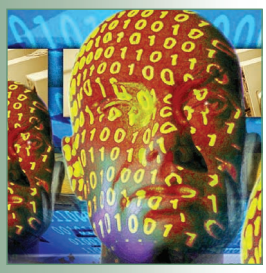

Efficient Data Distribution in a Web Server Farm

A novel locking protocol maintains data consistency in distributed and clustered file systems that are used as scalable infrastructure for Web server farms.

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

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

High-performance Web sites rely on Web server "farms"—hundreds of computers serving the same content—for scalability, reliability, and low-latency access to Internet content. Deploying these scalable farms typically requires the power of distributed or clustered file systems.

Building Web server farms on file systems complements hierarchical proxy caching.[1] Proxy caching replicates Web content throughout the Internet, thereby reducing latency from network delays and off-loading traffic from the primary servers. Web server farms scale resources at a single site, reducing latency from queuing delays. Both technologies are essential when building a high-performance infrastructure for content delivery.

In this article, we present a cache consistency model and locking protocol customized for file systems that are used as scalable infrastructure for Web server farms. The protocol takes advantage of the Web's relaxed consistency semantics to reduce latencies and network overhead. Our hybrid approach preserves strong consistency for concurrent write sharing with time-based consistency and push caching for readers (Web servers). Using simulation, we compare our approach to the Andrew file system and the sequential consistency file system protocols we propose to replace.

Data Consistency

File system design carries assumptions about workloads and application consistency needs that, when applied to Web serving, lead to inferior performance. File system designers assume that data are shared infrequently and that such data require strong (sequential) consistency.[2] Consistency describes how processors in a parallel or distributed system view shared data; the sidebar on related work in data consistency presents examples.

For Web serving, data are widely shared among many servers, as Figure 1 shows, and strong consistency guarantees

 1089-7801/01/$10.00 ©2001 IEEE

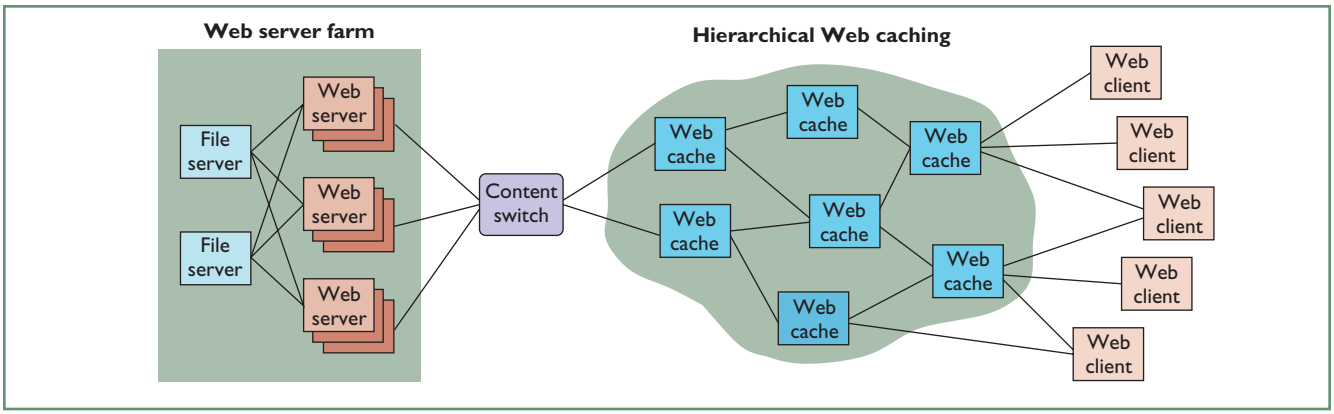

Figure 1. Server farms and hierarchical caches in the Web.

are irrelevant because they cannot be passed on to users (or applications) at the Web browser. The hypertext transfer protocol (HTTP) provides little consistency, and any guarantees the file system provides are lost in transit. System designers can trade consistency for performance in middleware, databases, and Internet applications,[3] and likewise in file systems.

We introduce a hybrid approach to enhance file system consistency for Web serving. The *publish consistency* model applies to data views that are stable during a given file's open/close session. This model relaxes semantics, permitting data to be read and written asynchronously, in parallel. We implement publish consistency in the *producer/consumer* locking protocol to replace callback invalidations—messages from the server invalidating cached versions—of stale data with push caching,[4] transmitting updates to Web servers. With push caching, Web server data are immediately available. However, we preserve strong consistency for writers, supporting fine-grained concurrent write sharing for applications (such as databases) that author Web data.

The producer/consumer (PC) protocol is particularly appropriate for files likely to be modified frequently, like stock quotes, weather information, and live image/data feeds (for example, Web cameras). The protocol does not address a Web site's referential (link) integrity; a protocol operating at file granularity cannot enforce consistency guarantees between multiple files that make up a site.

Publish consistency and producer/consumer locking are effective file system modifications that increase scalability and performance in Web server farms. Reduced network overheads allow for the deployment of more servers on a given network infrastructure, and lowered latencies allow more HTTP requests to be served on any given Web server. File systems give Web servers access to a rich data-sharing and data-management environment. By using a customized consistency model and protocol, file systems provide a scalable infrastructure for Web server farms without adding latency.

