Consistency and Locking for Distributing Updates to Web Servers Using a File System

Randal C. Burns and Robert M. Rees
IBM Almaden Research Center

Darrell D. E. Long†
University of California, Santa Cruz

Abstract

Distributed file systems are often used to replicate a Web site's content among its many servers. However, for content that needs to be dynamically updated and distributed to many servers, file system locking protocols exhibit high latency and heavy network usage. Poor performance arises because the Web-serving workload differs from the assumed workload. To address the shortcomings of file systems, we introduce the publish consistency *model well suited to the Web-serving workload and implement it in the* producer-consumer *locking protocol. A comparison of this protocol against other file system protocols by simulation shows that producer-consumer locking removes almost all latency due to protocol overhead and significantly reduces network load.*

1 Introduction

Distributed file systems are a key technology for sharing content among many Web servers. They provide increased scalability and availability [1]. We are particularly interested in large scale systems that serve dynamically changing Web content with a distributed file system: files containing HTML or XML that are concurrently updated (written) and served to Web clients (read). A good example of such a system is IBM's Olympics web site [2] which used the DFS file system [3] to replicate changing data like race results at a global scale.

Distributed file systems provide synchronized access and consistent views of shared data, shielding Web servers from complexity by moving these tasks into the file system. Most often file systems implement sequential consistency [4], where all process see changes as if they were sharing a single memory. However, Web servers do not require sequential consistency, because the data that they serve through the HTTP protocol does not need an instantaneous consistency guarantee. Web servers benefit from weakening consistency in order to achieve better performance.

Parallelism in the workload leads to performance problems when using an unmodified distributed file system to share dynamically updated data among Web servers. File systems assume that file data has affinity to a few clients at any one time. However, when serving Web content, load is balanced uniformly and all Web servers have equal interest in all files. For content that changes frequently, traditional file system consistency protocols utilize the network heavily and exhibit high latency.

Performance concerns aside, sequential consistency is the wrong model for updating Web data, since it results in errors when Web clients parse HTML or XML content. When a reader (Web server) and a writer (content publisher) share a sequentially consistent file, the reader sees each and every change to file data. Consequently, a Web server can see files that are in the process of being modified or rewritten. Such files are either incomplete or contain data from both the new and old version and cannot be parsed. However, before the writer begins and after the writer finishes, the file contains valid content. A more suitable model has the Web server continue to serve the old version of the file until the writer finishes. We call this *publish consistency*.

A more formal definition of publish consistency is based on the concepts of *sessions* and *views*. In a file system, the open and close function calls define a session against a file. Associated with each session is that session's image of the file data that we call a view. Data is publish consistent if (1) write views are sequentially consistent, (2) a reader's view does not change during a session, (3) a reader creates a session view consistent with the close of the most recent write session of which it's aware, and (4) readers eventually become aware of all write sessions. When opening a file, a reader obtains a view of the file data and uses it for an entire session. Modifications to files are not propagated to all readers instantaneously. However, all readers learn about all modifications eventually.

In publish consistency, relaxed synchronization allows us to implement cache coherency data locks that improve performance. We took the Web-serving workload into account when building producer-consumer locking for publish consistency. These locks efficiently replicate changes from a computer holding a producer lock (for writing) to the many computers holding consumer locks (for reading). These locks eliminate read latency, because each Web server that holds a consumer lock always has a recent view in its cache. Producer-consumer locks also reduce network usage by reducing the number of messages needed to update files to reflect recent changes.

†The work of this author was performed while a Visiting Scientist at the IBM Almaden Research Center.

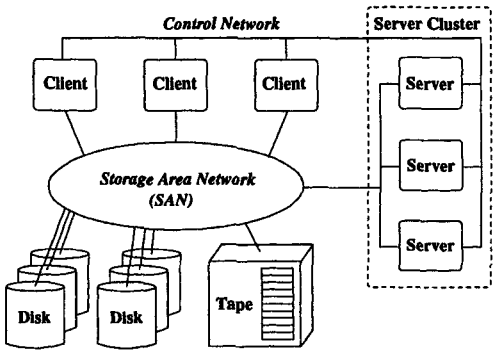


Figure 1: Storage Tank.

