Hierarchical Cache Consistency in a WAN
Extended Abstract

Jian Yin, Lorenzo Alvisi, Mike Dahlin, Calvin Lin
Department of Computer Sciences
University of Texas at Austin

Abstract

This paper explores ways to provide strong consistency for Internet applications scaling to millions of clients. We make four contributions. First, we identify the ways in which specific characteristics of data-access workloads affect the scalability of cache consistency algorithms. Second, we define two primitive mechanisms, *split* and *join* for growing and shrinking hierarchies. We show how these primitives can be implemented with a simple mechanism already present in a protocol for strong consistency that we have previously proposed. Third, we describe and evaluate policies for using split and join to address the fault tolerance and performance challenges of hierarchies. Finally, we compare various algorithms for maintaining strong consistency in a range of hierarchy configurations. We evaluate our algorithms using simulations.

1 Introduction

To prevent the rapid growth of Web traffic from degrading Internet performance, caching has become a ubiquitous Internet technology. However, web caching introduces the problem of maintaining consistency. With weak notions of consistency, innovative web services and new classes of program-driven applications—such as agents, robots, and distributed databases—will likely produce incorrect results. Thus, strong consistency—which guarantees that a client's read of an object returns the latest completed write of that object—will become increasingly desirable. Even in human-driven browsers, consistency polling can increase latency and reduce the effectiveness of large scale caches [3]. Thus, strong consistency can also be advantageous for current applications.

Strong consistency can be provided using callbacks, but simple callback algorithms are unacceptable for a WAN because servers may be forced to delay their writes indefinitely when there are client failures or network partitions. In previous work, we have shown how to combine strong consistency with timely server writes by using the notion of *Volume Leases* [12], a generalization of *leases*, which were first introduced for file systems [5]. Our results used trace-driven simulations to show that Volume Leases perform well.

The key questions that this paper answers are (1) how large a system can Volume Leases accommodate, and (2) what techniques can be used to scale to even larger systems. To answer these questions, this paper explores ways to combine volume leases with hierarchies for systems with millions of clients.

Adding hierarchy to a server-driven cache consistency scheme can yield three benefits. First, latency can be reduced if clients can register callbacks or renew leases by going to a nearby node in the consistency hierarchy rather than to the server. Second, it can improve network efficiency by forming a multicast tree and a reduction tree for sending invalidation messages to caches and gathering their replies. Third, it improves server scalability by distributing load and callback state across a collection of nodes.

However, using hierarchies for scalable consistency introduces its own challenges. Availability may suffer because hierarchical structures consist of multiple nodes that can fail independently. Also, latency can increase if the hierarchy must be traversed to satisfy requests. Finally, it unclear how the hierarchy should be configured.

This paper develops solutions for hierarchical consistency and addresses the three issues mentioned above. We make four contributions. First, we identify the ways in which specific characteristics of data-access workloads affect the scalability of cache consistency algorithms. Second, we define two primitive mechanisms, *split* and *join* for growing and shrinking hierarchies, and we show how these primitives can be implemented with a simple mechanism already present in the Volume Lease algorithms. Third, we describe and evaluate policies for using split and join to address the fault tolerance and performance challenges of hierarchies. Fourth, we examine and compare various algorithms for maintaining strong consistency in a range of hierarchy configurations.

We evaluate our algorithms using simulation. To study scalability and to evaluate how it is affected by different workload characteristics, we first use a series of synthetic workloads. To calibrate these results with realistic workloads, we also examine some smaller trace-based workloads. Overall, we find that even without hierarchies, volume leases can scale to large services with tens of thousands or hundreds of thousands of clients; with hierarchies, scalability beyond millions of clients appears feasible.

The thesis of this paper is not that all servers should provide strong consistency, but rather, that for Internet-scale systems, strong consistency is feasible for a wide range of applications. The rest of this paper proceeds as follows. Section 2 discuss a few ideas that are needed to understand this work. Section 3 describes our new algorithm whose performance we evaluate in Section 4. We close by discussing related work and drawing conclusions.

2 Background

This section describes four concepts necessary to understand hierarchical consistency: callbacks, leases, the Volume Leases algorithm, and reconnection under Volume Leases.

In server-driven consistency, a client registers *callbacks* with a server for objects that it caches [7, 9]. Before modifying an object, a server first sends invalidation messages to all clients that have registered interest in that object. The advantage of this approach is that servers have enough information to know exactly when cache objects must be invalidated. In contrast, in client-driven consistency schemes, such as those currently used in NFS and HTTP, clients periodically ask the server if objects have been modified. This creates a dilemma for the client. A short polling period increases both server load and client latency, while a long polling period increases the risk of reading stale cache data.

