Published as Technical Memo *HPL-SSP-2001-1*, Hewlett-Packard Labs, Storage Systems Program, March 2001.

Towards Global Storage Management and Data Placement

Alistair Veitch, Erik Riedel, Simon Towers and John Wilkes
Hewlett Packard Laboratories
{aveitch,riedel,stowers,wilkes}@hpl.hp.com

Abstract

As users' and companies' dependence on shared, networked information services continues to increase, we will see continued growth in large data centers and service providers. This will happen both as new services arise, and as services and servers are consolidated on one hand (for ease of management, outsourcing, and reduced duplication), and further distributed on the other hand (for fault-tolerance of critical services and to accommodate the global reach of companies and customers). This paper outlines the key research issues associated with the deployment and management of a global storage system to support this infrastructure. We build on our success in automatically managing local storage systems, and discuss how moving to a system of global data placement raises new challenges and areas of research. We believe that one of the key attributes of such a storage system is the ability to flexibly adapt to a variety of application semantics and requirements as they arise (many applications that will drive Internet data centers five years from now are only now being sketched on napkins) and as they change over time.

1 Introduction

We believe that the near future will continue many of the trends that we already see today. First, the size of data centers is increasing, as services are consolidated and the number of services offered over networks (both over intranets, and across the Internet) increases. Additionally, we see continued growth in the market for outsourcing, where external service providers take on the responsibility of managing and provisioning services traditionally provided by companies' own IT departments. We predict that extremely large data centers will be built and connected by high capacity networking links. A portion of these data centers will be owned by large companies for their own use, but many more will be owned by companies that will sell compute, storage and application services. For reliability, fault-tolerance and performance, it will be necessary to replicate and distribute both compute and storage capacity over a number of data centers. To meet changing demands, it will be necessary to move computation and storage location within the network. This will be facilitated by the fast and relatively cheap network connections between data centers, which allow data to move much more efficiently than the expensive compute and storage elements could (much less their attendant power, cooling, and physical infrastructures). We also believe that technology may soon make feasible a utility-like market in computing and storage services. We believe that this will be necessary in order for companies to balance or trade resources to meet variable, and rapidly changing, demands. The day-to-day management of these systems must be automated, with software that is able to design systems to meet an application's expected needs, configure the system without human intervention, provide service continuity in the event of a failure, and adapt the provisioning and design as the application's needs change.

While there are many challenges to building such a system, covering topics in networking, security, operating system design, systems management, and economics, we concentrate here on those inherent to providing the storage system abstractions and management. Section 2 describes our current system for local storage system management. Section 3 examines the new research issues inherent in extending the scope to a global scale, and describes some preliminary results in characterizing the behavior of a variety of inherently local applications in a global setting. Section 4 discusses related work and Section 5 concludes.

2 Local storage management

We have been working on the problem of storage system design and management for some time [Golding95]. Our recent work has demonstrated that it is possible to automatically design and configure a storage system consisting of one or more disk arrays to meet a set of application requirements and then to dynamically reconfigure the system as application needs change, without any human intervention. The steps that such a system goes through are to *gather workload requirements* (e.g. number of I/Os per second, bandwidth, phasing behavior, and capacity), *automatically provision and design* (based on a constraint such as least cost or highest performance), *configure the system* (provide device settings and establish logical to physical mappings between hosts and devices), *monitor the system* (to detect when requirements change, redesign, and efficiently migrate data), and a *runtime system* (to provide consistency control and allow migration to proceed online). These steps and the research items for each, are illustrated in Figure 1.

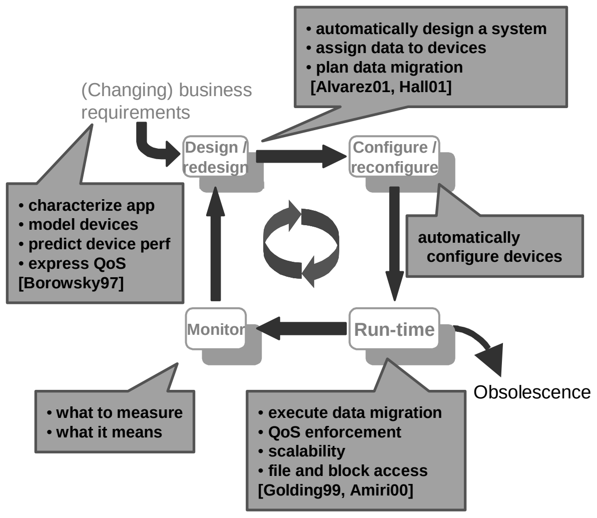

Figure 1: Storage system lifecycle. The components for automatic design and management of storage systems.

