Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency

Cary G. Gray and David R. Cheriton
Computer Science Department
Stanford University

Abstract

Caching introduces the overhead and complexity of ensuring consistency, reducing some of its performance benefits. In a distributed system, caching must deal with the additional complications of communication and host failures.

Leases are proposed as a time-based mechanism that provides efficient consistent access to cached data in distributed systems. Non-Byzantine failures affect performance, not correctness, with their effect minimized by short leases. An analytic model and an evaluation for file access in the V system show that leases of short duration provide good performance. The impact of leases on performance grows more significant in systems of larger scale and higher processor performance.

1 Introduction

Caching introduces the problem of ensuring consistency between the cached data and its primary location of storage. By *consistent*, we mean that the behavior is equivalent to there being only a single (uncached) copy of the data except for the performance benefit of the cache. With large caches, the traffic required to maintain consistency can be the dominant factor in cache performance.

Cache consistency protocols have been extensively studied in the work on shared memory multiprocessor architectures; this work relies on reliable, synchronous broadcast communication as provided by the system bus. A distributed system, however, can experience partial failures: a host may crash or messages may be lost. Existing approaches to consistency for file caches fall into two categories: those that assume reliable broadcast, and so do not tolerate communication failures, and those that require a consistency check for every read, and so fail to deliver good performance.

In this paper, *leases* are proposed as a consistency protocol that handles host and communication failures using physical clocks. An analytic model and an evaluation using file access characteristics of the V system show that short-term leases provide near optimal efficiency for a large class of systems in spite of the fault-tolerance provisions. We argue that leases are of increased benefit in future distributed systems of larger scale with their larger ratio of processor speed to network delay and larger aggregate rate of failure.

The next section describes leases and how they are used to implement cache consistency. Section 3 derives our simple analytic model for picking lease terms and explores their application using data from the V distributed system [4]. Section 4 describes some optimizations in lease management. Section 5 examines the fault-tolerance of leasing. Section 6 compares leases with other work on distributed cache consistency and related problems. The concluding section summarizes our results and speculates on further applications of leases and directions for future research.

2 Leases and Cache Consistency

A *lease* is a contract that gives its holder specified rights over property for a limited period of time. In the context of caching, a lease grants to its holder control over writes to the covered datum during the *term* of the lease, such that the server must obtain the approval of the leaseholder before the datum may be written. When a leaseholder grants approval for a write, it invalidates its local copy of the da-

This work was supported in part by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, by the National Science Foundation Grant DCR-83-52048, and by Digital Equipment Corporation.

tum.

A cache using leases requires a valid lease on the datum (in addition to holding the datum) before it returns the datum in response to a read, or modifies the datum in response to a write. When a datum is fetched from the server (the primary storage site of the datum), the server also returns a lease guaranteeing that the data will not be written by any client during the lease *term* unless the server first obtains the approval of this leaseholder. If the datum is read again within the term of the lease (and the datum is still in the cache), the cache provides immediate access to the datum without communicating with the server. After the lease expires, a read of the datum requires that the cache first extend the lease on the datum, updating the cache if the datum has been modified since the lease expired. When a client writes a datum, the server must defer the request until each leaseholder has granted approval or the term of its lease has expired.

We limit ourselves here to write-through caches, for doing so simplifies the explanation; extending the mechanism to support non-write-through caches is straightforward. Write-through gives clean failure semantics: no write that has been made visible to any client can be lost; applications must otherwise be prepared to recover from lost writes. Though the cost of write-through for file caches is considered prohibitive by some [16], the cost can be largely eliminated by giving special handling to temporary files [9, 24], since they receive the majority of writes.

To illustrate the operation of a file cache using leases, consider a diskless workstation being used for document production. When the workstation executes `latex` for the first time, it obtains a lease on the binary file containing `latex` for a term of (say) 10 seconds. Another access to the same file 5 seconds later can use the cached version of this file without checking with the file server. An access to this file after the 10-second term has expired requires the cache to check with the server. When a new version of `latex` is installed, the write is delayed until every leaseholder has approved the write. If some host holding a lease for this file is unreachable, the delay continues until the lease expires.

In the preceding example, the relevant reads and writes are not limited to operations on the contents of the file. In order to support a repeated open, the cache must also hold the name-to-file binding and permission information, and it needs a lease over this information in order to use that information to perform the open. Similarly, modification of this information, such as renaming the file, would constitute a write.