There are two challenges for server-driven consistency in large distributed systems. First, scalability is an issue, as large numbers of clients lead to large server state and large bursts of load when popular objects are modified. Second, performance in the face of failures is an issue. Servers cannot modify an object until clients have been notified that their cached object is no longer valid, so server writes can be delayed indefinitely while the server waits for acknowledgments from unreachable clients.

These issues can be addressed by introducing the notion of *leases* [5]. When a client registers a lease with a server, the lease specifies some time T during which the server will notify clients of updates. This improves scalability because servers only need to track active clients, and it improves fault tolerance because even if a client is unreachable, writes are only delayed until the client's lease expires. The lease length T represents a trade-off. Longer leases minimize the overhead of renewing leases, while short leases reduce server state and improve failure-mode write performance.

Leases do not perform well for web workloads because the interval between a client's reads is typically long, so object leases must be long to amortize the cost of lease renewals across many reads. The Volume Leases algorithm introduces the notion of a volume, which is a collection of objects that reside on the same server, and associates a lease with each volume. A client's cached object is valid only if both its object lease and corresponding volume lease are valid. The Volume Leases algorithm uses a combination of long object leases and short volume leases to break the tradeoff with lease lengths. Short volume leases allow servers to write quickly in the face of client and network failures: since clients can't read an object when its corresponding volume lease is invalid, in the worst case the server waits only for the the short volume lease to expire before modifying an object. The long object lease minimizes the overhead of renewing object leases, while the cost of renewing volume leases is amortized across the number of objects that reside in the same volume.

In the Volume Leases algorithm, a server maintains a list of *unreachable* clients whose volume leases expired while the server was attempting to invalidate an object lease. When a client on the unreachable list recovers and tries to renew its volume lease, the algorithm uses a reconnection protocol to restore consistency between the client's and server's lists of current object leases.

Because the reconnection protocol is a key building block for hierarchical caching, we describe it in detail. Each server maintains an epoch number. Whenever a server recovers from a crash, the epoch number is incremented and logged to a stable storage device before the server proceeds with normal operations. All messages from the server to the clients include the epoch number. When a client receives a message, the epoch number is recorded by the client and associated with this connection. The epoch number is also included when a client sends a volume lease request to a server. Upon receiving volume lease requests, the server grants a volume lease only if the epoch number in the message matches its own and if the client hasn't been moved to the unreachable list. Otherwise, the server sends the client a reconnect request. In response to a reconnect request, a client sends to the server the list of objects it currently caches and the version numbers of these objects. The server then compares the version numbers of the cached objects and the objects in the server. Object leases are granted to all objects whose versions match. The server invalidates all other cached objects. The volume lease is then granted. Note that all these tasks can be accomplished with one message from the server to the clients. When the client finishes updating its object leases, it sends a connect message back to the server, which then removes the client from the unreachable list.

3 Algorithms

We first describe the basic static hierarchy algorithm and discuss its performance and fault tolerance properties. We then present two primitive mechanisms, split and join, for reconfiguring the hierarchy. These mechanisms can be constructed with trivial additions to the basic Volume Leases algorithm. We then describe policies that use these mechanisms to enhance the fault tolerance and performance of the basic static hierarchy.

Both the static and dynamic versions of the algorithm assume that nodes participating in the consistency service have been identified and organized into an initial hierarchy. This study does not specify a particular mechanism for doing so.

For some systems, manually constructing the hierarchy will suffice; for some, such as the server-proxy-client configuration that we address in Section 4.3, automatic construction is trivial; and, for others, more sophisticated automatic strategies such as those described by Plaxton et. al [10] may be required. This hierarchy may be embedded on current clients and proxies, it may be coincident with a larger cache hierarchy [1] or it may be part of a separate data-location-metadata hierarchy [4, 11].

3.1 Static hierarchy

Our consistency hierarchy is a tree structure of interconnected nodes. We refer to the root as the origin server, to the leaves as clients, and to the intermediate nodes as consistency servers. Each node runs the standard Volume Leases algorithm, and each node acts both as a client and as a server, treating its parent as its server and its children as its clients. Each node thus satisfies lease requests from its children by returning a valid lease if it has one cached or—if it does not—by requesting a lease from its parent, caching the lease, and returning the lease to its child. Similarly, each node passes to its children with valid leases the invalidation messages it receives from its parent.

