Appears in *Proceedings of the 20th IEEE Int'l Performance,Computing and Communications Conference (IPCCC 2001),* April 2001.

When Local Becomes Global: An Application Study of Data Consistency in a Networked World

Erik Riedel, Susan Spence, Alistair Veitch
Hewlett-Packard Laboratories
{riedel,suspence,aveitch}@hpl.hp.com

Abstract

As users and companies depend increasingly on shared, networked information services, and as companies and customers become more international, computer systems will need to keep pace with a global scale. We believe that we will continue to see growth in large data centers and service providers as new information services arise and existing services 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 looks at storage systems in such a networked, global environment, studies the characteristics of applications' demands on the storage system and quantifies how these demands might be supported in such an environment. Our study focusses on a collection of traces from several large commercial workloads taken on single, centralized servers, and explores the inherent consistency and performance requirements if these same applications were moved into a networked, distributed setting. Rather than focussing on applications or services "designed for the Web", we look at applications designed to operate on local storage and examine how well or poorly they would fare if they were migrated to a distributed setting. Our results indicate that in many cases, the "inherent" level of consistency required and the "true" amount of sharing is sufficiently low that distribution may be feasible with reasonable performance, and without wholesale application changes.

1 Introduction

There is increasing demand for use of applications and access to data across globally-distributed sites. For many companies, this presents a challenge of how to use existing distributed applications in a widely-distributed context, when they were only designed for use in local area networks. It is an even bigger challenge to make existing, non-distributed applications and the associated, often large, datasets available to offices across the globe. Whatever the location of the applications and their data, there are a range of data access semantics that must be supported correctly: ranging from read-mostly data analysis, to loosely-coupled file-sharing to tightly-coupled database systems. *Global data placement* services provide the storage infrastructure to support such globally-distributed applications, including automatic management, replication, caching, and guaranteed performance levels. In this paper, we examine and assess the challenges of providing such services transparently to existing applications.

We analyze several existing applications to demonstrate the feasibility of supporting data placement in a globally distributed context. A number of application scenarios are presented, with a description of how the application might be distributed and consideration of what replication and consistency implications this distribution requires. The replication policies include full replication of an application's data and partial replication of only the data that is shared across multiple sites. The consistency requirements are influenced by the replication used and by the requirements of the application itself, and include cases where delayed propagation of updates is acceptable and others where strong consistency must be maintained.

We present an analysis of a number of traces taken from applications running on a centralized server and attempt to match these traces to the workload characteristics of the proposed distributed scenarios. The analysis identifies the working data sets used by different sites, distinguishes between the total amount of data accessed and the unique footprint of that data, and pinpoints potential data access conflicts. We use this data to reason about what the costs of replication and consistency maintenance might be and whether a global data placement system could cope with these costs and still support the necessary replication transparency for applications.

2 Application scenarios

Consider the increasingly common scenario of a multinational company. With offices all around the world, the company needs to access its data using a variety of applications efficiently and reliably across those globally-distributed sites. The new-world infrastructure is composed of a number of globally-distributed data centers. Each data center provides data storage and processors to run applications. The centers might be connected with dedicated links, or may use connections over the commodity Internet. We examine the implications of supporting global use of a range of applications which were originally intended for use only in a localized setting.

2.1 How applications might become global

The manner in which data may be feasibly distributed is largely dictated by the access semantics of applications. We describe three example scenarios to illustrate a range of access semantics, from read-only to multiple writers.

In the first application scenario, the company collates its latest sales figures and other statistics centrally each night and makes the latest statistics available to its sales offices around the world. Once a report has been issued, the distributed offices make read-only accesses to the statistics, for data analysis purposes.

The second application scenario addresses email. The company provides email access for each of its employees. Normally, the employees access their email at their home site. However, when they travel, they require remote access to their email, preferably with no performance degradation even when they are half-way across the world. This scenario involves only one or a small number of writers accessing the same data: the mail server writes messages to stable storage, awaiting delivery to the user; while the user can read and delete those messages, as well as writing new ones.

