conclude an agreement and allocate space for each other's use. This distributed, barter-based negotiation allows each site to decide what agreements to conclude and thus how to use its own resources. Moreover, we can study policies for deciding when to make trades that allow a site to make the most of its limited resources.

Consider the example shown in Figure 3. Figure 3(a) shows sites A and B, each of which have two gigabytes of space, and site C, which has three gigabytes of space. (A gigabyte is represented in the figure as a box.) Site A owns a collection of data labeled "1," site B owns collection "2," and site C owns collection "3." (A collection is an application unit, e.g., a set of technical reports, or a set of census files.) Each collection

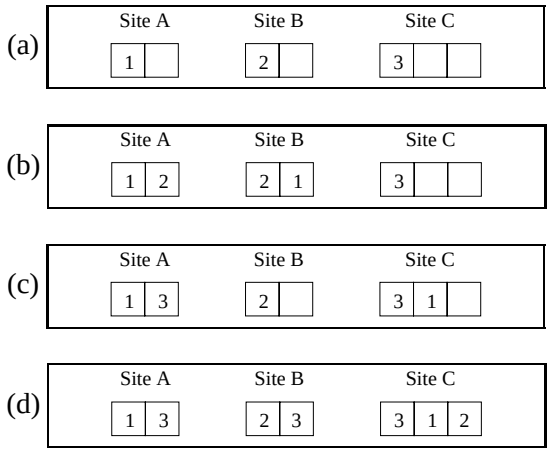

Figure 3: Data trading example.

requires one gigabyte of space. Sites A and B can trade their collections, resulting in the configuration of Figure 3(b). Collections 1 and 2 are now stored more reliably, because if one site goes out of business, goes on strike or burns down, another copy is available. However, now site C cannot trade with either A or B since neither site has free space for collection 3. Thus, collection 3 is not stored reliably.

A different trading order can result in a more desirable scenario. For example, say that from the initial configuration, site A first contacts C and offers a trade. The result is shown in Figure 3(c). Now there is still enough space to make another trade, this time between sites B and C. The resulting situation, in Figure 3(d), has all three collections reliably stored with two copies. A trading scheme should be effective enough so that sites can make local decisions about which sites to trade with, while still allowing other sites to replicate their collections. At the same time, trading must be flexible enough to deal with the appearance of new sites, new collections, and even new free storage added at an existing site.

This example illustrates that a great deal of care must be put into the local decisions that are made by each site. Although it is often impossible to make optimal decisions, especially without knowledge of future events (such as new sites, new storage space added to an existing site, etc.) we can study useful heuristics that tend to improve the overall reliability of the system. We can encapsulate these heuristics in *trading policies* that guide the local decision making at each peer. In this way, the continuous process of offering and accepting trades can be automated. The system, once configured with appropriate trading policies, autonomously replicates information to ensure high reliability.

4.1.1 Trading policies

We have studied several different policies that determine the the behavior of a trading peer. Examples of policies include:

Deed trading. One possibility is for peers to trade collections directly, and this is the approach assumed in the example presented above. This approach has the disadvantage that trades may not be very symmetric; for example, Stanford's collection may be much larger than MIT's, which results in a situation where MIT gives away more storage than it gets in return. A fairer scheme (and, it turns out, more reliable scheme) is one in which blocks of space are traded. For example, Stanford may give MIT 10 GB of space and in return get 10 GB of MIT's space. Each site can then use the space it has acquired as it sees fit. If MIT's collection is smaller than 10 GB, it may be able to use the space it acquired at Stanford to replicate several collections. The bookkeeping mechanism for tracking these trades is called *deeds*: a deed represents the right to use space at another site. Once MIT has acquired a deed for space at Stanford, it can use the deed, save it for the future, split it into smaller pieces, or trade the deed away to another site, as it sees fit.