Locking for Content Distribution

While the sequential consistency (SC) model—in which all processes see things as if they shared a single memory—is correct for many applications, Web servers don't require sequential consistency because they serve data through the weakly consistent HTTP protocol. Web-serving performance benefits from weakened file system consistency.

Performance concerns aside, sequential consistency is the wrong model for updating Web data, as it can result 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 change to file data, including files in the process of being modified. The Web server distributes these files to its clients where parsing errors occur when files are incompletely written. However, before the writer begins and after the writer finishes, the file contains valid content. A more suitable publish-consistency model has the Web server continue to serve the old version of the file until the writer finishes.

More formally, publish consistency is based on the concepts of *sessions* and *views*. In file systems, open and close function calls define a file session. Associated with each session is a view, or image, of the file data. 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 a recent write session, and (4) readers become aware of all write sessions within a bounded amount of time.

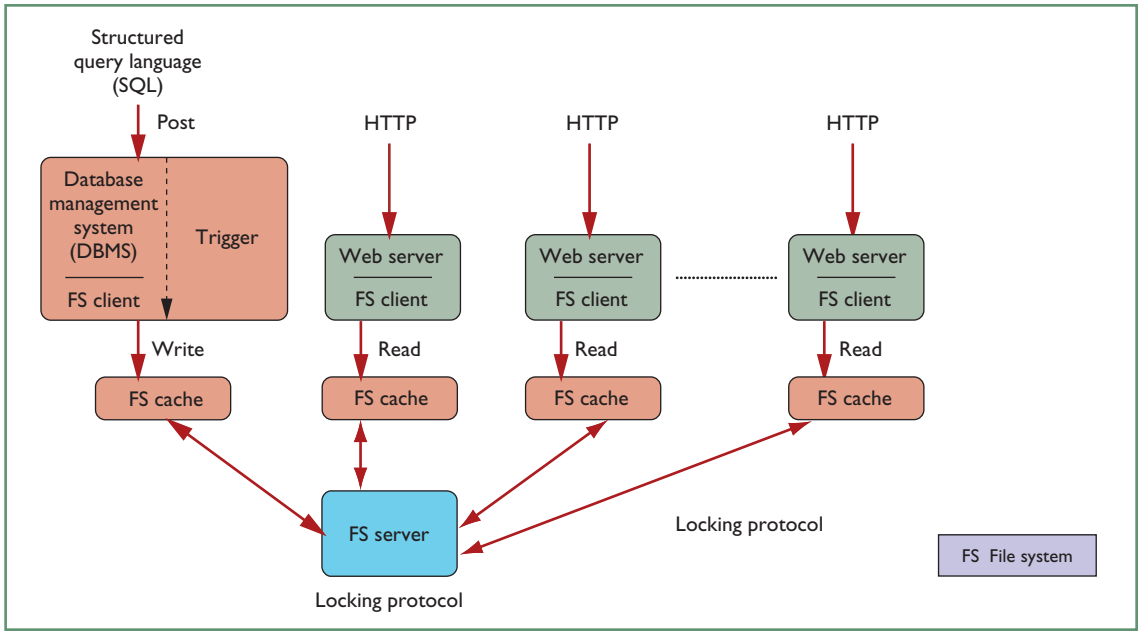

Figure 2. Example configuration for updating data among many Web servers.

When opening a file, a reader obtains and uses a view for an entire session. File modifications are not propagated to all readers instantaneously. Leases,[5] which are used to detect and recover from failures, bound the time until changes are propagated, similar to the delta consistency[6] and hierarchical leasing[7] models.

Sequential Consistency

For Web data distributed from one writer to multiple readers, SC produces lock contention, loading the network and slowing application progress. Contention increases with the number of servers in a Web server farm configuration, as in Figure 2, of many hundreds of Web servers integrated with a database management system (DBMS). Data updates to database tables, inserted through an interface like Structured Query Language (SQL), trigger Web content creation. The file system ensures that the content's new version is consistent at all Web servers.

The SC locking protocol performs poorly when active files are modified, which occurs when a file is read by multiple Web servers before, during, and after new results are written. The system's initial configuration has all Web server clients holding a shared lock (S) for reading the file. Figure 3a displays the locking messages required to update file data.

SC 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, Web server clients must request a shared lock on the file to read and serve the Web content. All messages occur for every Web server. In the best case, four messages—revoke, release, acquire, and grant—are transmitted between each Web server and the file system server.

Performance is worse when the DBMS file system is interrupted during file updating. In file system protocols, data locks are preemptible so that the system is responsive when multiple clients share file data. Lock contention can stall updates indefinitely.

Web serving presents the file system with a nontraditional workload, which lacks the properties a file system expects and therefore operates inefficiently. Specifically, the workload lacks client locality[8]—the file's affinity to a single client. Instead, all clients are interested in all files.

AFS consistency. Among existing systems, the Andrew file system (AFS) comes closest to publish consistency. AFS does not implement sequential consistency but synchronizes file data between readers and writers when files are closed. The AFS protocol fails to implement publish consistency because it lacks session and view concepts.

As shown in Figure 3b, all Web servers first hold the file open for read. Then an open instance registers a client for callbacks. The DBMS opens the file for writing, writes the data to its cache, closes the file, and then writes the cached-file copy back to the server. The file server notifies other clients

of the changes by sending invalidation messages to clients that have registered for a callback.