The third application scenario is concerned with the shared inventory and product data used by the company's sales centers and warehouses. The company traditionally has a single, shared store of data about customers and product inventory. However, in order to support a 24-hour, global distribution system, it is desirable for this data to be distributed and replicated across all the global sites for increased reliability and availability. This data may be read and written by multiple sites concurrently.

2.2 Use of replication and consistency

Having determined the access semantics of the applications, we can determine an appropriate placement of the data for the applications across globally-distributed sites.

For the data analysis scenario, each site should host its own *full replica* of the report on company statistics, for fast, local, read-only access. *No consistency* has to be maintained on the distributed replicas of the report, since the users only need read access to extract data for analysis.

For the email scenario, it is desirable to support *partial replication* of data to accommodate those users who travel. The stored email for one user could be replicated so that it is available at both the user's home site and at an office in another country that they visit regularly, while non-travelling users only need to have their email maintained at their home site. To generalize, the replication model is based on the location of those that access the data. If the data is only accessed by users at one site, it should be located at that site, resulting in no delay in reads and writes to that data. Alternatively, if the data has multiple writers at different sites, it should be replicated across the locations of those writers. Given that both the user and the mail system may write to a user's stored email, it is necessary to maintain consistency across the replicas, but *delayed consistency* is acceptable.

For the inventory scenario, it is desirable to support *full replication* of the complete data set at each site, supporting local warehouses, as any site may access any portion of the data. Since multiple sites may write to the store concurrently, it is then necessary to maintain *strong consistency* between these replicas.

We believe that a successful global data placement system will need to determine which of these mechanisms to use dynamically and automatically as applications run.

2.3 Cost of replication and consistency

To determine the feasibility of distributing applications across globally-distributed sites, we require some estimate of the costs of replication and consistency.

Many approaches exist for maintaining various degrees of consistency across replicated, distributed data. Determining the mechanisms most appropriate for maintaining consistency across globally-distributed data centers is the subject of future work. However, we can make some provisional, basic estimates here about how much it would cost, in terms of number of messages sent, bandwidth used and latency incurred. The costs are affected by the number of sites involved and the consistency requirements of the application. Making such estimates allows us to determine whether the access patterns observed in our traces leave us with sufficient opportunity to meet the costs of distributing the data.

We consider the following degrees of consistency for our application scenarios:

- *no consistency*, where once replicas have been created, updates are propagated only periodically (e.g. nightly);
- *delayed consistency*, where notification that the data has changed must be received before the next access occurs, although the actual update may not be complete;
- *strong consistency*, where the most recent version of the data must be available before the next access occurs.

The begin to quantify the latency costs, Table 1 shows the basic costs of communication between a machine at Hewlett-Packard Laboratories in Palo Alto and a number of machines around the globe. Using the *ping* command to send ICMP datagrams to the destination sites gives the round-trip time for the most basic IP message between those sites. The table displays the average cost in milliseconds of a one-way message deduced from these measurements, for repeated sends of 64 byte and 4 kilobyte packets, at different times of day. The 4 KB packet size matches the average request size seen in our traces.

While the size of the message has a minor effect on the time taken to send the message, the latency between the sites is

destination	6 pm PST 64B	6 pm PST 4KB	12 noon PST 64B	12 noon PST 4KB
local server	< 1	2	<1	2
Palo Alto, California	26	36	29	34
Glasgow, Scotland	85	94	89	95
Capetown, South Africa	346	1012	545	979
Canberra, Australia	140	148	139	148

Table 1. Cost of sending short and long one-way messages between machine at the HP Palo Alto site and several globally-distributed sites. All times in milliseconds.

obviously the most important factor. For instance, at 6 pm PST it takes 28 times as long to send a 4 KB packet from Palo Alto to Capetown as it does to send it between machines at different sites within Palo Alto.

The significance of the cost of a single message between sites becomes apparent when we consider the number of messages that may need to be sent between globally-distributed data centers in order to keep replicated data consistent.