Such hierarchies have the potential to improve performance by reducing both server load and by the latency of client lease renewals. In the Internet, a popular site might serve millions of clients, and by using a hierarchy, a server only communicates with and tracks its immediate children. This reduces memory state, average load for lease renewals, and bursts of load when popular objects are modified. Essentially, the consistency hierarchy forms a multicast tree for sending invalidation messages and a reduction tree for gathering replies. By the same token, if clients can renew leases by going to nearby intermediate consistency servers rather than to the root server, read latency and network load may be reduced.

However, the use of leases in the hierarchy is not guaranteed to reduce either server load or latency. When volumes are popular and frequently accessed, it is likely that consistency servers will hold valid leases and will respond to client requests without consulting their parents, and it is likely that the hierarchical "multicast" will achieve a large fan-out and significantly reduce server load. However, for unpopular or infrequently accessed volumes, the time between accesses to consistency nodes is likely to be longer than the volume lease, so the cached leases may often have expired when they are accessed. In these cases, many messages would traverse the entire hierarchy, increasing the average read latency without reducing server load.

A second problem with a static hierarchy is reliability. The hierarchy consists of a large number of nodes that can fail independently, and one node failure can effectively disconnect a subtree.

3.2 Join and split

The solution to both problems is to reconfigure the consistency hierarchy dynamically without breaking the guarantee of strong consistency. We propose a mechanism that uses two primitives: *join*, which removes an intermediate node from the hierarchy, and *split*, which adds an intermediate node to the hierarchy. Both primitives work on a per-volume basis—in our system different volumes can use different hierarchies.

Join and split can be trivially implemented using a mechanism already required by the Volume Leases algorithm. Recall that join removes a node from the hierarchy, connecting the children of the node directly to the node's parent. To implement join we augment the volume epoch number to include the ID of the parent node. When a child decides to initiate a join for a particular volume, it simply begins using its former grandparent as a parent. The old volume epoch number held by the child will not match its new parent, so the new parent will initiate the standard volume reconnection protocol to synchronize its state with its new child. Thus, going to a new parent in the hierarchical algorithm is no different than going to a server that has crashed and lost a client's state in the original Volume Leases algorithm. Similarly, to split the hierarchy, a child chooses a descendant of its parent and starts using the new node as its parent, again using the reconnection protocol to synchronize the state. Note that for both split and join, the decision to use a new parent is made by children. Such decisions are a matter of policy. Children can thus decide to find new parents to improve fault tolerance or they can be told to use new parents to improve performance.

3.3 Fault tolerant static hierarchy

Using join and split, an intermediate node failure can be handled as follows. If a node N cannot contact its parent P to renew a lease, it sends the renewal message to one of its ancestors A, triggering the volume reconnection protocol between N and A. Note that if A cannot send an invalidation to P, it does not try to contact N, but instead waits for the volume lease timeout; this means that parents only need to know about their immediate children, not their more distant descendents. Finally, when node P recovers, it can send hints to its list of (former) children suggesting that they split from A and join P instead.

3.4 Dynamic hierarchy configuration

For volumes where read frequency is high and there are many active clients, a deep hierarchy can reduce read latency and distribute load. However, for less popular objects, or for popular objects with low read frequency, intermediate hops can increase read latency without significantly reducing server load. Therefore, it is useful for different volumes to construct different dynamic hierarchies. These hierarchies are constructed out of the static hierarchy using the split and join mechanisms in response to changing workloads. Hence, a node can have different children in the static and dynamic hierarchies: we refer to the former as static children, and to the latter as simply children.

In the dynamic configuration algorithm a node monitors the number of lease requests it receives from its children and the fraction of these requests that it can satisfy locally during time intervals of length T. Using this data, it instructs its children to join with its parent if (1) the load from its children would not cause the load on its parent to exceed a threshold value, and (2) its children would receive better read latency by skipping the node and going directly to the parent. A node N performs the latency calculation as follows.

Let C1 be the cost for a child of N to renew a lease cached at N, and let C2 be the cost for N to renew a lease cached at its parent. If the fraction of renewals that N satisfies locally is F, then the expected latency that a child of N pays to renew a lease is C1 + (1-F) C2. Assuming that the cost of accessing N's parent is about the same for both N and N's child, the expected cost after a join is C2. When C1 + (1-F) C2 is greater than C2 by some threshold, N instructs its children to perform a join unless doing so would raise the load of the parent to an unacceptable level.