1.1 A Direct Access File System

A brief digression into the file system architecture that implements producer-consumer locking helps motivate our solution and its advantages. In the Storage Tank project at IBM research we are building a distributed file system on a storage area network (SAN). A SAN is a high speed network designed to allow multiple computers to have shared access to many storage devices.

For our distributed file system on a SAN, clients access data directly over the storage area network. Most traditional client/server file systems [5, 6, 3] store data on the server's private disks. Clients dispatch all data requests to a server that performs I/O on their behalf. Unlike traditional file systems, Storage Tank clients perform I/O directly to shared storage devices on a SAN (Figure 1). This direct data access model is similar to the file system for Network Attached Secure Disks [7] that uses shared disks on an IP network and the Global File System [8] for SAN attached storage devices. Clients communicate over a separate control network for protocol messages – locking and metadata.

Our SAN file system and producer-consumer locks have synergy. In particular, by separating metadata traffic from data traffic, we efficiently distribute new versions of files by sending only metadata about the new version to the client rather than the file data. Clients obtain file data later in order to service HTTP requests and then only need to acquire the changed portions, rather than re-reading the whole file. This differs from non-SAN producer-consumer implementations in which the whole new file must be pushed to the consumers. Our experiments present comparisons of producer-consumer locking and other locking systems for both SAN and traditional distributed file systems.

2 Locking for Wide Area Replication

The limitation of using a file system for the wide-area replication of Web data is performance. Generally, file systems implement data locks that provide sequential consistency. For distributing Web data from one writer to multiple readers, sequential consistency produces lock contention. Contention loads the network and results in slow application progress. We illustrate lock contention, propose a new set of locks that address this problem, and show how our SAN file system takes unique advantage of these locks.

2.1 Sequential Consistency

We use posting race results in an Olympics Web site [2] as an example application to show how sequential consistency locking works for Web servers. A possible configuration of the distributed system (Figure 2) has hundreds or thousands of Web servers integrated with a DBMS. Race results and other frequently changing data are inserted into the database through an interface like SQL or ODBC. The insertion updates database tables and sets off a database trigger that creates new Web content. The file system ensures that the new version of the content is consistent at all Web servers.

Poor performance occurs for a sequential consistency locking protocol when updates occur to active files. For this example we assume that the file is being read concurrently by multiple Web servers before, during, and after new results are being posted and written. For race results during the Olympics, a good example would be a page that contains a summary of how many gold, silver, and bronze medals have been won by each country. This page has wide interest and changes often. We use sequential consistency locks on whole files for this example (the same locks used by DFS [3].) The system's initial configuration has the clients at all Web servers holding a shared lock (S) for the file in question. Figure 3(a) displays the locking messages required to update file data in a timing diagram.

Sequential consistency locking performs best when the file system client at the database is not interrupted while updating. In this case, the writing client requests an exclusive lock (X) on the file. The exclusive lock revokes all concurrently held shared locks. After the writer completes, the client at each Web server must request a shared lock on the file to read and serve the Web content. All messages to and from the Web server occur for every server. In the best case, four messages – revoke, release, acquire, and grant – go between each Web server and the file system server.

The situation is much worse when the file system at the DBMS is interrupted while updating the file. In file system protocols, data locks are preemptible so that the system is responsive when multiple clients share file data. For example, clients that request read locks on the file data can revoke the DBMS's exclusive lock before

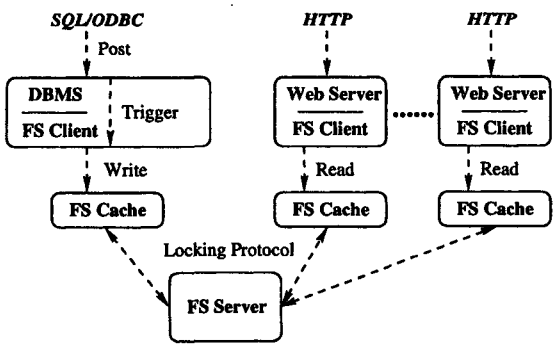


Figure 2: Replicating data among many Web servers.

it completes writing. The DBMS must reobtain an exclusive lock to finish. The more Web servers there are, and the more likely the update will be interrupted. Contention for the lock can stall the update indefinitely.