In scenario one, there is a clear benefit in replicating the report to all the sites that require it for analysis, rather than requiring the sites to access one centralized copy. The cost of repeated read requests over the network to the centralized copy is high and this cost will be significantly affected by the latency between the sites. Directing the read requests to a local replica avoids this cost and, in this scenario, requires no extra overheads for maintaining consistency, since the replicas are not being changed concurrently.

In scenario two, the partial replicas are expected to be kept consistent with the original, with some delay in the synchronization of the replicas being acceptable. Taking a simple approach, the minimum number of messages possible for propagation of one update to R replicas is *2R,* i.e. *R* messages resulting in *R* acks.

In scenario three, a number of full replicas are maintained, with strong consistency between the replicas. Using a worst-case two-phase commit protocol in this context would result in *3R* messages to propagate a single update to R replicas: one commit request, one response and one subsequent commit command per replica [Coulouris94].

2.4 Consistency at the storage level

There are a wide range of options for ensuring consistency of data directly in applications - from using a database management system with strong consistency guarantees to various forms of middleware such as object-oriented databases, distributed object systems, or simply by implementing communication between distributed sites directly. Using such mechanisms allows each individual application to choose the semantics and performance requirements that are most appropriate for the behavior its users expect. The difficulty is that each individual application must be modified to operate using the common middleware platform, which can require large amounts of coding and often wholesale re-design of the software - an investment that is often difficult to make without a guarantee that it will pay off.

In most systems today, there is already a common interface to storage - applications are written against a file `open`, `close`, `read`, `write` interface at the file system level, which translates into a set of lower-level block requests. This commonality of interface makes it attractive to provide global data placement as a service at this level. This is already being done in two different fashions today. At the block level, remote mirroring across disk arrays connected by a wide-area link [EMC00] provides replication of all write operations by applications at a primary node to a second backup node, but only one node is active at any given time. These systems can operate synchronously (ensuring complete consistency of the backup), asynchronously (delaying some operations), and variants of batch windows (sending periodic updates), however these choices must be made all-or-nothing for the system, and do not differentiate between different classes of access.

The second option for higher-level access is use of a distributed file system - this layer has more information about access patterns and types of access, but still provides a limited set of interfaces for specifying different consistency semantics [Kistler92, Peterson97, Thekkath97, Ji00].

We believe that a successful system for global data placement should operate at the lowest-level of these interfaces in order to provide the maximum compatibility with existing applications, while at the same time taking into account as much “higher-level” knowledge as possible. The system should not expose additional semantics, but should observe the behavior of applications and adapt the consistency and distribution schemes that are most appropriate for each subset of data and application behavior. The data in Table 2 shows the characteristics of several applications to illustrate the range of observed access patterns.

If the storage system is aware of whether requests are synchronous or asynchronous, and whether requests are for data or metadata, then it can better optimize the choice of distributed semantics. The next section will explore this in more detail by considering concrete examples of the scenarios introduced in the previous section.

3 Trace analysis

This section considers a number of traces taken on single, centralized servers for a set of commercial workloads as introduced in Table 2. We use a different trace to illustrate each of the scenarios introduced in the previous section. The intent is to consider applications that currently operate

application	total	metadata		synchronous metadata	
	req/s	req/s	%	req/s	%
cello/netnews	20.2	9.1	45%	3.0	15%
cello/users	17.7	9.0	51%	6.2	35%
openmail/server	17.0	8.7	51%	5.3	31%
tpc-h/update	9,120.0	1,920.0	21%	1,920.0	100%
tpc-h/throughput	2,330.0	0.5	0%	0.5	0%
tpc-c/oltp	538.0	–		-	

Table 2. *Application access characteristics.* Disk request breakdowns for a number of traced applications. The cello trace is 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 500 GB of storage. *cello/users* represents all of the user activity on the machine, with system and server processes excluded. *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 RF1 from the Power Test, and one hour from the Throughput Test of a 300 GB benchmark run on an 8 processor HP N-class server. *tpc-c* is a 116 warehouse benchmark run on an HP K-class server with about 50 disks.

on single systems with local storage, and evaluate the cost and complexity of distributing them across global sites.