Similarly, to determine when to initiate a split, a node monitors the requests from its children, and simulates the hit statistics for any skipped static child. When these statistics show that the expected read latency for a group of children would decrease by connecting to a skipped static child, the node instructs that group of children to perform a split. Simulating hit statistics is easy, because for each message it suffices to check the simulated lease and increment the number of hits or misses. A node may also initiate a split if its load exceeds some threshold.

4 Evaluation

Our evaluation of the hierarchical consistency consists of three parts corresponding to different deployment configurations and workloads. First, we examine an aggressive deployment model to characterize the factors that affect the behavior of the core algorithms and to determine the performance limits of our approach. Second, we examine a simple clustered-server configuration in which the hierarchy is used to distribute the algorithm across a LAN cluster to improve scalability but not latency. Third, we examine a configuration embedded on the server-proxy-client infrastructure that is common today.

Our methodology is to evaluate these algorithms using simulations. To stress scalability and to evaluate how different aspects of workloads impact scalability, we first use a series of synthetic workloads. Then, to calibrate these results, we also examine a smaller, trace-based workload in the context of the server-proxy-client configuration.

Based on these experiments, we reach the following primary conclusions:

- For the aggressive deployment scenario with flexible hierarchy configurations, static hierarchies can reduce latency compared to the flat Volume Lease algorithm for high request-rate services, but they can increase latency for low request-rate services. In contrast, the dynamic version always performs as well as the flat algorithm for low request rates and as well as the static hierarchy for high request rates.

- For workloads with modest request rates in the range of many current web services, the flat Volume Leases algorithm with a single server can scale to client populations in the tens or hundreds of thousands of nodes; distributing the consistency algorithm across a group of nodes—either in a cluster or across a WAN—via hierarchies can provide scalability to millions of clients even under very aggressive workloads.

- In the server-proxy-client configuration, which models a simple deployment path given current infrastructures, the simple static hierarchy performs well for our web trace workload; this configuration has the added benefit that it might also provide a controlled way to traverse firewalls to deliver consistency signals. The synthetic workload suggests that there may be other workloads for which the dynamic algorithm's flexibility is desirable.

Our methodology makes several significant assumptions and simplifications. For our latency estimates, we do not simulate network or server contention. We use a simple network topology model (described in more detail below) to make our analysis tractable. Our synthetic workloads simulate one object per volume, which may understate the apparent benefit of hierarchies because long-lived object leases are much easier to cache in the hierarchy than short volume leases; the small number of object per volume may also hurt the relative performance of the static algorithm.

Due to space constraints for the extended abstract, we abbreviate our results section as follows: we do provide details about our system configuration to put the results in perspective, but we present the main results in bullet form.

4.1 Generic hierarchy

Our Generic Hierarchy configuration represents a system with relatively few constraints on deployment. We examine this configuration to understand the basic behavior of the core algorithms as we vary several key parameters. This configuration also models an aggressive deployment strategy such as might be employed within a large cache service or in a system where collections of servers and cache systems coordinate to provide consistency.

The consistency hierarchy is a tree with one server at its root, C clients at its leaves, and $l-1$ levels of intermediate nodes. For simplicity, we assume that the degree d at all levels of a tree are the same, with $d^l = C$. We use a simple cost model for accessing consistency servers. Lease renewal latencies between any internal or leaf node in a subtree and the root of that subtree are equal and increase with the number of subtree's leaves as follows: subtrees with 100 or fewer leaves have a latency of 30 ms, subtrees with 10,000 or fewer leaves have a cost of 100 ms, and subtrees with more than 100,000 leaves have a cost of 400 ms; costs for subtrees of other sizes are estimated through interpolation. These costs are meant to be suggestive of department-, enterprise-, and Internet-scale delays, but do not represent any specific system.

We use a synthetic workload and examine the average read latency and server load. We simulate the accesses of a collection of clients to a single volume. Out of N_{total} clients, we choose a subset of size N_{active} clients that access the volume with per-client inter-access times determined using an exponential distribution around an average value t_{read} expressed as a ratio of the average inter-access time to the volume lease renewal time.

Figure 1 shows the average lease renewal latency as per-client read frequency is varied, and Figure 2 shows lease renewal latency as the fraction of clients that access the volume in question is varied. Both sets of graphs have the same general shape because both increase the total request rate as they move right, but they represent different dimensions of the design space.