Short lease terms have several advantages. One is that they minimize the delay resulting from client and server failures (and partitioning communication failures). When the server cannot communicate with a client, the server must delay writes to a file for which the failed client holds a lease until that lease expires.[1] When a server is recovering after crashing, it must honor the leases it granted before it crashed. This is most easily done if it remembers the maximum term for which it had granted a lease, and it delays writes to all files for that period, effectively increasing the time to fully recover by the maximum term. Alternately, the server can maintain a more detailed record of leases on persistent storage, but the additional I/O traffic is unlikely to be justified unless terms of leases are much longer than the time to recover.

Short leases also minimize the false write-sharing that occurs. *False sharing* refers here to a lease conflict when no actual conflict in file access exists. Specifically, false sharing occurs when a client writes to a file which is covered by a lease held by another client when the other client is not currently accessing the file. False sharing introduces the overhead of a callback to the leaseholder(s) (thereby delaying the requesting client and loading the leaseholder and server) in a situation where without leases there would be no conflict. In the extreme, a lease term should be set to zero if a client is not going to access the file before it is modified by another client.

Finally, short lease terms reduce the storage requirements at the server, since the record of expired leases could be reclaimed. However, the storage overhead for the server to keep track of the leases it has granted is modest. The server requires a record of each leaseholder's identity and a list of the leases it holds; each lease requires only a couple of pointers. For a client holding about one hundred leases, the total is around one kilobyte per client. Even if this were a problem, it could be reduced by recording leases at a larger granularity, so that each client holds few leases, at the expense of some increase in contention. We show later how the per-client record can be eliminated for the most common class of widely-shared files.

Longer-term leases are significantly more efficient both for the client and server on files that are accessed repeatedly and have relatively little write-sharing. This may be observed in the Andrew file system project [10], which went from using a lease term of zero in the prototype to effectively a lease term of infinity in the revised version.[2] The next section presents an analytic model of lease performance and determines appropriate lease terms using parameters based on data from the V distributed system.

3 Choosing the Lease Term

The choice of lease term is based on the trade-off between minimizing lease extension overhead versus minimizing

[1] To avoid starvation of writes, the server does not grant new leases on a file when a write is waiting for approval or for leases to expire.

[2] At the expense of failing to guarantee consistency after a communications failure.

Symbol	Description
N	number of clients (caches)
R	rate of reads for each client
W	rate of writes for each client
S	number of caches in which the file is shared
m_{prop}	propagation delay for a message
m_{proc}	time to process a message (send or receive)
ϵ	allowance for uncertainty in clocks
t_S	lease term (at server)

Table 1: Performance parameters

false sharing. This trade-off applies to minimizing both server load and client response. Lease space overhead is a less critical consideration and so is ignored as a factor. In addition, the rate of failures is assumed to be low enough to have no significant effect on the average response time, especially with short-term leases. Finally, we consider here only on-demand extension of leases rather than periodic extension or other options such as noted in Section 4.

3.1 A Simple Analytic Model

We consider a system consisting of a single server, characterized by the performance parameters given in Table 1. That is, the server has one file and N clients for that file, where each client's reads and writes follow Poisson distributions with rates R and W, respectively. The file is shared by S of the caches at each point it is written. There is at most one lease per client for the file.

We assume that the message processing time[3] m_{proc} at both the sender and recipient and the message propagation time m_{prop} are the same for all hosts. Thus, a message is received $m_{prop} + 2m_{proc}$ after it is sent and a unicast request and reply takes $2m_{prop} + 4m_{proc}$. Multicast messages are sent once, and received with high probability by the recipients using a multicast facility [5, 6]; it requires time $2m_{prop} + (n+3)m_{proc}$ to send a multicast message and receive n replies.[4].

For a lease with term t_S, the effective term t_C at the cache is

$$t_C = \max(0, t_S - (m_{prop} + 2m_{proc}) - \epsilon)$$

because t_C is shortened by the time for the cache to receive the lease $m_{prop} + 2m_{proc}$ plus the allowance ϵ for clock skew. Thus, in systems with large propagation delays between clients and large clock skew, the server must provide a proportionally larger lease term t_S if the lease term at the clients is to be effectively greater than zero.

If a cache handles an expected Rt_C reads over the term of the lease not counting the read that results in the lease request, the cost of the lease request is amortized over $1 + Rt_C$ reads, so that the rate of extension-related messages handled by the server is

$$\frac{2NR}{1 + Rt_C}$$

adding an average delay of

$$\frac{2(m_{prop} + 2m_{proc})}{1 + Rt_C}$$

to each read request.