The example application presents a non-traditional workload to the file system. The workload lacks the properties a file system expects and therefore operates inefficiently. Specifically, the workload does not have client locality [9] – the affinity of a file to a single client. Instead, all clients are interested in all files.

Performance concerns aside, sequential consistency is the wrong model for updating HTML and XML. As we discussed in Section 1, reading clients cannot understand intermediate updates and byte-level changes. So sequential consistency leaves content invalid when Web servers see modifications to files at a fine granularity. Publish consistency addresses this problem by allowing writers to distribute updated views only at the end of a write session.

2.2 AFS Consistency

Among existing systems, the Andrew file system (AFS) comes closest to publish consistency. AFS does not implement sequential consistency. Rather, it chooses to synchronize file data between readers and writers when files opened for write are closed. Previous research [9] argues that data are shared infrequently to justify this decision.

Figure 3(b) shows the protocol used in AFS to handle the dynamic updates in our example. At the start of our timing diagram all Web servers hold the file in question open for read. In AFS an open instance registers a client for *callbacks* – messages from the server invalidating their cached version. The DBMS opens the file for writing, writes the data to its cache, and closes the file. On close the cached copy of the file is written back to the server. The file server notifies the Web servers of the changes by sending invalidation messages to clients that have registered for a callback.

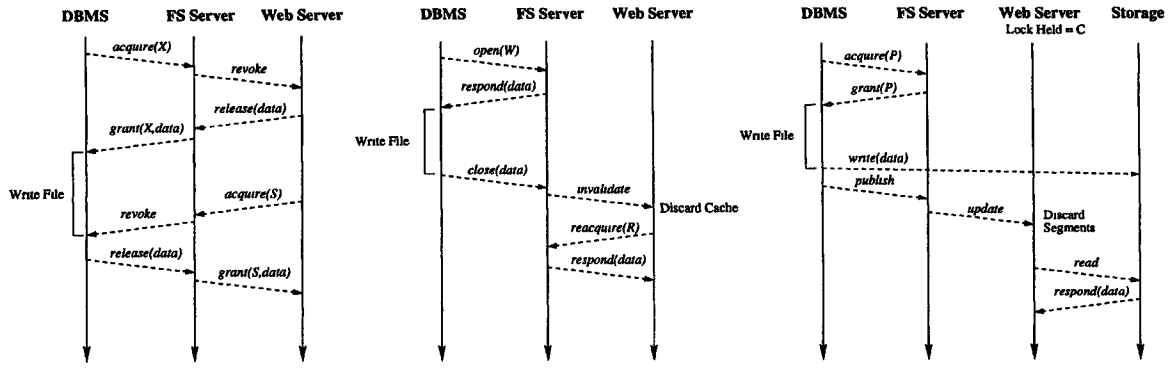
When compared with the protocol for sequential consistency, AFS saves only one message between client and server. However, this belies the performance difference. In AFS all protocol operations are asynchronous; *i.e.*, operations between one client and a server never wait for actions on another client. Using the AFS protocol, the DBMS obtains a write instance from the server directly and does not need to wait for a synchronous revocation call to all clients. Another significant advantage of AFS is that the old version of the file is available at the Web servers concurrently with it being updated at the DBMS.

The disadvantage of AFS is that it does not correctly implement publish consistency. The actual policy implemented at the client is to forward changes to a file to reading clients whenever a writing client submits modifications to the file system server. In general, AFS clients write modified data to the server when closing a file, which results in publish consistency. However, sometimes AFS writes data to a server before the file has been closed. This occurs in two ways. First, when a file has been open for more than 30 seconds a timer writes modified file data to a server. Second, if a client's cache becomes full it may send modified file data to a server to make free space in its cache. In either case reading clients can see partial updates which is a violation of publish consistency.