Advertising policy. A site advertises the amount of storage space it is willing to trade away. In the simplest case, a site advertises all of the space it has free. However, higher reliability over the long term can be achieved by reserving some space for future use. Then, a site only advertises a fraction of its available resources at any one time. This ensures that there is always some space to trade away, which may be needed if the site gets a new collection that it must replicate by proposing new trades.

Remote site selection strategy. When a site wishes to make trades, it must decide which remote sites to offer trades to. One possibility is to choose the remote site that has the lowest probability of failure, estimated based on previous history, reputation, or the quality of components at the site. However, this policy is counterproductive if every peer uses it, because the "high reliability" sites quickly become overloaded. A more effective policy is for peers to choose a small set of "trusted" trading partners, and trade repeatedly with those partners.

Bidding policy. When Stanford asks MIT for a trade, the two sites may exchange equally sized blocks of space. An alternative is for Stanford to offer a trade by saying that it needs a certain amount of space at MIT (say, 10 GB), and asking how much space MIT would want in return. If MIT is eager to trade, because it has many collections to replicate, it may offer a low *bid*, asking for only 5 GB in return. On the other hand, if MIT is reluctant to trade, say because its local space is becoming scarce, then it may offer a high bid, asking

for 15 GB in return. In this way, Stanford can contact multiple sites, get bids from each one, and then accept the most attractive offer. In this scenario, each site must decide, based on its local circumstances, what bid to offer for each trade.

These and other policies have been examined in more detail in [5, 6, 7]. These papers also describe our trading simulator, a system we have built to simulate trading sessions using different policies. Our simulator has allowed us to identify policies which provide the highest reliability in different circumstances.

4.2 Awareness algorithms

One of the most critical components in an archival repository is the awareness mechanism, used to notify clients and replication partners of inserted objects. We have surveyed various awareness schemes (including snapshot, timestamp and log based), and can model them all as variations of a single unified scheme. This makes it possible to understand their relative differences and strengths.

In particular, we have focused on a signature-based awareness scheme that is especially well suited for digital libraries. In Section 2.1, we described how objects handles are constructed from a signature over the objects themselves. When an archive needs to notify another archive or a client about which objects have changed, it must send a list of object handles. However, it is often difficult to determine which objects the client already knew about and which are "new," so an archive often must send a list of handles for all of the objects it has, and let the client sort out whether any have been added. Another possibility is to compute a single signature over the list of signatures, and send just this single signature. The client can also compute a signature over the list of signatures it already knew about. Only if these single signatures differ does the whole list need to be sent. Moreover, this scheme can be performed hierarchically, so that the list can be subdivided multiple times, with signatures computed over each subdivision. Then, only a fraction of the list must be resent when there is a change. For more information about awareness services, see [8].

5 Related work

The digital library community has begun to focus on the problem of designing and implementing long term archives [16, 13, 15]. Some of these efforts share our focus on preserving the bits of digital objects with systems that are more robust than traditional backups [3] and more reliable than traditional replication [22, 20]. Other researchers have looked into higher level issues, such as policy issues surrounding archiving [2],

preserving the ability to emulate computer architectures [21] or protecting the anonymity of publishers [12]. These goals can be met with layers built on top of our architecture. Still others have built active archival repositories, such as the Computing Research Repository [17].

Many aspects of our work are related to research in areas outside of archiving and fault tolerance. For example, there has been much work lately examining distributed searching mechanisms [1, 18] similar to our routing indices. At the same time, distributed negotiations, like our data trading, have been studied in the distributed database community [14] or for other resource management problems [19].

6 Conclusion

Building a reliable digital archive is a challenging but important problem. We have designed an archival repository architecture to meet this challenge. Our design relies upon the cooperation of individual sites to produce a system that is more reliable than the sum of its parts. We have examined the components of an archive, the techniques that can be used to efficiently and effectively replicate information, and built simulators to evaluate the effectivenes of an archive over a very long time span. Our system is a critical infrastructure that provides protection for digital objects in the face of a wide variety of failures.