AFS uses only one fewer message between client and server than the SC protocol, but this belies the performance difference. In AFS, operations between one client and a server never wait for actions on another client. Another significant AFS advantage is that the old file version is available at the Web servers concurrently with its being updated at the DBMS. Callback invalidations, however, prevent data from being continuously available.

A disadvantage of the AFS protocol is that it does not correctly implement publish consistency. The actual policy is to forward file changes to reading clients whenever a client writes data. Sometimes, however, AFS writes data before the file is closed. This occurs first, when a file has been open longer than 30 seconds and a timer writes back cached data, and second, when a client's cache becomes full and it writes data to a server to free memory. In either case, reading clients can see partial updates, which violates publish consistency.

Implementing publish consistency. The PC locking protocol implements publish consistency correctly and also improves performance. PC locking eliminates almost all protocol latency. Initial data access carries a onetime cost, but subsequent reads are immediate. Furthermore, for the Web-serving workload, our protocol lessens network utilization by minimizing the messages needed to keep a file consistent, as Figure 3c shows. In PC locking, a single update message pushes changes to clients. In contrast, the AFS and SC protocols require data to be revoked and then later re-obtained, which incurs latency as HTTP requests await the data's arrival.

We capture publish consistency in two data locks: a producer (*P*) and a consumer (*C*) lock. A producer lock can be held by only one client at a time and lets a client write and cache data for write. A consumer lock lets a client read and cache data. It can be obtained by any client.

Push caching improves PC locking performance. In push caching, the *P* lock holder sends an explicit publish message, and the server notifies all clients holding *C* locks of the updated file. Push caching keeps data publish-consistent and obviates callback invalidation. Therefore, readers always have valid cache data and never incur latency while waiting for data to arrive. PC locking also reduces the number of protocol messages by eliminating the messages needed to re-obtain data after a callback.

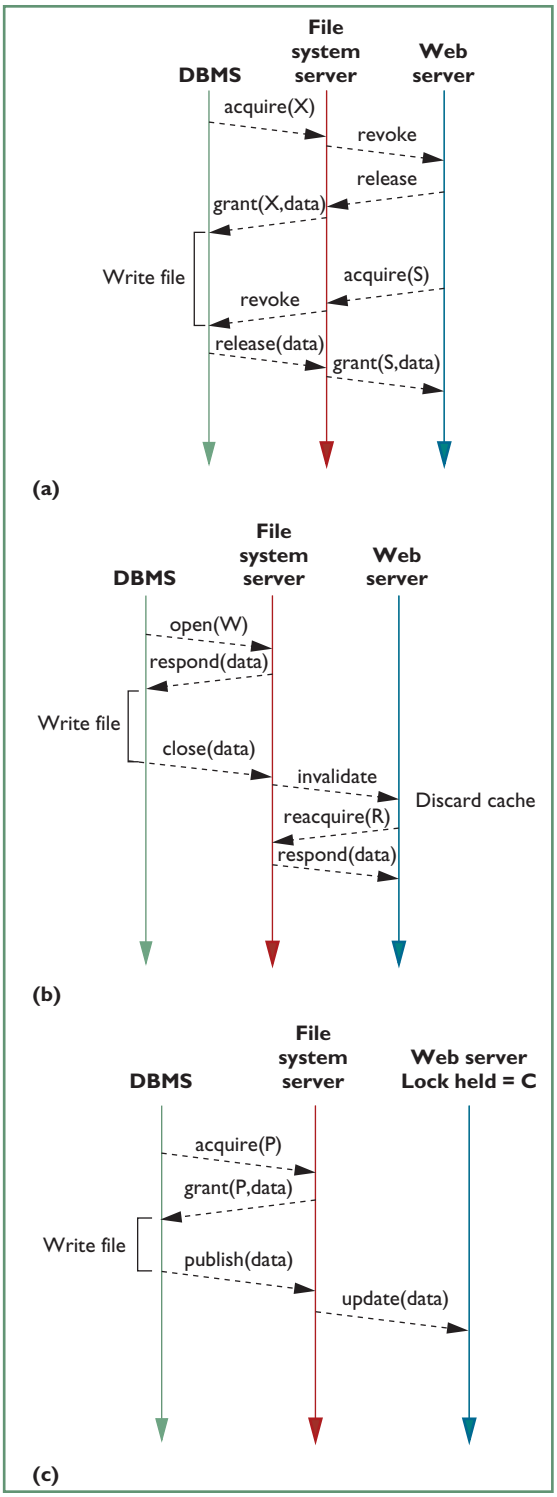

Figure 3. Locking protocols for distributing updates in a file system.

PC locking implements the session-and-view model of publish consistency. During a view, a reading client sees data consistent with the end of

Related Work on Data Consistency

Although Lamport introduced sequential consistency,[1] subsequent work on multiprocessors and shared memories introduced many related models. Most modern file systems implement sequential consistency. Less-strict consistency models include NFS, which updates data based on time.[2] AFS[3] updates data when a server is aware of changes but does allow inconsistencies. We're unaware, however, of any file systems that propagate updates based on session views as producer/consumer locking does.