To interpret these graphs it is helpful to consider where different classes of services might lie or where a single service might lie under different workloads. For example, a weather service with a 10-second lease period and for which an average client that uses the service visits once per day for a minute would correspond to a read frequency of less than 0.001 reads per volume lease period per client. Similarly, a news service whose typical users visit for 5 minutes during the 8-hour working day would correspond to a volume renewal frequency near 0.01 per volume lease period per client. That same service's read frequency might jump above 0.1 or even near 1 for periods of time during news events of widespread interest (e.g., a Ford Bronco chase) as clients constantly monitor the news for new developments. Similarly, emerging program-driven applications might span a wide range of the parameter space.

With respect to lease renewal latency in the Generic Hierarchy, the main observations are as follows:

- Hierarchies can significantly reduce latency for active and popular services.

- The dynamic hierarchy succeeds in matching the latency of the flat Volume Leases algorithm for less active or less popular services while matching the performance of the static hierarchy for busier services. Relative to flat Volume Leases, the static hierarchy can hurt latency for less active or less popular services but can help latency for active and popular services.

- The dynamic hierarchy appears to be a good default choice for this configuration. If a service's access patterns are known precisely and if these access patterns do not change much, then either the flat Volume Leases or static hierarchy may be reasonable.

Finally, note that the variations among different depths of underlying static trees for each graph depend on interactions between the number of clients under each level of a subtree and our assumptions on the network distances between subtrees as a function of subtree size. So this experiment should not be used for general comparisons between the number of levels that should be used in the underlying hierarchy.

Figure 3 shows similar experiments but with 100,000 total clients (20,000 of them active) rather than 1,000,000. By comparing these results to those with more clients, we gain intuition about the effects of scaling the client population that may help predict system behavior for populations larger than the 1,000,000 that we are able to simulate.

- As expected, increasing (decreasing) the total number of clients decreases (increases) the per-client request rate for which hierarchies begin to pay off relative to the flat Volume Leases configuration. We also varied the number of active clients from the population (graph omitted to save space) and found similar results.

Figure 4 shows how server load varies with client request rate hierarchies spanning one million clients. (Results for varying the number of active clients or simulating a universe of 100,000 clients are omitted, but are qualitatively similar).

- The flat Volume Leases algorithm scales to hundreds of thousands of clients under workloads corresponding to a range of reasonable web access patterns.

- The addition of hierarchies supports scalability to many millions of clients under nearly arbitrary workloads because it bounds the rate of requests at the root to one request per volume lease period per immediate child of the root.

4.2 Server cluster

Due to space constraints, we omit details of our server cluster experiment. The main idea is that the hierarchical consistency mechanisms can be used not only to distribute consistency algorithms across a WAN, but also to split a consistency service across a clustered web server. Although an algorithm built from the ground up for splitting consistency state and load across a cluster might marginally outperform our more general mechanisms, such an algorithm would have to solve the same basic problems of fault tolerance, distributing invalidations, gathering acknowledgments, and partitioning state that our algorithm handles, so the simplicity of using a single framework for both LAN and WAN distribution appears attractive.

Figure 5 shows the load on the server in the server cluster hierarchy where the server and all of the internal nodes of the consistency hierarchy are located in a tightly-coupled cluster, and the lowest internal nodes in the hierarchy communicate across a WAN with the clients. We do not show results for latency because this configuration is not designed to improve latency, just load-scalability; the latency measurements do not vary significantly across different configurations. Similarly, the dynamic hierarchy does not have any significant advantage over the static one.

- For the server-cluster configuration, the static hierarchy (with split and join for fault tolerance) provides a simple mechanism to scale the flat Volume Leases algorithm by distributing it across a group of nodes in a cluster; dynamic configuration to minimize latency is not required.

4.3 Server, proxy, client

Figures 6 and 7 show the latency and load measurements when the hierarchy algorithms are run on the server-proxy-client underlying hierarchy. Figure 8 shows latency for several selected volumes under a trace workload. The trace workload is the DEC trace [2], and we configure the system to have all clients under a single proxy and have each volume represented by a single server that communicates directly with proxies. The 8 servers are the 4 most popular ones, and 4 of medium popularity.

- As illustrated by the synthetic workload, as was true for the Generic Hierarchy, the dynamic hierarchy may be needed to accommodate the full range of services.
- The trace workloads include multiple objects per volume, and long object leases are easier to cache in a hierarchy. As a result, the static hierarchy begins to pay dividends even with relatively low access rates.
- For many current web workloads, the simple static hierarchy using the simple server-proxy-client hierarchy may be a reasonable deployment option. This configuration might also provide a controlled way to traverse firewalls to deliver consistency signals.