References

[1] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in power-law networks. *Phys. Rev. E,* 64:46135–46143, 2001.

[2] N. Beagrie. Developing a policy framework for digital preservation. In *Proc. of the Sixth DELOS Workshop on Preservation of Digital Information*, June 1998.

[3] Ann Chervenak, Vivekenand Vellanki, and Zachary Kurmas. Protecting file systems: A survey of backup techniques. In *Proc. Joint NASA and IEEE Mass Storage Conference*, March 1998.

[4] B. Cooper, A. Crespo, and H. Garcia-Molina. Implementing a reliable digital object archive. In *Proc. European Conf. on Digital Libraries (ECDL)*, Sept. 2000. In LNCS (Springer-Verlag) volume 1923.

[5] B. Cooper and H. Garcia-Molina. Creating trading networks of digital archives. In *Proc. 1st Joint ACM/IEEE Conference on Digital Libraries (JCDL)*, June 2001.

[6] B. Cooper and H. Garcia-Molina. Peer-to-peer data trading to preserve information. *ACM Transactions on Information Systems*, 20(2), Apr. 2002.

[7] B. F. Cooper and H. Garcia-Molina. Bidding for storage space in a peer-to-peer data preservation system (extended version). In *Proc. of the International Conference on Distributed Computing Systems*, July 2002.

[8] Arturo Crespo and Hector Garcia-Molina. Awareness services for digital libraries. In *Lecture Notes in Computer Science*, volume 1324, 1997.

[9] Arturo Crespo and Hector Garcia-Molina. Archival storage for digital libraries. In *Proc. ACM Int'l Conf. on Digital Libraries*, 1998. Accessible at http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1998-0082.

[10] Arturo Crespo and Hector Garcia-Molina. Modeling archival repositories for digital libraries. In *Proceedings of the Fourth European Conference on Research and Advanced Technology for Digital Libaries (ECDL)*, 2000.

[11] Arturo Crespo and Hector Garcia-Molina. Cost-driven design for archival repositories. In *Proceedings of the First Joint Conference on Digital Libraries (JCDL)*, June 2001.

[12] R. Dingledine, M.J. Freedman, and D. Molnar. The FreeHaven Project: Distributed anonymous storage service. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[13] J. Kubiatowicz et al. OceanStore: An architecture for global-scale persistent storage. In *Proc. ASPLOS*, Nov. 2000.

[14] M. Stonebraker et al. An economic paradigm for query processing and data migration in Mariposa. In *Proc. Int. Conf. on Parallel and Distributed Information Sys.*, Sep. 1994.

[15] Y. Chen et al. A prototype implementation of archival intermemory. In *Proc. ACM Int'l Conf. on Digital Libraries*, 1999.

[16] J. Garrett and D. Waters. Preserving digital information: Report of the Task Force on Archiving of Digital Information, May 1996. Accessible at http://www.rlg.org/ArchTF/.

[17] Joseph Halpern and Carl Lagoze. The Computing Research Repository: Promoting the rapid dissemination and archiving of computer science research. In *Proc. ACM Int'l Conf. on Digital Libraries*, Aug. 1999.

[18] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *Proc. of the 1st Int'l Workshop on Peer to Peer Systems (IPTPS)*, March 2002.

[19] T. Mullen and M. Wellman. A simple computational market for network information services. In *Proc. Int. Conference on Multi-Agent Systems*, June 1995.

[20] D. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *SIGMOD Record*, 17(3):109–116, September 1988.

[21] J. Rothenberg. Ensuring the longevity of digital documents. *Scientific American*, 272(1):24–29, Jan. 1995.

[22] G.A. Schloss and M. Stonebraker. Highly redundant management of distributed data. In *Proc. of Workshop on the Management of Replicated Data*, pages 91–95. IEEE, IEEE Computing Society, Nov. 1990.