2.3 Implementing Publish Consistency

We offer a locking option that improves performance when compared with AFS and sequential consistency, implements publish consistency correctly, and takes advantage of the SAN architecture. Our locking system reduces the protocol latency that a Web Server sees when accessing data to nearly nothing. There is a one time cost to initially access data, but all subsequent reads are immediate. Furthermore, for the Web-serving workload, our protocol lessens the network utilization by reducing the number of messages to keep a file consistent among many Web servers.

We capture publish consistency in two data locks: a producer lock (P) and a consumer lock (C). Any client can obtain a consumer lock at any time, and a producer lock is held by a single writer. Clients holding a consumer lock can read data and cache data for read. Clients holding a producer lock can write data and allocate and cache data for writing, with the caveat that all modifications use copy-on-write semantics to retain a read-only version for readers and an active version used by the writer. File publication occurs when the P lock holder releases its lock (Figure 3(c)). Upon release the server notifies all clients of the new location of the file. Recall that clients read data directly from the SAN. Servers do not transmit data to the clients.



(a) Sequential consistency (b) AFS consistency (c) Publish consistency

Figure 3: Locking protocols for distributing data to Web servers.

Clients that receive a publication invalidate the old version of the file and, when interested, read the new version from the SAN. The client need not invalidate the whole new file. Instead, it only needs to invalidate the portions of the file that have changed. Because changed blocks have been written out-of-place, the client knows that portions of the file for which the location of the physical storage has not changed have not been written and its cached copy is valid.

Looking at our example using C and P locks, we see that the protocol uses fewer messages between the writing client and server than both AFS and DFS, and, like AFS, all operations are asynchronous. The DBMS requests and obtains a producer (P) lock from the server. When the DBMS completes its updates, it closes the file and releases the P lock. The writer must release the P lock on close to publish the file. The server sends updates that describe the location on the SAN of the changed data to all clients that hold consumer locks. The clients invalidate the affected blocks in their cache.

Producer-consumer locking saves messages when distributing updates to readers. In producer-consumer locking, changes are pushed to clients in a single update message. In contrast, AFS and sequential consistency require data to be revoked and then later reobtained.

C and P locks dissociate updating readers' cache contents from writing data. With C and P locks, the Web servers are not updated until the the P lock holder releases the lock. This means that the P lock holder can write data to the SAN and even update the location metadata at the server without publishing the data to reading clients.

For our SAN file system, this design assumes that storage devices cache data for read. The Web servers read data directly from storage, and they all read the same blocks. With a cache, a storage controller can service most reads at memory speed. Without a cache, all reads would go to disk and any communication savings of producer-consumer locking could potentially be lost by repeatedly reading the same data. Most often SAN storage systems are implemented with caches and reliability features (RAID and/or mirroring.)

Producer-consumer locking is not restricted to SANs, and analogous protocols are possible for traditional file systems. We will show that even for traditional architectures producer-consumer locking reduces network utilization and has near-zero read latency.

Producer-consumer locking on traditional file systems exacerbates a performance problem with this design – a phenomena we call *overpublishing*. Overpublishing occurs when the server publishes versions that clients do not read. The publish message has network costs that are never recovered by avoiding future server requests.

On a SAN, publishing only sends location information, a small amount of data. The client reads the published file data only when requested. However, on a traditional file system the whole file contents are transmitted and publishing data consumes network bandwidth for write-dominated workloads.

The overpublishing phenomena is not of concern for Web serving in general and in particular for Web serving with Storage Tank. Producer-consumer locking is designed to address a specific, yet important, workload. The Web-serving workload consists mostly of reads on many Web servers: writes are less frequent. The frequency of reads prevent this workload from exhibiting overpublishing.

3 Simulation Results

We constructed a discrete event simulation of the presented locking protocols to verify our design and understand the performance of the various protocols for a Web-serving workload. The goal of this simulation is to compare the network usage and latency of sequential

consistency, AFS consistency, and producer-consumer locking. We present results both for file systems running without a SAN and for the locking protocols running on a SAN file system architecture. For simulations on SAN hardware, we have reformulated AFS and sequential consistency protocols to separate data and metadata.