5 Related work

This section abbreviated for extended abstract.

Our study builds on efforts to assess the cost of strong consistency in wide area networks. Gwertzman and Seltzer [6] compare cache consistency approaches through simulation and conclude that protocols that provide weak consistency are the most suitable to a Web-like environment. In particular, they find that an adaptive version of polling exerts a lower server load than an invalidation protocol if the polling algorithm is allowed to return stale data 4% of the time. We arrive at different conclusions. In particular, much of the apparent advantage of weak consistency over strong consistency in terms of network traffic comes from clients reading stale data [8]. Other studies have noted significant numbers of consistency-related polling “misses” to unmodified and cached objects under current client-polling approaches [3].

We also build on the work of Liu and Cao [8], who use a prototype server invalidation system to evaluate the overhead of maintaining consistency at the servers compared to client polling.

6 Conclusions

In this paper we have shown that the Volume Leases algorithm can provide strong consistency for Internet services with tens of thousands of clients. We have also shown how the Volume Leases can be applied to hierarchical caches to perform well for workloads with millions of clients. The key mechanisms, join and split, can be implemented using a trivial extension of the Volume Leases algorithm. Finally, we have evaluated a number of hierarchy configurations, and our results show that a dynamically configurable hierarchy provides almost arbitrary amounts of scalability.

References

[1] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A Hierarchical Internet Object Cache. In *Proceedings of the 1996 USENIX Technical Conference*, January 1996.

[2] Digital Equipment Corporation. Digital’s Web Proxy Traces. ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html, September 1996.

[3] B. Duska, D. Marwood, and M. Feeley. The Measured Access Characteristics of World-Wide-Web Client Proxy Caches. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[4] S. Gadde, J. Chase, and M. Rabinovich. Directory Structures for Scalable Internet Caches. Technical Report CS-1997-18, Duke University Department of Computer Science, November 1997.

[5] C. Gray and D. Cheriton. Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 202–210, 1989.

[6] J. Gwertzman and M. Seltzer. World-Wide Web Cache Consistency. In *Proceedings of the 1996 USENIX Technical Conference*, January 1996.

[7] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

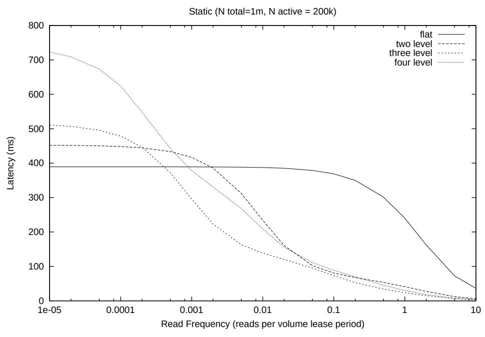
[8] C. Liu and P. Cao. Maintaining Strong Cache Consistency in the World-Wide Web. In *Proceedings of the Seventeenth International Conference on Distributed Computing Systems*, May 1997.

[9] M. Nelson, B. Welch, and J. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems*, 6(1), February 1988.

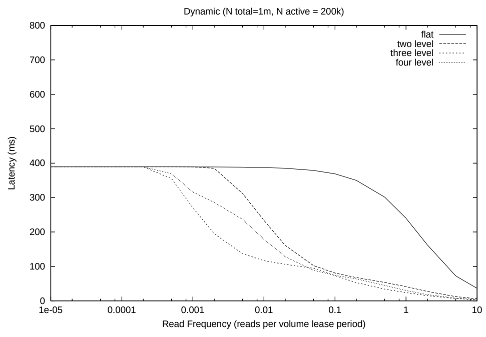
[10] C. Plaxton, R. Rajaram, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, June 1997.

[11] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design Considerations for Distributed Caching on the Internet. In *Proceedings of the Nineteenth International Conference on Distributed Computing Systems*, May 1999.

[12] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Volume Leases to Support Consistency in Large-Scale Systems. *IEEE Transactions on Knowledge and Data Engineering*, February 1999.