When it receives a write, the server multicasts a request for approval to all of the leaseholders and processes the replies. Assuming the writer is one of the leaseholders, one approval message can be saved if the request for a write carries the implicit approval of the requesting cache.[5] Obtaining approval therefore requires one multicast request message plus $S - 1$ approvals, for a total of S messages.[6] The time t_a to gain approval is

$$t_a = 2m_{prop} + (S+2)m_{proc}$$

for $S > 1$. Thus, the delay is at most t_a and the load at most NSW.

In the file cache environment, we expect lease terms on the order of seconds and message times (including t_a) in the range of milliseconds. We therefore do not consider cases in which t_a is a significant fraction of t_S. The exception is $t_S = 0$: it is important to recognize that a zero lease term is better than a very short lease term because a non-zero t_S and zero t_C means that writes are penalized but reads do not benefit.

When the lease term is zero or there is no sharing, the load and delay are limited to those due to extensions of leases. For $S > 1$ and $t_S > 0$, though, the server sends and receives

$$\frac{2NR}{1 + Rt_C} + NSW \qquad (1)$$

consistency-related messages per unit time, and the average delay of

$$\frac{1}{R+W}\left(\frac{2R(m_{prop} + 2m_{proc})}{1 + Rt_C} + Wt_a\right) \qquad (2)$$

[3]The processing time does not include processing that occurs after the packet is sent or before it is received, only processing that is on the critical delay path. Queueing delays due to congestion are ignored, as is the second-order effect of response time on request rate.

[4]The average propagation and processing times m_{prop} and m_{proc} include a normal level of retransmissions, and so our estimates are reasonable for multicasts to small numbers of recipients. When the number of recipients (and replies) is large, the delay and processing overhead increase as more retransmissions may be required.

[5]This optimization is particularly important to allow the common case of an unshared file to be handled with a single unicast request-response from the client to the server. It means that a longer term always decreases the server load for unshared files.

[6]Without multicast, it would require $2(S-1)$ messages.

is added to each read or write.

The load for zero lease term is $2NR$; a term longer than t_a produces a lower load if

$$2NR > \frac{2NR}{1 + Rt_C} + NSW$$

Defining a *lease benefit factor* as

$$\alpha = \frac{2R}{SW},$$

the preceding condition holds if $\alpha > 1$ and

$$t_C > \frac{1}{R(\alpha - 1)}.$$

A sufficiently long lease term will reduce server load whenever α is greater than one. Larger values of α and R imply better performance for short terms.[7] Intuitively, α measures the ratio of reading to writing, scaled by the additional overhead caused by sharing.

In extending this analysis to handle multiple files, we note that the load due to multiple leases sums directly. The cache can batch its requests for extensions so that a single request covers many files. R and W then correspond to the total rates for all covered files, and so are higher; the higher absolute rate of reads increases α, and so the benefit is greater. In general, a cache should extend together all leases over all files that it still holds.

The server load due to consistency is roughly proportional to the number of messages handled (sent or received) by the server. If we know the fraction of server load due to consistency for a lease term of zero, we can calculate the (relative) total load from the (relative) consistency load. Similarly, application-level response time includes other processing in addition to the time required to ensure consistency.

The formulas we have derived can be used to predict performance in a specific system, given the appropriate parameters, as demonstrated in the next section.

3.2 Expected Performance In V

The expected performance of leases with the V file caching mechanism [9] is determined using the analytic model developed above and system performance parameters collected from measurement, given in Table 2. The measurements are determined from a trace of file access traffic generated by recompiling the V file server, with the file service and client programs executing on MicroVAX II workstations connected by Ethernet. The message times are based on separate timings of V inter-process communication. The trace includes only one client so there are no writes to shared files. We have calculated estimates for different degrees of sharing to illustrate its effect over a plausible range.

[7] When unicast is used to request approval, the corresponding definition is $\alpha = R/(S-1)W$.

rate of reads	R	0.864 /sec
rate of writes	W	0.039 /sec
message propagation time	m_{prop}	1.0 msec
message processing time	m_{proc}	0.25 msec
allowance for clocks	ϵ	100 msec

Table 2: Parameters for file caching in V.