The discrete event simulation was constructed using the YACSIM toolkit [10]. We simulated the locking protocols through four different components: a reading client, a writing client, a network, and a protocol server. These components conform with Figure 2.

We conducted multiple experiments varying the write interval and number of Web servers while keeping the read rate constant. Write intervals were varied to explore the affects of lock contention on the system. At lower write intervals clients refresh their cache with new versions more frequently. At higher write intervals clients obtain a new version of a file less frequently and service many read requests against that version out of their cache. For all clients, reads occur according to a Poisson process with a mean rate of 1 read per second at each client. We varied write intervals between 1 and 60 seconds. However, we present results only for 10 second write intervals.

3.1 Latency

The latency results (Figure 4) describe the time interval between an incoming read request and the file being available at the client for reading. When the file is not immediately available at the client, the client is assessed the time required to communicate with the server, the server to obtain the data through the protocol, and the server to deliver the data. When the data is already at the client, the read occurs in no time. Because we do no charge any time for processing the file locally, we measure protocol latency, rather than overall read latency.

Results (Figure 4) show latency in seconds as a function of the number of reading clients in the system for a write intervals of 10 seconds. The graphs contains curves for publish consistency (PC), sequential consistency (SC), and AFS consistency (AFS).

For publish consistency, latency is negligible at all times. In this protocol, a read lock is obtained once at the start of the simulation and is never revoked. A reading client always has data available. Consequently, this result is trivial.

For the sequential consistency and AFS protocols, latency increases super-linearly with the number of reading clients. When the writing client modifies the file, each reading client must have its cache invalidated and then must request a new lock from the server to reobtain data. All of these messages share the same network resource and use the resource at the same time. More clients results in more resource contention and therefore more latency.

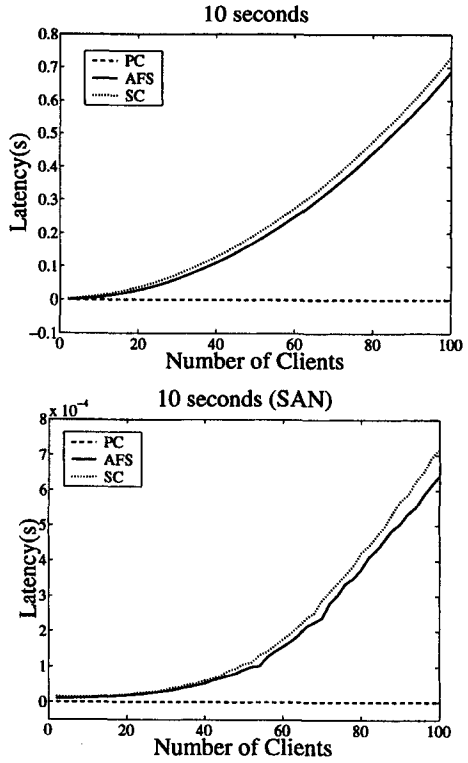


Figure 4: Protocol latency for readers.

Contention accounts for the super-linear growth of the latency curves for AFS and sequential consistency protocols and serialization between readers and writers accounts for the difference between these curves. Sequential consistency invalidates read copies before writing data whereas AFS updates data and then invalidates the readers. Sequential consistency adds latency while the clients wait for the writer to obtain its write lock, modify the file, and release the lock.

Reduced read latency is the key performance advantage of publish consistency when compared with sequential and AFS consistency. Producer-consumer locking removes almost all read latency. This dramatic improvement is most important when the number of clients becomes large.

3.2 Network Usage

The network usage results (Figure 5) describe the utilization of the network resource shared by Web servers. Utilization can be considered the fraction of the networks total capacity that a protocol uses or equivalently, for an ideal network, the percentage of time that the network is busy.