3 Research issues

Our work on global data placement will extend this technology for making local placement and optimization choices, to expand the scope of the system into a world of distributed data centers. This will require extending the workload attributes to make them true quality of service (QoS) specifications, including the effects of data streams from outside the data center, modelling the effects of global networks on performance and extending the data placement algorithms to take a much wider range of distribution into consideration, ensuring that the storage is secure in a hostile environment, designing means of replicating data, and choosing appropriate algorithms for keeping it consistent. The following subsections describe each of these issues in turn.

3.1 Quality of service measures

As computer systems and data become more critical to users and businesses, there is an increased need to provide stable and predictable responses in the face of rapidly fluctuating demand. Storage service providers must give QoS guarantees in the form of service level agreements for predictable latency, throughput, availability, integrity and security. To meet these, we need means of specifying and enforcing each measure, which current storage systems do not provide. Existing work on QoS provisioning has mostly concentrated on networking and multimedia storage, with a focus almost entirely on performance, whereas a much richer set of measures are required for service providers.

3.2 Data location

Ensuring that data is in the right global location is one of the key focus areas for our research. Network performance is constantly improving, but there are a number of reasons, not least of which is the speed of light or the inevitable congestion problems, that will result in latency and bandwidth penalties to access data that is located outside of the local data center. This means that data will need to migrate between global data centers in order to have the data that each application needs co-located with the processors that are currently operating on that data. In a highly dynamic and global system, it is likely that the access patterns for any given data set will change over time, perhaps following the daily cycle as users in different timezones become more or less active. To accommodate this, the management system must be able to transparently move data from one data center to another while maintaining the quality of service established for the system. This will require optimization not only among resources with different performance or cost attributes, but also trade-offs of workload changes against migration costs. For many applications, it may be sufficient to slowly evolve location as workload changes - in web service, for example, with tens of thousands of users accessing the same files - while other applications will be sensitive to the metrics such as first-byte-latency that have been much optimized in the case of local file systems with a variety of prefetching schemes [Patterson95, Kimbrel96].

One way of viewing such a system is as a network of “cache” devices – each data center is merely a pool of storage, which at any one time happens to be caching a subset of the global storage. Some of the most important problems in this space are those of policy – deciding which of these devices to use, when to move data from one to another, how many copies to keep and how to decide when to change the current layout. If brokering needs to occur, it will be necessary to develop means to allows systems to negotiate with each other, and build this into the optimization engine. Automating the placement of data to ensure that load is equally balanced across both devices within a data center and between data centers, while meeting QoS guarantees, is the key issue in designing a global computing environment.

3.3 Data replication and consistency

A recent study of ASPs (Application Service Providers) examined the kinds of applications that were being successfully deployed in an outsourcing model and found that applications that are not “designed for the Web” are being poorly received due to performance and scalability concerns. This report argues that the most effective applications for outsourcing are those that have been designed or explicitly adapted to be distributed [Cherry00]. One of the key challenges for global data placement is to provide mechanisms where applications not explicitly designed for distributed access can still be effectively supported.

For many applications, the most efficient solution for the data location problem will be to maintain multiple replicas of a particular piece of data. This, along with the core requirements of high availability in the event of local failures, will make it necessary to store the same data item in

	total	*updates*		*metadata*		*synchronous*		*sync metadata*		
application	*req/s*	*req/s*	*%*	*req/s*	*%*	*req/s*	*%*	*req/s*	*%*	*comment*
cello/tracing	75.3	41.5	55%	0.1	0%	5.5	7%	0.03	0%	
cello/backup	297.5	77.9	26%	73.5	25%	127.2	43%	52.8	18%	
cello/netnews	20.2	15.1	75%	9.1	45%	7.5	37%	3.0	15%	high update rate
cello/compile	0.6	0.5	79%	0.2	29%	0.3	55%	0.1	19%	high sync meta
cello/clearcase	5.1	3.2	63%	0.6	12%	3.7	72%	0.5	9%	
cello/email	2.1	1.6	78%	0.2	8%	0.4	19%	0.1	7%	
cello/httpd	0.1	0.03	31%	0.03	34%	0.03	26%	0.02	16%	
cello/netscape	0.6	0.5	85%	0.2	28%	0.2	39%	0.1	12%	high write rate
cello/others	21.4	12.5	58%	0.3	48%	12.0	56%	6.9	32%	
openmail/server	17.0	12.7	75%	8.7	51%	11.8	69%	5.3	31%	high sync meta
tpc-h/query	1,289.5	3.4	0%	0.5	0%	1,289.5	100%	0.5	0%	read-only
tpc-h/update	9,122.5	3,056.2	34%	1923.8	21%	9,122.5	100%	1923.8	100%	
tpc-h/throughput	2,326.8	388.9	17%	0.5	0%	2,326.8	100%	0.5	0%	read-mostly
tpc-c/oltp	537.9	265.6	49%	–		537.9	100%	–		
web/hp.com	8,002.9	0.1	0%	110.7	1%	8,002.9*	100%		100%	read-only
web/cello	5.7	0.0002	0%	0.003	0%	5.7*	100%		100%	