3.1 Rate of reads and updates

The first application property we must consider when moving from a local system to a global system is the amount of data that must be transferred around the globe in a particular scenario. This is shown in Table 3, giving the total volume of data transferred, and the total amount of unique data (the *footprint*), and the total amount of data modified in the trace period (the *write footprint*).

3.1.1 TPC-H benchmark

The TPC-H benchmark models a decision support system, with a number of query streams performing data analysis, and a concurrent stream of updates adding and deleting records. Decision support allows considerable flexibility in consistency of the underlying database. For most queries, cursor stability [Gray92] is sufficient and these systems are often used in a batch update mode, where updates are regularly propagated (e.g. nightly or weekly).

3.1.2 TPC-C benchmark

The TPC-C benchmark models an OLTP system. A number of transaction types run against a shared database to enter orders, deliver them, and monitor inventory. Separate streams of read/write transactions are run in parallel. We use the traces of three TPC-C streams to model three distributed sites running a range of transactions that read from and write to a replicated warehouse database. Such a database might be used to manage inventory in a global company with many warehouses, or might form a smaller part of other applications such as call logs or customer data used by a company's global service centers or call centers.

3.1.3 Email trace

The openmail trace is taken from a centralized email server used by a large number of users. The data is stored as a logically coherent database, but is accessed largely independently - there will be some amount of overlapping data in maintaining the index of messages and pointers to messages sent to different users, but the majority of the data should be read and written by only single users.

3.1.4 File system trace

The cello trace was taken over a number of days on a server supporting a small group of researchers. A range of applications are run on the server, including compilation, email, web browsing and source code control. The data shown here only considers processes run by individual users, and not system processes such as backup and a netnews server that also runs on the same machine. The intent is to model a scenario with any type of data shared in different ways, perhaps across global design centers or call centers - any system with a shared set of data used at multiple sites, with low day-to-day overlap of data use.

3.2 Determining what to replicate where

The data in Table 4 considers two scenarios for data shared among a number of users that work largely independently, but occasionally share data. We use the *openmail* and *cello* traces and consider what happens if users are distributed across three global sites, rather than being located at one

application	length	total requests	request rate	volume of data read/written		footprint		write footprint	
		(000s)	req/s	MB	MB/s	MB	MB/s	MB	MB/s
cello/users	1 hr	381	105	2,100	0.6	907	0.25	498	0.14
cello/users	24 hr	1,370	380	8,410	0.1	2,800	0.03	1,370	0.02
openmail/server	1 hr	61	17	211	0.1	75	0.02	49	0.01
tpc-h/update	5 min	2,740	9,120	10,700	36.0	329	1.1	286	1.0
tpc-h/throughput	1 hour	8,400	2,330	868,000	241.0	-	-	-	-
tpc-c/oltp	2 hr	4,220	540	9,860	1.4	1,700	0.24	1,400	0.19

Table 3. *Amount of data moved and footprints.* Summary for data operations in several traces. The total number of requests and request rates - which would have to be network messages in a distributed store. The total volume of data transferred, and the total volumes of unique data used, as well as the amount of unique data written.

application	length	volume of data read/written (MB)			site footprint (MB)				write footprint (MB)			overlap footprint (MB)			
		site A	site B	site C	all	A	B	C	A	B	C	AB	AC	BC	ABC
cello/users	24 hr	8,400	775	614	3,160	2,800	328	178	1,370	263	119	134	48	25	23
openmail/server	1 hr	211	124	23	117	75	54	13	49	47	5	17	1.3	1.2	1.1

Table 4. *Site footprints and overlap.* Overall statistics for footprints in the distributed variant for two of the traces. The data shows the total amount of data read or written at each site, the total combined footprint (unique blocks), and the footprint at each site, the write footprint (unique blocks written) at each site, and the overlap footprints across sites.

central site. Users are distributed by randomly assigning them to sites, with 60% at site A, 30% at site B and 10% at site C - e.g. modelling a company with three design centers around the world. The tables shows the footprints at each site, and the overlap across sites (as illustrated in Figure 1).