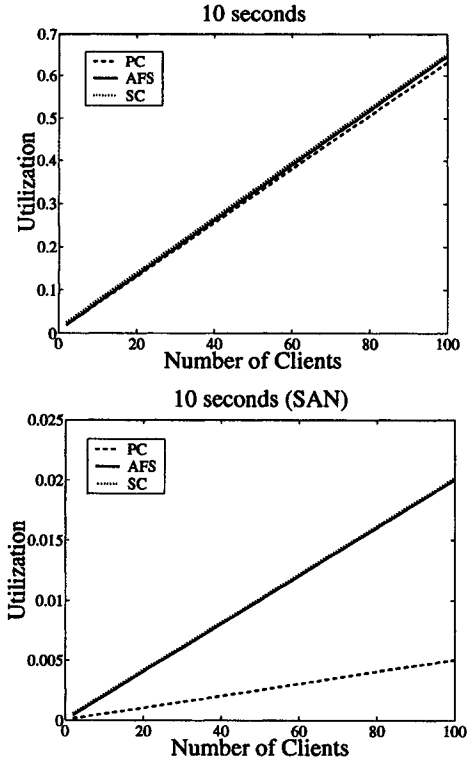


Figure 5: Network utilization.

For all locking protocols network usage increases linearly with the number of reading clients. This relationship holds until the network reaches capacity. Unlike latency, contention has no effect on network simulation results. Protocols always send the same number of messages to distribute updates.

The network complexity of the different protocols accounts for the differences in the results. The producer-consumer locking protocol saves messages between the servers and reading clients when compared with AFS and sequential consistency. Therefore, as the number of clients increase, the savings increase commensurately. The network usage of producer-consumer locking therefore grows at a different rate than AFS and sequential consistency protocols.

In traditional file systems not all messages are equal. The messages saved by AFS and publish consistency locking are protocol only messages and do not contain data. They are not as expensive as the publish and grant messages that contain file data. However, in the SAN environment, no messages contain data and the reduced message complexity has a larger effect.

3.3 Overpublishing

Producer-consumer locks exploit the properties of the Web-serving workload to reduce latency and network utilization. However, as we discussed in Section 2.3, when workload assumptions are incorrect, producer-consumer locks exhibit a behavior called overpublishing.

To simulate overpublishing, we varied the rate at which client read requests are serviced, focusing on read rates that are slower than the rate at which a file is being written and published. We ran the simulation for 100 reading clients. In Figure 6, we see that when reads occur less frequently than writes in non-SAN file systems, producer-consumer locking utilizes the network resource more heavily than AFS or sequential consistency. For a fixed number of clients, the number of protocol messages in the producer-consumer protocol is approximately the same regardless of read workloads.

Overpublishing is insignificant for SAN file systems, because the fixed cost of publishing just metadata is so small.

4 Related Work

Consistency describes how the different processors in a parallel or distributed system view shared data. Lamport defined the problem and introduced sequential consistency [4]. Subsequent work on multiprocessors and shared memories introduced many other similar but more liberal models [11].

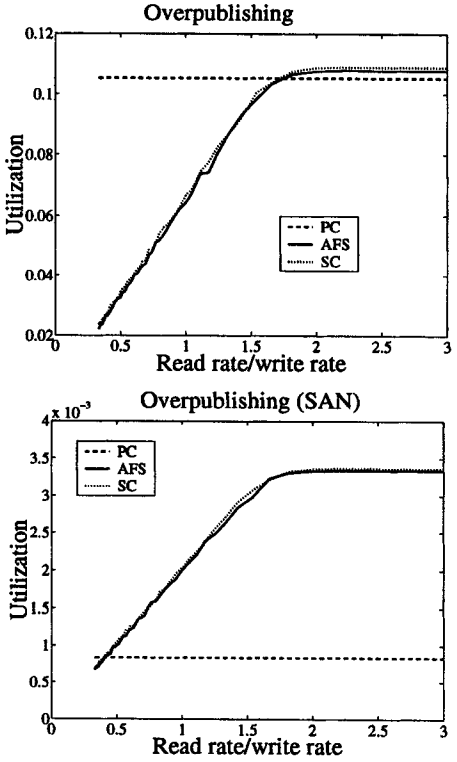


Figure 6: Overpublishing (network utilization.)

Consistency in file systems is derived directly from multiprocessor research. Most modern file systems, including DFS [3], Frangipani [12], and xFS [13], implement sequential consistency. Other file systems implement consistent views in a less strict manner. NFS updates views based on time [5]. Other systems like AFS [9] update views whenever the server is aware of changes, but do allow distributed data views to be inconsistent at any point in time. However, no file systems known to the authors provide delayed update propagation based on sessions views as we do with producer-consumer locking.