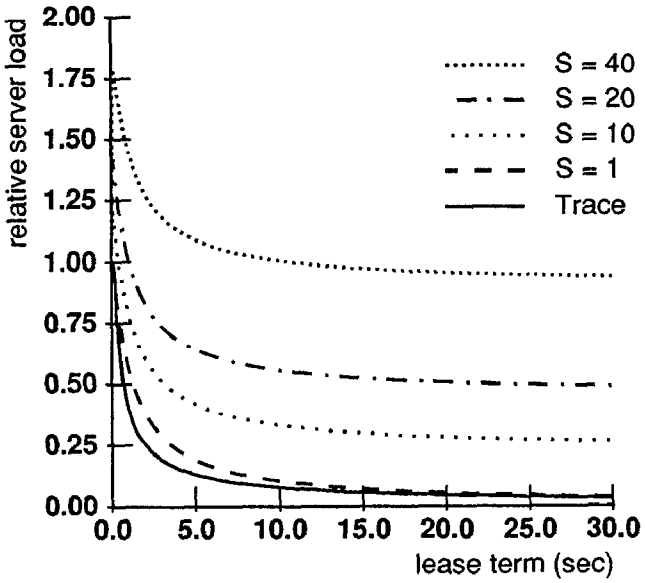

Figure 1: Relative Server Consistency vs. Lease Term

Figure 1 gives the relative server load for consistency as a function of the term, computed using formula 1 from Section 3.1. The curve labelled *Trace* was determined using a trace-driven simulation of the cache and server. The proximity of this curve to the no-sharing ($S = 1$) curve, derived from our analytic model, validates the model for this case. We note that the knee of the *Trace* curve is sharper and at a lower term. This (favorable) discrepancy is to be expected because actual file access is burstier than that given by a Poisson distribution. This burstiness implies that short terms should perform even better than our estimates indicate.

From Figure 1, most of the benefit of a non-zero lease term is gained by a term of just a few seconds. For example, at $S = 1$, a term of 10 seconds reduces the consistency traffic to 10% of that for a zero term. The load for consistency must be considered as it affects the total on the server. At a lease term of zero, consistency accounts for

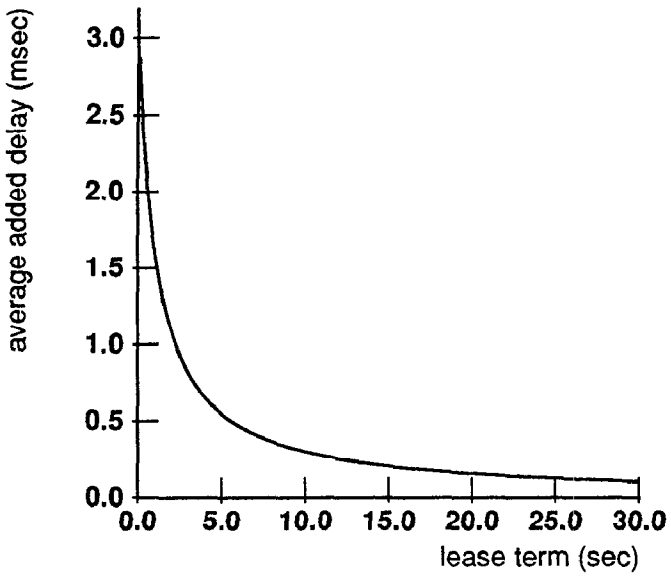

Figure 2: Delay due to consistency.

30% of the server traffic in the trace, so that the actual benefit is a 27% reduction in total server traffic, to a level just 4.5% above that for infinite term. At $S = 10$, total server traffic is 20% less than for a zero term and 4.1% over that for an infinite term. Longer terms provide relatively little additional reduction in server load yet introduce all the disadvantages of longer lease terms. Thus, a short lease term of (say) 10 seconds appears as a good choice for these file access characteristics, given the advantages of short leases described earlier and the insignificant reduction in server load provided by longer lease terms.

Figure 2 shows the average delay added to each read or write by consistency, as a function of the lease term. Because writes are a small fraction of all operations, the delay added to shared writes contributes little to the average delay, and the curves for $S = 1$ to $S = 40$ are indistinguishable in the graph as shown. Again, much of the benefit of leases is gained with lease terms in the 10 second range. Because many programs have significant compute time between file accesses, the improvement in response time for longer lease terms is insignificant.

We expect that the same result would apply to Unix-like systems even though our measurements of access rates are different from those that have been reported [8, 17] in longer-term traces of Unix systems. For example, our ratio of reads to writes is almost an order of magnitude higher that those reported elsewhere. Several factors account for this difference. First, operations on temporary files (which account for a large fraction of the writes) do not appear because they are handled specially by the V file cache, in a manner analogous to using a local disk for temporary files. Second, unlike most other traces, our measurements include program loading and access to information about files (such as directory lookups), both of which are predominantly reads. Finally, the *read* and *write* measurements correspond to when a file is opened for reading or closed (committed) with writing, as opposed to each time a block is read or written; the directory operations therefore are a larger fraction of the (logical) reads and writes.