The publish consistency model and producer/consumer locking protocol described in the main text have adapted Web-caching concepts to file systems. Many Web-caching protocols have explored trade-offs between consistency and performance. However, Web caches form multilevel hierarchies, so Web caching differs substantially from file systems, which are usually organized as client-server or peer-to-peer systems. Many Web protocols, such as those used by the Harvest cache[4] and Internet Cache Protocol (ICP),[5] cause data objects to expire on the basis of time-to-live (TTL). Client-acquired data are cached until they're invalid, then discarded. No server callbacks occur. Techniques to extend TTL include adaptive timers to reduce bandwidth and inconsistency.[6] Some researchers believe that the Internet requires stronger consistency guarantees, which can be achieved through callback-invalidation protocols similar to those used in file systems.[7]

Our work is more closely related to protocols employing push caching[8] with relaxed consistency, including those based on hierarchical leasing[9] and multicasting updates.[10] Researchers are currently combining these concepts into a standard protocol for the Internet.[11]

References

1. L. Lamport, "How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs," *IEEE Trans. Computers*, vol. C-28, no. 9, 1979.
2. D. Walsh et al., "Overview of the Sun Network File System," *Proc. 1985 Winter Usenix Tech. Conf.*, Usenix Assoc., Berkeley, Calif., Jan. 1985.
3. M.L. Kazar, "Synchronization and Caching Issues in the Andrew File System," *Proc. Usenix Winter Tech. Conf.*, Usenix Assoc., Berkeley, Calif., Feb. 1988.
4. A. Chankhunthod et al., "A Hierarchical Internet Object Cache," *Proc. 1996 Usenix Tech. Conf.*, Usenix Assoc., Berkeley, Calif., Jan. 1996.
5. D. Wessels and K. Claffy, "ICP and the Squid Web Cache," *IEEE J. Selected Areas in Comm.*, vol. 16, no. 3, Apr. 1998.
6. J.S. Gwertzman and M. Seltzer, "World-Wide Web Cache Consistency," *Proc. Usenix Ann. Tech. Conf.*, Usenix Assoc., Berkeley, Calif., Jan. 1996.
7. P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World Wide Web," *IEEE Trans. Computers*, vol. 47, no. 4, Apr. 1998.
8. J.S. Gwertzman and M. Seltzer, "The Case for Geographical Push-Caching," *Proc. Fifth Workshop Hot Topics in Operating Systems*, IEEE Computer Soc. Press, Los Alamitos, Calif., May 1995.
9. J. Yin et al., "Volume Leases for Consistency in Large-Scale Systems," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 4, July/Aug. 1999.
10. D. Li and D.R. Cheriton, "Scalable Web Caching of Frequently Updated Objects Using Reliable Multicast," *Proc. 2nd Usenix Symp. Internet Technologies*, Oct. 1999.
11. D. Li, P. Cao, and M. Dahlin, "WCIP: Web Cache Invalidation Protocol," tech. report draft-danli-wrec-wcip-01.txt, Internet Draft, Mar. 2001.

a previous write session. To achieve this, PC locking isolates reading clients from current writers. *C* and *P* locks dissociate updating readers' cache contents from writing data; the *P* lock holder can write data at the file system server without releasing updates to reading clients. On closing a file, the writing client sends an explicit publish message. Clients receiving an update replace the old file view with the new view. However, current reader sessions continue to access the view the client opened; only new clients see the most recent view.

PC locking requires support from both the file system client and server to implement views and guarantee session semantics. Different views must be open and served concurrently. To permit this, we allow for multiple instances of our file data structures at both the client and server. We associate a monotonically increasing view number with each instance. Clients, servers, and the client-server protocol perform operations in the context of this view number so that the file system can bind requests to the appropriate data. Old views are reclaimed by reference counting when no open instances remain and all clients have been updated to newer views.

Multiple versions of the same file must coexist on stable storage to support views. Clients aren't required to cache all data for open files; therefore, clients might need to read data from an old view even after the file has been updated. To support multiple versions of a stored file, not just in caches, our file system employs out-of-place writing, or copy-on-write. This technique writes file updates to new storage, rather than the storage from which they were read. Different views of the same file keep data in different storage addresses.

Copy-on-write is increasingly prevalent in modern file systems, like those choosing a log-structured[9] data layout. Distributed file systems[10] support copy-on-write as part of a snapshot facility for backup-restore, which is how we implement out-of-place writing.

PC locking has a potential performance problem in *overpublishing,* which occurs in write-dominated workloads when the server pushes versions that clients don't read. Push caching assumes that clients read each version of the file. If so, the band-

width cost of pushing data is less than that of a client later requesting data from the server. Overpublishing is not a Web-serving concern because reads dominate the workload. Although PC locking is not a general solution, it improves Web-serving performance.

Simulation Results

We constructed a discrete-event simulation that models a configuration of our system similar to that shown in Figure 2 to verify our locking design and to examine performance in a Web-serving workload. To compare the network usage and latency of SC, AFS, and PC locking protocols in Web-server farms, we studied two sample workloads: a stock ticker and a Web camera.

We built the discrete-event simulation with the YACSIM toolkit (http://www-ece.rice.edu/~rsim/rppt.html). We simulated the locking protocols through four components—a reading client, a writing client, a network, and a protocol server—conforming with the configuration in Figure 2.

We modeled the network component as an ideal Ethernet, conducting experiments for both Fast Ethernet and Gigabit Ethernet. An *ideal* network is fair—messages are delivered in the order that computers present them to the network—and carries its full bandwidth up to the channel capacity. In reality, a network is not guaranteed to be fair, but this is the expected behavior, particularly when network load is light. Furthermore, unfair behavior varies in type and frequency depending on the network hardware. The locking protocols themselves can be run on any network. We factored hardware specifics out by modeling the network as ideal and restricted our studies to low network utilization.