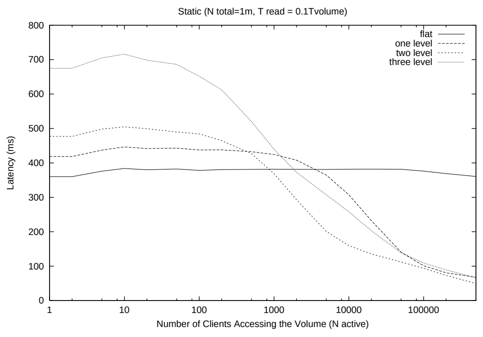


(a) Static

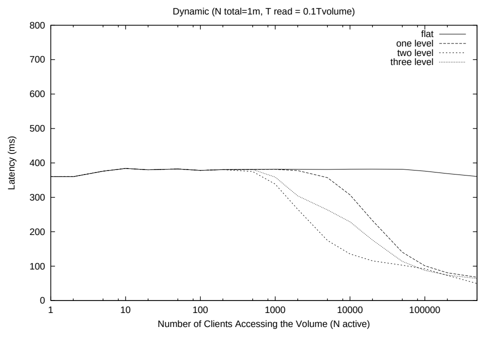


(b) Dynamic

Figure 1: Average read latency as the per-client read frequency is varied for a hierarchy of one million clients, of which 200,000 access the volume in question. (a) Shows performance of the static hierarchy and (b) shows performance of the dynamic hierarchy. For each figure, the lines show performance for the algorithms running on static trees of different maximum depths. The falling average latency for very high request rates under the flat hierarchy is due to volume lease renewal hits at the clients, themselves.

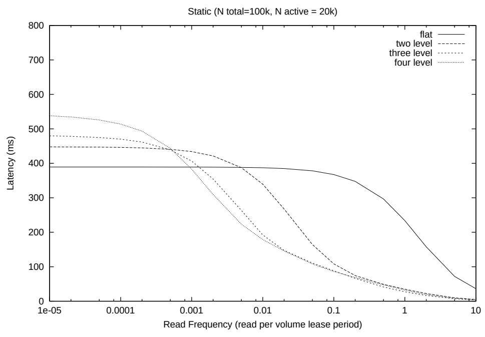


(a) Static

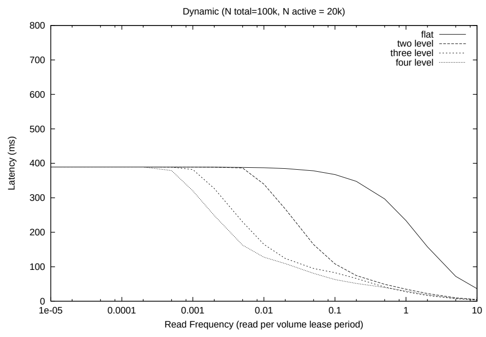


(b) Dynamic

Figure 2: Average read latency as the number of active clients varies for a hierarchy of one million clients, each issuing requests to a volume at a rate of 0.1 requests per volume lease period.

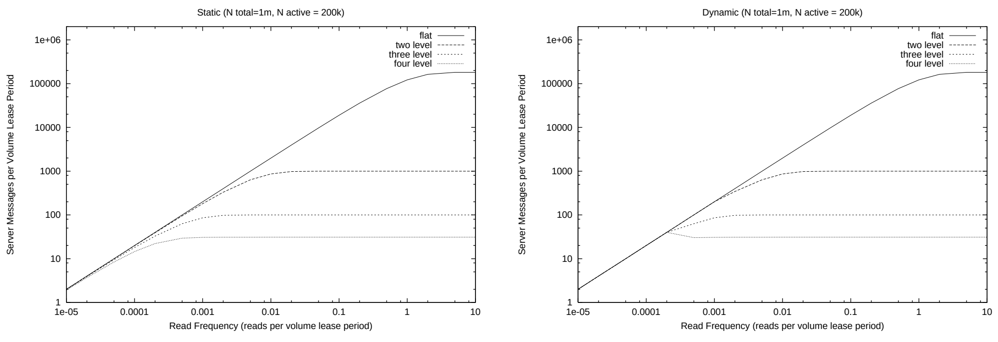


(a) Static



(b) Dynamic

Figure 3: Average read latency as the per-client read frequency is varied for a hierarchy of 100,000, of which 20,000 access the volume in question.



(a) Static (b) Dynamic

Figure 4: Average server load for handling renewal requests as the per-client read frequency is varied for a hierarchy of one million clients, of which 200,000 access the volume in question.