When these factors are considered, the composition of this short trace is fairly consistent with those of the longer term traces of Unix systems. Only the last factor represents a departure from the more common semantics of the Unix file system; the other two factors are consequences of the cache design in V and might be profitably employed in a Unix system. Supporting Unix semantics, where read and write correspond to block-level operations, would give a higher absolute rate of reads, but a somewhat lower ratio of reads to writes (because the ratio of reads to writes for file blocks is lower than for other file-system data). The performance of leases in such a system would be qualitatively similar; the higher rate of reads would give the curves a sharper knee, favoring fairly short terms, while the more frequent writes makes it more sensitive to sharing.

3.3 Applicability to Future Distributed Systems

Several trends anticipate properties of future distributed systems. Systems are being extended over wider-area networks, increasing the delay for communication. The speed of processors also continues to grow. Finally, larger numbers of hosts, both clients and servers, are being tied together within a single system.

Larger propagation delay between clients and servers means that the impact of lease extensions and invalidations on response time is greater. Figure 3 shows the added delay on a network where the round-trip time is 100 milliseconds, while all other parameters remain as in our previous analysis. In this case, a 10 second term degrades response by 10.1% over using an infinite term and a 30 second term degrades it by 3.6%. Thus, with a significant increase in propagation delay, slightly longer lease terms may be appropriate, but terms in the 10-30 second range still appear to be adequate.

Faster client processors reduce the amount of time for computation between read and write requests, so that the number of operations occuring within a term increases. The higher rate pushes the knee of the load curve lower. The impact on application-level response time is almost identical to that of a slower network: the fraction of time spent in communication delay is larger, so that the signif-

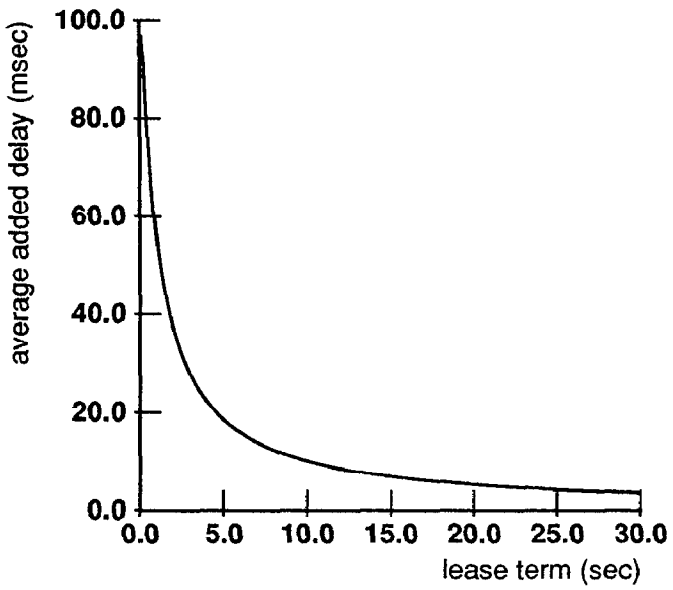

Figure 3: Added delay with 100 ms round-trip time.

icance of consistency is greater.

Increased numbers of clients and servers have no significant effect unless it increases the level of write-sharing, which we do not expect to be the case. In fact, there is no evidence that the level of write-sharing has increased over the very modest levels measured by Montgomery [15] in Multics over 12 years ago. Leases have the benefit of increasing the ratio of clients to servers (by reducing consistency overhead), thereby reducing the cost (or improving the performance) of large-scale systems.

4 Options for Lease Management

Lease management in the server admits several options that may be exploited to improve performance. The server controls the term of the leases it grants; it is also free to wait for a lease to expire instead of seeking approval of a write. The client is free in deciding when to request extension of leases, when to relinquish them, and when to approve a write. The combinations of these options give different trade-offs between load and response time.

For example, the client may anticipate the expiration of its leases and request extension before the covered file is accessed. Doing so improves response time by eliminating the added delay for reads, but it does so at the cost of increased load for the server. In particular, an idle client continues to request extensions even when files are not being accessed, and because the cache continues to hold leases it may increase the level of contention due to false sharing.