We modeled the workload at reading clients, corresponding to HTTP requests at Web servers, according to a heavy-tailed distribution (a Pareto random variable). Statistical studies have shown that heavy-tailed distributions characterize well the bursty nature of Internet traffic. While desirable for their accuracy, heavy-tailed distributions can lead to simulation instabilities,[11] so we also verified simulation results against more stable workloads, like Poisson read processes.

For write workloads in our system, we used data (stock ticker and Web camera) updated at fixed intervals.

Our choice to drive the simulation with an artificial workload reflects suitable trace-data availability. While many Web studies use traces to drive simulations,[7] we know of no Web traces correlating reads and writes across a large installation. Other consistency studies have used synthetic workloads for similar reasons.[12] In all experiments, we focused on the network usage and latency associated with keeping a single file consistent in a Web farm. Simulation and a synthetic workload let us model large-scaled systems; however, without suitable traces, it is unreasonable to infer an overall system workload and study interactions between files.

Web Cameras

For the Web camera workload, PC locking eliminates substantial latencies for Web serving. The workload was moderately sized, consisting of 64-Kbyte files updated relatively infrequently: one-minute intervals, 10 hits per second at each Web server. Many Internet data sources have these change semantics, most containing larger data sets that would exacerbate latency problems. Two large classes of such data are geographic information and live images.

The latency results in Figure 4 (next page) describe the time interval between an incoming read request and the file's availability at the client. When the file was not immediately available, the client was assessed the time required to communicate with the server, the server to revoke locks (this step is protocol-specific), and respond to the clients. When data were already at the client, the read occurred immediately. The latency results include average latencies (all requests) and latencies for cache misses (all reads with nonzero latency).

An ideal network carries its full bandwidth up to the channel capacity.

For the PC locking protocol, latency is always negligible. A read lock is obtained once at the simulation's start and never revoked. A reading client always has data available; consequently, this result is trivial. When the writing client modifies data, the server sends an update to the reading client, replacing the old version. The client always holds a valid copy of a file.

For the SC and AFS protocols, latency increases superlinearly with the number of reading clients. When the writing client modifies the file, each reading client must have its cache invalidated and request a new lock to re-obtain data. All messages share the same network resource and use it simultaneously. More clients increase both resource contention and latency.