Table 1. *Application access characteristics.* Disk request breakdowns for a number of server applications. All the cello traces are from a timesharing server used by the 20 members of our research group, a 4 processor HP-UX server with 4 GB of memory and a total of 500 GB of storage. All of the I/Os performed by this machine over a 24 hour period are broken down by the applications that produced them. *tracing* is the I/O done by the tracing system itself, *backup* is running a nightly incremental, *netnews* is an innd server and associated processes, *compile* is all the compiler, make, linker, and editor invocations, *clearcase* is a source code management system that maintains a constantly updated version of the HP-UX source tree on our server, *email* is all I/O due to sendmail and the various email reading programs used in the group, *httpd* is our group web server, *netscape* is all the browsers running on the system, and *others* captures all I/O not included in any of the previous categories. *openmail* is a trace of 1 hour of an OpenMail server with 3,000 users in one of HP's data centers. The *tpc-h* is Q5 and RF1 from the Power Test, and one hour from the Throughput Test of a 300 GB benchmark running of an 8 processor N-class server with over 1 TB of total storage. *tpc-c* is a 116 warehouse benchmark run on a K-class server with about 50 disks. Finally, the web loads are based on access logs of the HP.com web site and our group web server. Note that all rates are averages over an hour or more of trace, so the peak loads on the systems are significantly higher than this. *we have assumed that all requests to web servers are synchronous because there is someone waiting for the result, on the other hand, many users of the web have been trained to accept delays in accessing pages, so perhaps these should be classified as largely asynchronous instead.

multiple places and keep it consistent. The ability to adapt the consistency management within the system to the (varying) requirements of individual applications is a key to making global data placement viable. We believe that there is sufficient variety in application requirements to make a flexible system possible and efficient. A short list of well-understood access semantics includes:

- ***read-only*** – data that is written once and then read by many clients has the simplest consistency requirements and allows aggressive replication to match the location and frequency of reads.
- ***read-mostly*** – if the data is only written infrequently, then the overhead of requiring synchronous updates of a number of global replicas may be acceptable.
- ***single-writer or partitionable*** – if there is only a single writer for a given piece of data, then consistency can be maintained with token schemes that follow a primary replica or schemes such as publish consistency [Burns00] or optimistic methods [Amiri00, Adya95] that assume interference-free updates, but provide schemes for rolling back or re-applying changes in the rare case that conflicts do occur. If some amount of inconsistency can be tolerated, than schemes for asynchronous updates and mirroring also apply.
- ***multiple-writers*** – in applications with high degrees of sharing, maintaining consistency will be more expensive, and optimistic mechanisms may begin to degrade in favor of more pessimistic schemes such as locking. One of the challenges of our system is to minimize the amount of multiple-writer sharing that occurs - to eliminate false sharing wherever possible, and be able to detect the effective partitioning of data items, even when this is not explicitly present in the application.

Ideally this would be done transparently, without any changes to existing application code, and we feel that a large amount of the necessary information and flexibility is available even with today's storage interfaces that were designed assuming resources are local. There are three levels at which consistency among replicas can be maintained:

- ***within the block storage interface*** - this capability has been explored for the wide-area case as synchronous mirroring, where data is mirrored between a primary and a secondary site for disaster-tolerance. Fully synchronous mirroring can be quite expensive, so many installations employ some form of asynchronous mirroring where updates propagate more slowly. The asynchronous technique also allows some degree of coalescing of requests and captures some of the rapid overwrites that are common in many workloads. In order to begin to evaluate the cost of such schemes across different applications, Table 1 shows the characteristics of a number of block-level I/O traces. All of these traces were taken on a local file system that assumed disks were local to the processing node. The challenge for global data placement is to determine how these applications can be supported in the global case without being explicitly designed for it. Much of the data is not very promising - a high fraction of requests are updates, as most reads hit in the file system cache [Baker91], most applications have a high proportion of metadata accesses [Ruemmler93], and a high fraction of requests are synchronous, where the storage system does not have much leverage in scheduling or delaying. The most promising data is the final column which shows that the amount of synchronous metadata (which would incur the highest latency penalties in a distributed system), while still high in several applications, is quite low in a number of them, making a adaptive consistency mechanisms attractive. To more closely examine how much actual sharing takes place within these applications Table 2 shows a subset of the same applications and

	block sharing			*metadata sharing*		
application	*(000s)*	*pids*	*interlv*	*(000s)*	*pids*	*interlv*
cello/netnews	1,722	43%	4%	619	43%	11%
cello/compile	43	12%	0%	7	21%	0.1%
cello/clearcase	1,170	65%	0%	23	58%	12%
cello/email	463	30%	0%	17	32%	3%
cello/httpd	68	37%	8%	7	41%	18%
cello/netscape	32	9%	3%	5	14%	2%
cello/others	–	–	–	743	90%	1%
openmail/server	117	25%	1%	29	26%	0.1%

Table 2. *Application sharing.* Sharing of data and metadata blocks for a number of the traced applications. The number of unique blocks accessed, the fraction of blocks accessed by more than one process id over the period of the trace, and the fraction of blocks accessed by interleaved process ids (where two accesses by the same process are interrupted by another process). The columns to the right show the same data considering only metadata blocks. The durations of the trace are one day for all the cello applications except *httpd* which is ten days, and one hour for *openmail.*

quantifies the degree of sharing. Adaptive consistency has already been studied in the area of distributed shared memory systems [Mosberger93, Amza99] where pages are kept consistent across remote nodes. We hope that some of the mechanisms from this work will be applicable in the context of storage and the more storage- intensive applications in the data center.

- ***within the file system*** – there is a large body of work on consistency in distributed file systems [Kistler92, Anderson96, Thekkath97, Peterson97, Bolowsky00]. These systems are applicable for network file system workloads, where users are explicitly aware that their data is stored remotely. The large majority of data today is still stored in local file systems that are slowly becoming distributed to some degree by the growth of storage area network technology [Phillips98]. This means that applications designed for local file systems will now be forced into a distributed context. Depending on how the distribution of applications is accomplished, this may mean that many of the benefits currently provided by large file system buffer caches will be challenged. An outline of this problem is shown in Table 3 which compares the number of requests made to the file system and the number of requests that eventually filter into the disk sub-system. Local caches provide a big benefit, and maintaining consistency across distributed caches will require additional system overhead.

	requests/s		*updates/s*		*pid sharing*	
application	*app*	*disk*	*app*	*disk*	*app*	*disk*
cello/netnews	59.6	20.2	22.6	15.1	25%	43%
cello/compile	27.4	0.6	4.0	0.5	42%	12%
cello/clearcase	89.0	5.1	31.7	3.2	6%	65%
cello/email	17.0	2.1	4.4	1.6	17%	30%
cello/httpd	0.1	0.1	0.0	0.0	38%	37%
cello/netscape	41.9	0.6	16.5	0.5	1%	9%

Table 3. *Local caching.* The total request rate, update rate, and degree of sharing into the file system cache (file system calls), and how much of it filters down to the disks (as block requests). The request rate to the disk is always reduced from the rate into the cache, by factors of 10 and more in many cases. In addition, the sharing behavior changes drastically in several of the applications between application and storage system. Sharing is defined as in the previous table: the fraction of the blocks or files accessed by more than one unique process id over the course of the trace.

- ***by the application itself*** – applications that have been designed to operate in a distributed environment can manage their own consistency through the use of specialized protocols. If this were generally true, it be the most efficient for global data placement as well, since applications can exactly express their needs, rather than relying on limited interfaces, but it will often considerably increase the complexity of the application. Recent work on such an approach proposes middleware which can manage consistency based on application-specific criteria and provide a spectrum of consistency guarantees [Yu00]. The challenge for global data placement is to provide such a range of semantics to a range of server applications without major (or any) changes in the code.