Research on view management in database systems also has similarity to publish consistency, because databases implement multiple versions with session semantics. However, an inspection of the details reveals that the data structures and techniques share little in common. This area is broad and well developed. Bernstein et al. present a nice overview of concurrency control and view management [14].

The publish and subscribe paradigm is increasingly used as a simple and efficient mechanism for information routing and multicast in object based distributed systems [15]. This work differs in that publishing is explicit, rather than implicit in the data consistency model.

5 Conclusions

We have remedied the semantic and performance shortcomings of using a file system for distributing dynamic updates to Web content in a large scale system. With the sequential consistency implemented by most file systems, Web servers can see incomplete changes to file data that can result in parsing errors at Web clients. We have developed the publish consistency model that gives Web servers session semantics to files. Publish consistency prevents parsing errors.

From publish consistency we derived the producer-consumer cache coherency locks that reduce network utilization among Web servers and eliminate protocol latency. Simulation results verify these claims. Producer-consumer locks reduce the messages required to distribute updates to file system clients and thereby reduce network load. With existing file system protocols, latency grows more than linearly with the number of reading clients. Producer-consumer locking eliminates all protocol latency from the reading clients, excepting startup costs to obtain the first lock.

References

[1] S. Hannis, "AFS as part of the IBM WebSphere performance pack," in *Proceedings of Decorum '99*, Mar. 1999.

[2] "Transarc's DFS provides the scaleablility needed to support IBM's global network of Web servers." http://www.transarc.com/Solutions/Studies/EFS_Solutions/DFSOly/dfsolympic.htm, 1999.

[3] M. L. Kazar, B. W. Leverett, O. T. Anderson, V. Apostolides, B. A. Bottos, S. Chutani, C. F. Everhart, W. A. Mason, S. Tu, and R. Zayas, "DEcorum file system architectural overview," in *Proceedings of the Summer USENIX Conference*, June 1990.

[4] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE Transactions on Computers*, vol. C-28, no. 9, 1979.

[5] D. Walsh, B. Lyon, G. Sager, J. Chang, D. Goldberg, S. Kleiman, T. Lyon, R. Sandberg, and P. Weiss, "Overview of the Sun network file system," in *Proceedings of the 1985 Winter Usenix Technical Conference*, Jan. 1985.

[6] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West, "Scale and performance in a distributed file system," *ACM Transactions on Computer Systems*, vol. 6, Feb. 1988.

[7] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, H. Gobioff, E. Riedel, D. Rochberg, and J. Zelenka, "Filesystems for network-attach secure disks," Tech. Rep. CMU-CS-97-118, School of Computer Science, Carnegie Mellon University, July 1997.

[8] K. W. Preslan, A. P. Barry, J. E. Brassow, G. M. Erickson, E. Nygaard, C. J. Sabol, S. R. Soltis, D. C. Teigland, and M. T. O'Keefe, "A 64-bit, shared disk file system for Linux," in *Proceedings of the 16th IEEE Mass Storage Systems Symp.*, 1999.

[9] M. L. Kazar, "Synchronization and caching issues in the Andrew file system," in *Proceedings of the USENIX Winter Technical Conference*, Feb. 1988.

[10] J. R. Jump, "YACSIM reference manual," Tech. Rep. available at http://www-ece.rice.edu/~rsim/rppt.html, Rice University Electrical and Computer Eng. Dept., Mar. 1993.

[11] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: A tutorial," *IEEE Computer*, vol. 29, Dec. 1996.

[12] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A scalable distributed file system," in *Proceedings of the 16th ACM Symposium on Operating System Principles*, 1997.

[13] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang, "Serverless network file systems," *ACM Transactions on Computer Systems*, vol. 14, Feb. 1996.

[14] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Publishing Company, 1987.

[15] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman, "An efficient multicast protocol for content-based publish-subscribe systems," in *Proceedings of the 19th International Conference on Distributed Computing Systems*, May 1999.