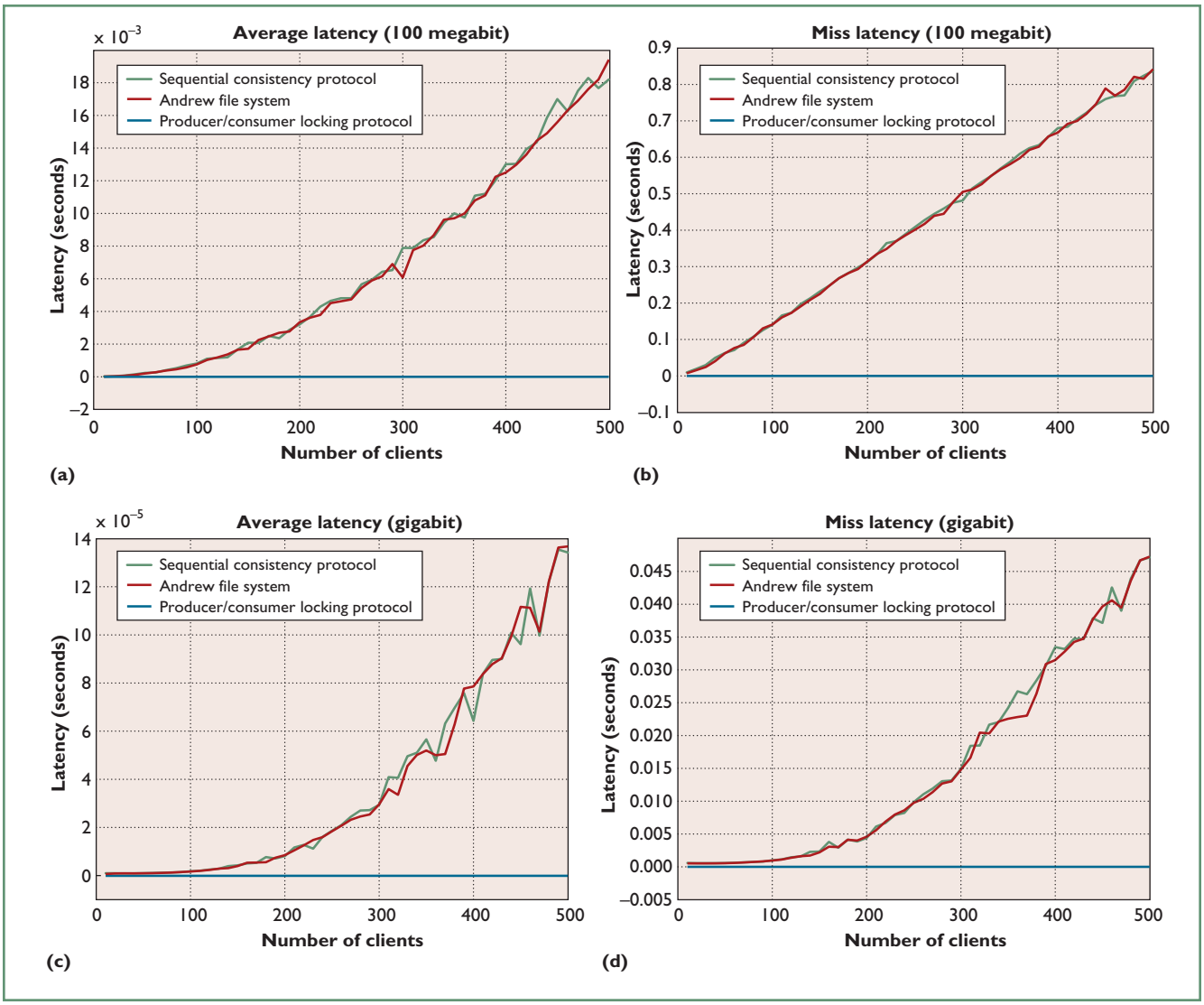

Figure 4. Read latency results on Fast and Gigabit Ethernets for the Web camera workload.

Latency reductions significantly affect end-to-end latencies in the Web. Previous research indicates that average Web latencies range from 100 to 200 milliseconds throughout the cache hierarchy.[13] On Fast Ethernets, we save approximately 20 milliseconds at just the highest level of this hierarchy for server farms of 500 computers. For cache misses, we reduce latency by almost a full second. Even on Gigabit networks, we reduce latency by 50 milliseconds for cache misses. These latency results are for single files. As server farms expand and the files to be kept consistent multiply, savings become more significant.

Stock Ticker

In the stock ticker workload, the PC locking protocol reduces the network overhead of keeping data consistent. The workload consisted of frequently updated files containing minimum data. We modeled files of about 100 bytes (so that with packet overheads and metadata, update messages were 256 bytes). These files were rewritten every second and hit one hundred times per second on each Web server. Many Internet data sources—stock and financial quotes, sports scores, statistical sources, and so on—have these change semantics.

While latency results still showed performance improvements, the benefit of PC locking on this workload lay elsewhere. Again, PC locking exhibited no latency, as Figure 5 shows, where AFS and SC showed superlinear growth. However, the latency's absolute values were so small that the savings were not important to overall performance. Average latency for 100-megabit networks was less than 40 microseconds for a Web farm of 200 machines, and miss latency was less than 3