Global data placement will have to offer a range of consistency levels and protocols across all three of these layers in order to most efficiently support both existing and emerging applications. Consistency levels and mechanisms might be explicitly specified as part of an application's QoS attributes, but - as with many of the performance attributes we use today - in many cases they will have to be inferred by observing the applications' access patterns.

3.4 Security

Data must be secure at all times, especially in a system where facilities are shared amongst many different organiations. Storage providers need to make guarantees that only those who are authorized to access the data can do so. This requires strong authentication, authorization and encryption mechanisms, none of which are necessary in the context of local storage systems. A very basic version of this has gotten some recent attention in the zoning functionality of fibre channel switches [Brocade00], but more comprehensive mechanisms are needed in the context of the general-purpose networks that future storage devices using iSCSI [Satran01] will be subject to. Recent work in survivable systems [Wylie00] has begun to focus on the trade-offs for a storage security infrastructure. Table 4 shows some of the key drivers that determine the cost of a security system for a particular application, which are the number of objects being protected, and the number of different principals accessing the set of shared objects. The question for both is which level of abstraction to provide for different classes of users and data.

	objects (000s)		*principals*	
application	*files*	*blocks*	*pids*	*users*
cello/netnews	296	1,722	434	2
cello/compile	3	43	968	11
cello/clearcase	16	1,170	3,542	3
cello/email	6	463	1,385	18
cello/httpd	0.4	68	9	1
openmail/server	–	117	1,166	–
tpch/throughput	–	–	123	6
web/hp.com*	70	2,974	–	129,351
web/cello*	0.8	56	9	272

Table 4. *Security metrics.* The key parameters are the number of objects being protected, and the number of principals accessessing these objects. The table shows the total count of files and blocks, and the number of unique process ids and usernames over the course of the trace period. *users in this case is defined as unique host names making accesses.

3.5 Overall system management and control

The management system described in Section 2 is able to determine appropriate data placement in the local case. We anticipate a hierarchy of such systems as we move to a more global system, with some optimization best done within the data center, and a global view to control movement among distributed replicas. The combined systems would operate at a large range of time scales and granularities, but both will depend on the ability to accurately and efficiently model the behavior of all the components and links across the system in order to optimize appropriately.

4 Related work

The Coda work at CMU [Kistler92] and Bayou at Xerox [Peterson97] explored semantics for applications operating in the presence of disconnected or loosely connected clients accessing a consistent central store, and Odyssey [Noble97] explored the adaptation of individual user applications to changes in the level of connectivity.

OceanStore [Kubiatowicz00] proposes an architecture for creating a persistent global store that relies on large numbers of encrypted replicas, distributed around the world, to provide security and availability. The OceanStore update model is relatively expensive, requiring a large number of network messages, and would be strained by applications requiring fine-granularity update semantics. The target system for OceanStore is only loosely coupled and covers a different set of applications - managing the data of a large number of individual users persistently - than the data-intensive processing of large applications and databases.

The founders of Akamai have built a successful company by distributing web content using servers positioned at strategic points in the Internet and intelligent distribution algorithms [Karger97, Harchol99]. These systems work well due to the relatively weak consistency guarantees and low update rates of web content. A comprehensive system must encompass all data types, and provide a much richer range of functionality for both distribution and consistency.

5 Conclusions

We have presented our vision of the future as a set of large, distributed data centers that provide services to a global set of clients. These data centers enhance the maintainability of the systems and provide the cost effectiveness of service consolidation. However, centralization will always have the disadvantage of "putting all the eggs in one basket". For this reason, and in order to efficiently support large and fluctuating demands, we believe that data centers must be able to automatically manage replicated storage and computation resources among themselves. Such a system would also support trading of resources among data centers owned by different organizations. In this paper, we have identified some of the key research issues in extending current systems which do local storage management to the global case. For several of these issues, we have provided initial measurements of applications that illustrate the potential benefits of a using a flexible and adaptive approach supported by automatic system design and optimization.

Acknowledgements

We have benefitted from discussions with various members of the HP Labs Computer Systems and Technology Laboratory, particularly John Janakiraman, Magnus Karlsson, Lance Russell and Hans Stork as well as all the members of the Storage Systems Program. We also thank Magnus for providing the excellent log analysis tool we used for the web workloads.

References

[Adya95] A. Adya, R. Gruber, B. Liskov and U. Maheshwari. Efficient optimistic concurrency control using loosely synchronized clocks. *SIGMOD*, May 1995.