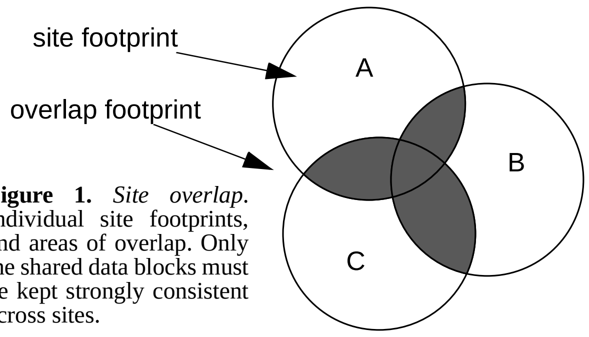

Figure 1. *Site overlap.* Individual site footprints, and areas of overlap. Only the shared data blocks must be kept strongly consistent across sites.

This data would be used to determine the distribution of data in a system using partial replication, where the data local to a site would be kept only at that site, and the data that overlapped between sites might be kept fully replicated. It is promising that the footprints of unique data are significantly smaller than the total amount of data moved (meaning that local access is beneficial) and that the size of the overlapping footprints is small (indicating low degrees of sharing over short time scales).

3.3 Cost of maintaining strong consistency

In order to evaluate the cost of maintaining consistency across full replicas, we consider the cost of propagating updates to several global sites. There are two aspects to this cost, depending on whether a strongly consistent or a more weakly consistent model is used. In the strongly consistent model, the key cost is the additional latency of a write request that must be committed to several global replicas. In the weakly consistent model, or a model using optimistic concurrency [Amiri00], the key metric is the potential for conflicts given the propagation time of requests to the remote sites. Figure 2 considers both of these factors and shows the number of accesses in our traces that might conflict given the across-the-world latencies introduced in Table 1. Again, the results are quite promising for all three traces. Only a bit more than one percent of the write accesses in the traces may cause a conflict, even with the very long latency between Palo Alto and Capetown. For the cello trace, the number of potential conflicts is nearly zero.

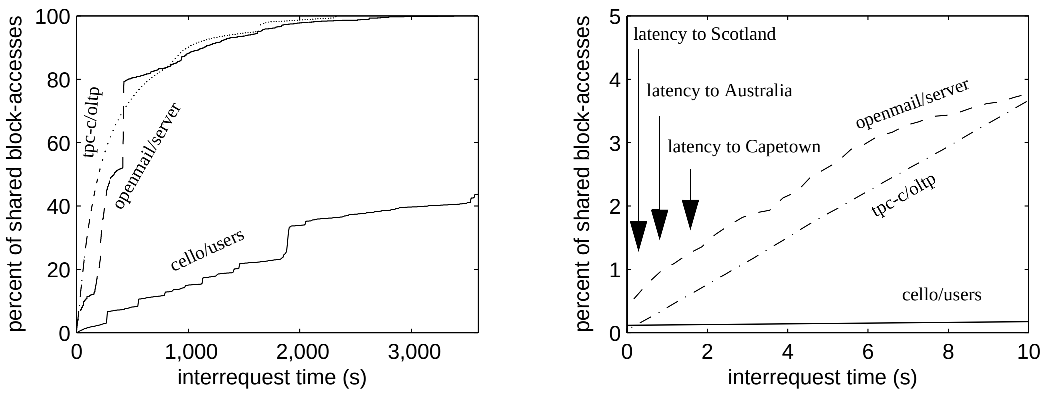

Figure 2. *Potential consistency conflicts.* The time between a write request to a block and the next read or write request to the same block from a different site, for all blocks accessed at more than one site. The chart shows the cumulative percentage of interrequest times for all of the write/access pairs in the *tpc-c/oltp*, *openmail/server* and *cello/users* traces. The left chart shows up to a latency of 1 hour, and the right chart shows only the lower left corner of the first chart, up to 10 second latency. The arrows indicate the time to send the 3 messages of a 2-phase commit to each of the locations specified. This allows us to quantify the number of accesses that might be impacted by a strong consistency protocol requiring such messaging. These charts are based on the number of individual pairs of accesses, so many of the accesses may be to the same blocks.