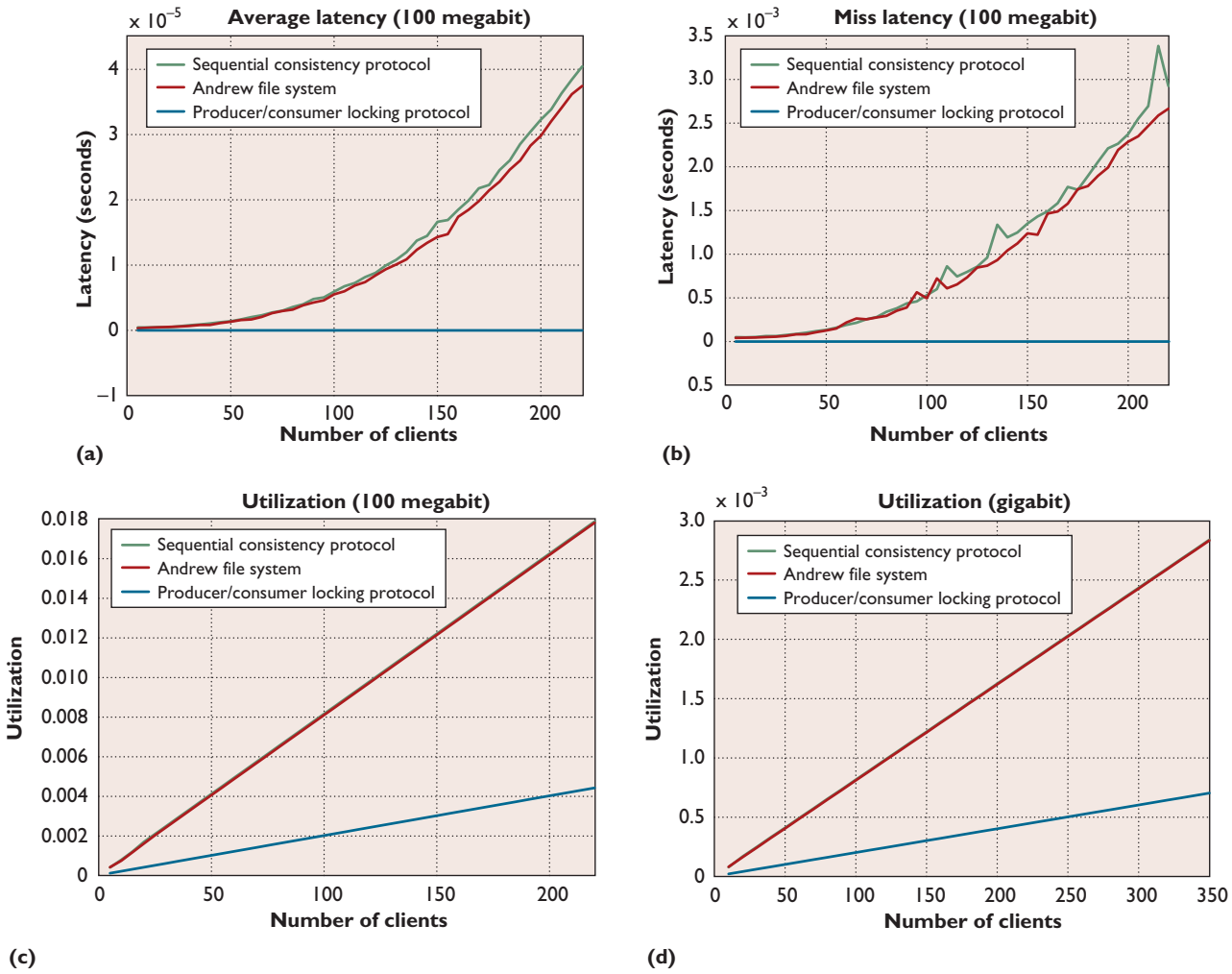

Figure 5. Read latency and network utilization results on Fast and Gigabit Ethernets for the stock ticker workload.

milliseconds. For end-to-end latencies estimated to be in the hundreds of milliseconds, these values are negligible.

Because the data in update messages are small, the reduced message complexity of PC locking results in significant network savings. The results in Figure 5 describe network resource utilization shared by Web servers. Utilization is the fraction of the network's total capacity that a protocol uses or equivalently, for an ideal network, the percentage of time that the network is busy. For all locking protocols, network usage increases linearly with the number of reading clients. Unlike latency, contention is not a factor in network utilization.

Network utilization limits the scalability of consistency protocols. For Fast Ethernet, 200 Web servers require more than 1.5 percent of all bandwidth to keep a single file consistent. Network usage scales linearly with the number of files (a best-case assumption), and 70 such files would saturate the interconnect. Similarly, 700 files would saturate a gigabit network with 200 servers. PC locking extends the number of files that can be kept consistent to more than 300 and 3,000, respectively. We are concerned with scalability in number of files and in number of Web servers, for which network usage also scales linearly. In any case, plausible combinations of files and servers saturate even gigabit networks, an effect that PC locking can mitigate.

For larger files, like our Web camera workload, PC locking offers only a minor utilization advantage. The network savings of PC locking are from saving messages; however, inspection of the protocols in Figure 3 reveals that the saved messages contain no data. For larger data, the cost of send-

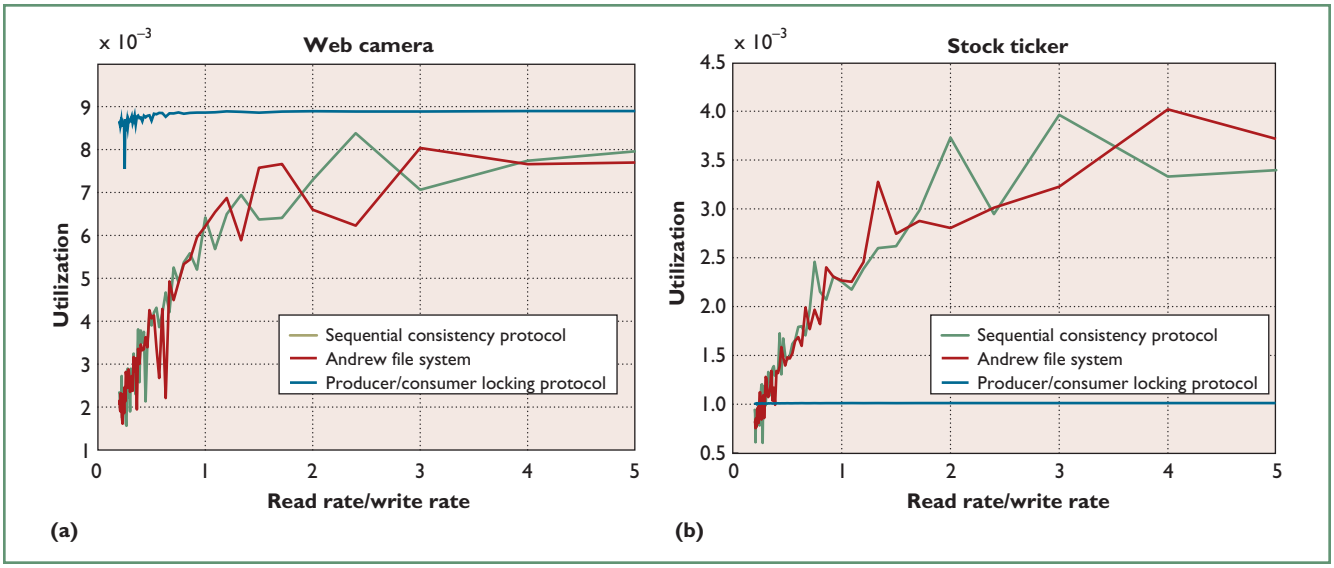

Figure 6. Overpublishing (network utilization).

ing the data dominates any savings. The only way to reduce network costs for large files is to update them less frequently.

Overpublishing

PC locks exploit the properties of the Web-serving workload to reduce latency and network utilization. However, as discussed earlier, when workload assumptions are incorrect, PC locks overpublish (using more network bandwidth than AFS or SC). Overpublishing occurs when the server pushes data that clients don't read. The update message has network costs that are never recovered by avoiding future data requests.