[Alvarez01] G. Alvarez, E. Borowsky, S. Go, T. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch and J. Wilkes. Minerva: an automated resource provisioning tool for large-scale storage systems. *Submitted for publication. A tech report version will be available by May.*

[Amiri00] K. Amiri, G. Gibson and R. Golding. Highly concurrent shared storage. *Intl. Conference on Distributed Computing Systems*, April 2000.

[Amza99] C. Amza, A. Cox, S. Dwarkadas, L. Jin, K. Rajamani and W. Zwaenepoel. Adaptive Protocols for Software Distributed Shared Memory. *Proc. of the IEEE* 87(3), March 1999.

[Anderson96] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli and R. Wang. Serverless Network File Systems. *ACM Trans. on Computer Systems* 14(1), February, 1996.

[Baker91] M. Baker, et al. Measurements of a distributed file system, *SOSP*, October 1991.

[Borowsky97] E. Borowsky, R. Golding, A. Merchant, L. Schreier, E. Shriver, M. Spasojevic and J. Wilkes. Using attribute-managed storage to achieve QoS. *Proc. 5th Intl. Workshop on Quality of Service*, June 1997.

[Bolowsky00] W. Bolosky, J. Douceur, D. Ely, M. Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. *SIGMETICS*, June 2000.

[Brocade00] Brocade Communications. *SilkWorm 6400 product specification*, October 2000.

[Burns00] R. Burns, R. Rees and D. Long. Consistency and locking for distributing updates to web servers using a file system. *Workshop on Performance and Architecture of Web Servers*, June 2000.

[Cherry00] Cherry Tree & Co. *2nd generation ASPs – Spotlight Report*, September 2000.

[Golding95] R. Golding, E. Shriver, T. Sullivan and J. Wilkes. Attribute-managed storage. *Workshop on Modelling and Specification of I/O*, October 1995.

[Golding99] R. Golding and E. Borowsky. Fault-tolerant replication management in large-scale distributed storage systems. *Proc. Symposium on Reliable Distributed Systems*, October 1999.

[Hall01] J. Hall, J. Hartline, A. Karlin, J. Saia and J. Wilkes. On Algorithms for Efficient Data Migration. *12th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 2001.

[Harchol99] M. Harchol-Balter, T. Leighton and D. Lewin. Resource Discovery in Distributed Networks, *18th Symposium on Principles of Distributed Computing*, May 1999.

[Karger97] D. Karger, E. Lehman, F. Leighton, M. Levin, D. Lewin and R. Panigraphy. Consistent hashing and random trees: Distributed cachine protocols for relieving hot spots on the World Wide Web, *29th ACM Symposium on Theory of Computing*, May 1997.

[Kimbrel96] T. Kimbrel et. al. A Trace-driven comparison of algorithms for parallel prefetching and caching. *OSDI*, October 1996.

[Kistler92] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Code File System. *ACM Trans. on Computer Systems* 10(1), 1992.

[Kubiatowicz00] J. Kubiatowicz, et al. OceanStore: An Architecture for Global-Scale Persistent Storage. *ASPLOS*, December 2000.

[Mosberger93] D. Mosberger. Memory Consistency Models. *Technical Report 93/11*, University of Arizona, 1993.

[Noble97] D. Noble, M. Satyanarayanan, D. Narayanan, E. Tilton, J. Flinn and K. Walker, Agile Application-Aware Adaptation for Mobility. *SOSP*, October 1997.

[Patterson95] R. Patterson, G. Gibson, E. Ginting, D. Stodolsky and J. Zelenka. Informed Prefetching and Caching. *SOSP*, December 1995.

[Peterson97] K. Petersen, M. Spreitzer, D. Terry, M. Theimer and A. Demers. Flexible update propagation for weakly consistent replication. *SOSP*, October 1997.

[Phillips98] B. Phillips. Have Storage Area Networks Come of Age? *IEEE Computer* 31(7), 1998.

[Satran01] J. Satran, et al. iSCSI draft standard. *www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-03.txt*

[Thekkath97] C. Thekkath, T. Mann and E. Lee. Frangipani: A Scalable Distributed File System. *SOSP*, October 1997.

[Wylie00] J. Wylie, M. Bigrigg, J. Strunk, G. Ganger, H. Kiliccote and P. Khosla. Survivable information storage systems. *IEEE Computer*, August 2000.

[Yu00] H. Yu and A. Vahdat. Design and evaluation of a continuous consistency model for replicated services. *OSDI*, October 2000.