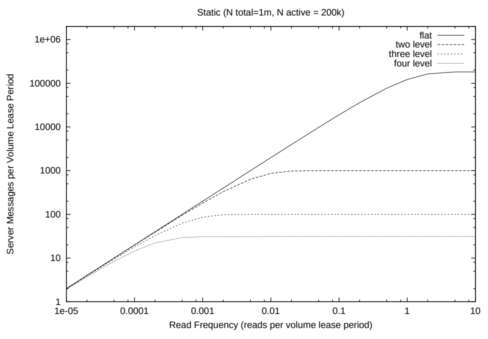
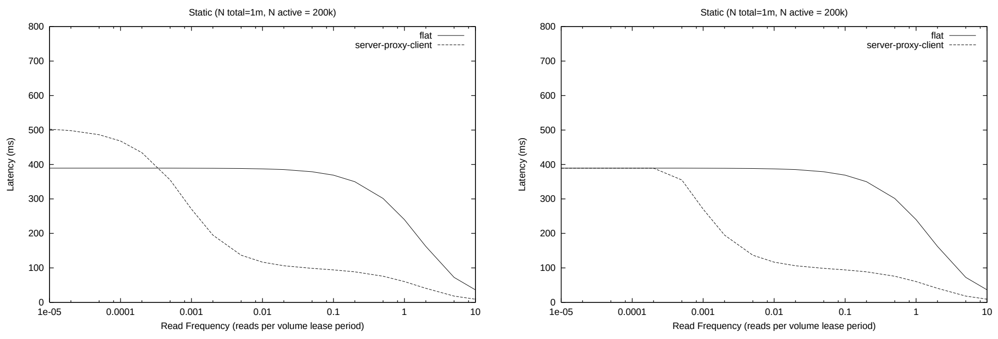
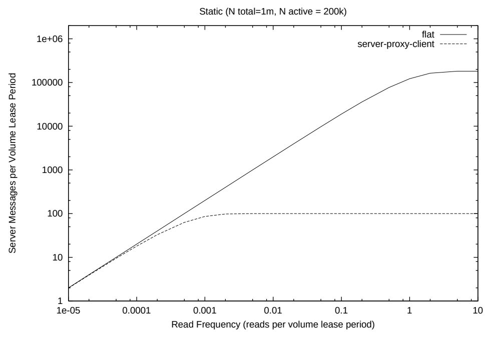


Figure 5: Server lease renewal load as the per-client read frequency is varied for a static server cluster hierarchy serving one million clients, of which 200,000 access the volume in question.

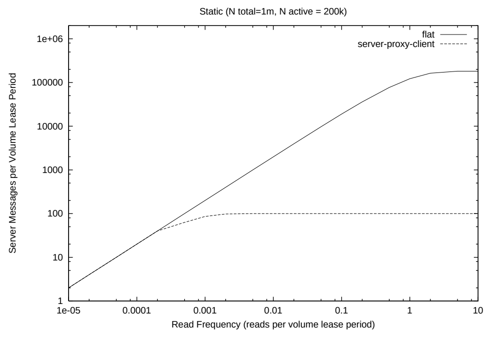


(a) Static (b) Dynamic

Figure 6: Average read latency as the per-client read frequency is varied for a server-proxy-client hierarchy of one million clients, of which 200,000 access the volume in question. In the server-proxy-client hierarchy the internal nodes in the consistency hierarchy are all proxies serving 10,000 clients each.



(a) Static



(b) Dynamic

Figure 7: Server lease renewal load as the per-client read frequency is varied for a server-proxy-client hierarchy serving one million clients, of which 200,000 access the volume in question.

	Med 1			Med 2			Med 3			Med 4		
	flat	static	dyn	flat	static	dyn	flat	static	dyn	flat	static	dyn
Latency (ms)	160.5	129.4	135.4	99.0	89.5	92.1	55.6	61.2	57.3	276.3	297.0	279.7
Load (server msgs/read)	0.41	0.23	0.27	0.25	0.16	0.20	0.14	0.12	0.14	0.69	0.57	0.64

(a) Trace results for four medium-loaded volumes.

	Large 1			Large 2			Large 3			Large 4		
	flat	static	dyn	flat	static	dyn	flat	static	dyn	flat	static	dyn
Latency (ms)	84.1	30.8	30.7	123.2	51.1	51.2	133.0	46.7	46.7	68.9	39.3	39.6
Load (server msgs/read)	0.21	0.03	0.03	0.31	0.05	0.05	0.33	0.03	0.03	0.18	0.06	0.07

(b) Trace results for four heavily-loaded volumes.

Figure 8: Average read latency and fraction of renewal requests sent to the server for the four medium-loaded and four heavily-loaded volumes from the DEC trace workload under a server/proxy/client hierarchy in which the internal node in the consistency hierarchy is the proxy serving the DEC clients.