To simulate overpublishing, we varied the rate at which client read requests were serviced, focusing on read rates slower than the rate at which a file was written and published. We ran the simulation for a file system with 100 reading clients. Figure 6 shows that when reads occurred less frequently than writes, PC locking more heavily utilized the network resource. SC and AFS locking's network usage grew proportionately with the increasing read rate, whereas PC locking's network usage was about the same for all read rates.

This finding matched our intuition. For a given write rate and a fixed number of clients, the amount of network bandwidth should be the same. This effect is noticeable for the substantial update size of the Web camera workload. Alternatively, the stock ticker workload did not exhibit significant overpublishing, because data were small and publishing incurred minimal network costs.

PC locking does not perform well on all workloads, particularly for workloads dominated by write. Exploring overpublishing helps us to understand PC locking's properties, but overpublishing has little effect on our system, which targets the read-dominated, Web-serving workload.

Conclusions

Our next step is to validate our simulation results against an implementation of this protocol. Publish consistency and producer-consumer locking are key technologies for scalable Web hosting. We are deploying these technologies in the Storage Tank file system project at IBM research (http://www.almaden.ibm.com/cs/storage.html).

Publish consistency is only one of many consistency options we are implementing. We also provide protocols specific to the needs of transaction processing, data mining, and write-sharing workloads.

Consistency is only one aspect of customizing a file system. We are exploring other elements of file system architectures for application-specific semantics and workloads. In particular, requirements vary for replication, data placement, and fault tolerance. Considering all of these elements jointly will allow file systems to provide a convenient and efficient infrastructure for emerging and future applications in addition to Web serving.

References

1. D. Li, P. Cao, and M. Dahlin, "WCIP: Web Cache Invalidation Protocol," tech. report draft-danli-wrec-wcip-01.txt, Internet Draft, Mar. 2001.
2. L. Lamport, "How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs," *IEEE Trans. Computers*, vol. C-28, no. 9, 1979.

3. H. Yu and A. Vahdat, "Combining Generality and Practicality in a Conit-Based Continuous Consistency Model for Wide-Area Replication," *Proc. Int'l Conf. Distributed Computing Systems*, (ICDCS 21), IEEE Computer Soc. Press, Los Alamitos, Calif., 2001.
4. J.S. Gwertzman and M. Seltzer, "The Case for Geographical Push-Caching," *Proc. Fifth Workshop Hot Topics in Operating Systems*, IEEE Computer Soc. Press, Los Alamitos, Calif., May 1995.
5. R. Burns, R. Rees, and D. Long, "An Analytical Study of Opportunistic Lease Renewal," *Proc. Int'l Conf. Distributed Computing Systems*, (ICDCS 21), IEEE Computer Soc. Press, Los Alamitos, Calif., Apr. 2001.
6. A. Singla, U. Ramachandran, and J. Hodgins, "Temporal Notions of Synchronization and Consistency in Beehive," *Proc. 9th ACM Symp. Parallel Algorithms and Architectures*, ACM Press, New York, 1997.
7. J. Yin et al., "Volume Leases for Consistency in Large-Scale Systems," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 4, July/Aug. 1999.
8. M.L. Kazar, "Synchronization and Caching Issues in the Andrew File System," *Proc. Usenix Winter Tech. Conf.*, Usenix Assoc., Berkeley, Calif., Feb. 1988.
9. M. Rosenblum and J.K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Trans. Computer Systems*, vol. 10, no. 1, Feb. 1992.
10. M.L. Kazar et al., "DEcorum File System Architectural Overview," *Proc. Summer Usenix Conf.*, Usenix Assoc., Berkeley, Calif., June 1990.
11. M. Crovella and L. Lipsky, "Long-lasting Transient Conditions in Simulations with Heavy Tailed Workloads," *Proc. Winter Simulation Conf.*, Soc. for Computer Simulation, San Diego, Calif., 1997.
12. J.S. Gwertzman and M. Seltzer, "World-Wide Web Cache Consistency," *Proc. Usenix Ann. Tech. Conf.*, Usenix Assoc., Berkeley, Calif., Jan. 1996.
13. P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World Wide Web," *IEEE Trans. Computers*, vol. 47, no. 4, Apr. 1998.

Randal C. Burns is a research staff member at IBM's Almaden Research Center in San Jose, California. He earned PhD and MS degrees in computer science from the University of California, Santa Cruz, and a BS in geophysics from Stanford University. His research interests include distributed data management, storage systems, concurrency control, data placement, and I/O scheduling. He is a member of the IEEE and the ACM.

Robert M. Rees is a senior member of the technical staff at IBM's Almaden Research Center in San Jose, California. He was the chief architect for the Adstar Distributed Storage Manager, now known as the Tivoli Storage Manager, and is currently the leader and chief architect for the Storage Tank project. He earned a BS in computer science from the University of California, Santa Cruz, in 1995.

Darrell E. Long is a professor of computer science, and director of the Storage Systems Research Center at the Jack Baskin School of Engineering, at the University of California, Santa Cruz. He is also a consultant to IBM Research. He earned MS and PhD degrees in computer science and engineering at the University of California, San Diego, and a BS in computer science from San Diego State University. He has authored more than 80 research publications and holds four patents. He is a member of the ACM, a senior member of the IEEE, and the Chair of the Scholars for the Usenix Association. His research interests include storage systems, reliability, mobile computing, and multimedia distribution.

Readers can contact Randal Burns at K56/B3 IBM Almaden Research, 650 Harry Road, San Jose, CA 95120, randal@almaden.ibm.com.