4 Related Work

For consistency at the storage level, the bulk of previous work is in the context of distributed file systems [Kistler92, Peterson97, Thekkath97, Bolosky00, Ji00]. These use a variety of methods for achieving consistency, many of which are applicable to the applications presented here. However, they each allow only a single model of consistency, while we believe that the system should offer multiple models, and adjust to an application's requirements with minimal changes to access semantics, allowing a much wider range of applications to make use of these facilities.

OceanStore [Kubiatowicz00] proposes an architecture for a persistent global storage system that relies on large numbers of encrypted replicas, distributed around the world, to provide security and availability. The focus is on a very decentralized system targeted at large numbers of individual users, rather than centralized stores and processing.

Akamai has built a successful company by distributing web content using servers at strategic points in the Internet and intelligent distribution algorithms [Karger97]. These systems work well due to the relatively weak semantics and low update rates of web content, leaving room for systems that encompass a wider range of data types and semantics.

At the block level of storage access, there may be lessons in work on distributed shared memory systems [Mosberger93, Amza99], as well as in mechanisms developed explicitly for storage systems such as various forms of optimistic concurrency [Adya95, Amiri00] or specialized semantics [Burns00, Yu00] applicable to a subset of workloads.

Various forms of concurrency control have been studied for many years in the context of database systems [Gray92], object-oriented databases [Butterworth91, Lamb91], and in distributed object systems [Birman93, Liskov96, Little96].

5 Conclusions

We have considered a number of scenarios for distributing a single data store across a number of global sites. Using block-level traces of I/O activity for a number of applications from single server, local systems, we have proposed scenarios for distributing the data sets used, and evaluated the costs of maintaining consistency among these replicas. Our initial results are promising in that the amount of data that must be kept strongly consistent in the applications we studied is quite low, and the cost of maintaining even strong consistency across a number of replicas may be bearable. Our future work will be to identify the set of consistency schemes that can be applied in different scenarios, to determine automatic ways to identify which portions of a data set must be strongly replicated and which can be weakly replicated, and to quantify (and minimize) the impact on performance as seen by end users.

References

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

[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.

[Birman93] K. Birman. The process group approach to reliable distributed computing. *CACM* 36 (12), 1993.

[Bolosky00] 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.

[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.

[Butterworth91] P. Butterworth, A. Otis and J. Stein. The Gemstone Object Database Management System. *CACM* 34 (10), 1991.

[Coulouris94] G. Coulouris, J. Dollimore and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, 1994.

[EMC00] EMC Corporation. Symmetrix Remote Data Facility (SRDF). *Product Description Guide*, June 2000.

[Golding99] R. Golding and E. Borowsky. Fault-tolerant replication management in large-scale distributed storage systems. *SRDS*, October 1999.

[Gray92] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*, September 1992.

[Ji00] M. Ji, E. Felten, R.Wang and J. Singh. Archipelago: an island-based file system for highly available and scalable Internet services. *USENIX Windows Symposium*, August 2000.

[Karger97] D. Karger, E. Lehman, F. Leighton, M. Levin, D. Lewin and R. Panigraphy. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web, *STOC*, May 1997.

[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.

[Lamb91] C. Lamb, G. Landis, J. Orenstein and D. Weinreb. The ObjectStore database system. *CACM* 34 (10), 1991.

[Liskov96] B. Liskov, et al. Safe and efficient sharing of persistent objects in Thor. *SIGMOD*, June 1996.

[Little96] M. Little and S. Shrivastava. Using application specific knowledge for configuring object replicas. *Third International Conference on Configurable Distributed Systems*, May 1996.

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

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

[Thekkath97] C. Thekkath, T. Mann and E. Lee. Frangipani: a scalable distributed file system. *SOSP*, October 1997.

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