The server can use these options to optimize the handling of installed files, which account for a significant proportion of shared access. *Installed files* are files such as commands, header files and libraries which are part of the standard system support. These files are widely shared, heavily read and only infrequently written. In the trace taken from V, they account for almost half of all reads, but no writes. The handling of installed files is optimized by using a smaller number of leases to cover these files,[8] such as one per major directory, and multicasting an extension covering leases on installed files to all clients periodically, eliminating the need for clients to request extensions of these leases. Additionally, the server can simply eliminate a lease from the multicast extension when a file covered by the lease is to be modified. The write operation then proceeds as soon as the lease has expired. This approach eliminates the need for the server to contact a large number of clients when an installed file is updated and the resulting implosion of responses. Given the significant probability of the server having to wait for lease timeout because one of the many client machines is unreachable, write operations to installed files do not necessarily experience higher delay as a result of this optimization. This optimization also eliminates the need for the server to keep track of the leaseholders for installed files. Finally, it eliminates added delay at the client cache for reads of installed files because, in the absence of writes to installed files, these leases do not expire.

Finally, the server can set the lease term based on the file access characteristics for the requested file as well as the propagation delay to the client. In particular, a heavily write-shared file might be given a lease term of zero. A lease given to a distant client could be increased to compensate for the amount the lease term is reduced by the propagation delay and for the extra delay incurred by the client to extend the lease. In general, a server can dynamically pick lease terms on a per file and per client cache basis using the analytic model, assuming the necessary performance parameters are monitored by the server.

5 Fault-Tolerance

Leases ensure consistency provided that the hosts and network do not suffer certain Byzantine failures including clock failure. More specifically, consistency is maintained in spite of message loss (including partition), and client or server failures (assuming writes are persistent at the server across a crash). Moreover, availability is not reduced by the caches because an unreachable client at most briefly delays write access by other clients.

Leases depend on well-behaved clocks. In particular,

[8] Multiple files per lease can also result in a form of false sharing. We ignore this effect with installed files because the rate of update is so low.

a server clock that advances too quickly can cause errors because it may allow a write before the term of a lease held by a previous client has expired at that client. Similarly, if a client clock fails by advancing too slowly, it may continue using a lease which the server regards as having expired. The opposite errors—a slow server clock or fast client clock—do not result in inconsistencies, but do generate extra traffic since a client will regard leases to have expired before the server does, Such failures are much less common than either crashes or communication failures; they can be detected quickly by either a synchronization protocol or by including explicit timestamps in lease-related messages.

We also regard it as a reasonable assumption that clocks at the nodes of a distributed system are synchronized within ϵ which is small relative to the lease terms of several seconds. Synchronized time is required for other aspects of file access as well, such as the file-modified times used by the Unix *make* facility. As a minimum, the correct functioning of leases requires only that clocks have a known bounded drift, in which case the lease term can be communicated as its duration t.

6 Related work

Previous caching file systems that have guaranteed consistency have mostly used either a zero term or an infinite lease term. Sprite [16], RFS [1] and a prototype of the Andrew file system [18] use a zero-term lease at the granularity of file opens; Sprite and RFS use an infinite term while a file is open. The Andrew prototype experienced excessive server load from consistency checks as the system configuration was scaled [10]. Non-zero term leases appear applicable to all three systems with significant performance improvement over their current designs, especially with faster processors and larger network latency.

The later Andrew file system [10, 11] basically uses an infinite term, relying on the server to notify the client when cached data is changed. If communication with a client fails (at the transport level), the server allows updates to proceed, possibly leaving the client operating on stale data. The client does not learn of the error until it next attempts to communicate with the server; polling with a period of ten minutes is used to limit the interval for which inconsistent data may be used. Andrew uses a separate immutable volume for installed files to avoid the cost of their update under the normal mechanism. Our work suggests that a short-term lease would be adequate for Andrew, as opposed to the infinite term, allowing updates to be deferred in the client failure case long enough to avoid inconsistency. The other benefits of short leases are then available as well, including the ability to handle installed files well within the same framework, using the optimizations we have described.

Burrows's MFS [2] and the Echo file system of Mann et al. [13] both use *tokens*, which can be regarded as limited-term leases, but supporting non-write-through caches. With extension, our analysis of performance could be profitably applied to these systems.

Other systems have avoided the consistency problem by either not guaranteeing consistency, as done by NFS [21], or by prohibiting write-sharing, as done in the Cedar file system CFS [19]. We believe that the simplicity and efficiency of leasing together with the importance of consistency and write-sharing make these solutions less attractive in the future. In particular, we note that the *soft state* required for leasing is compatible with the so-called *stateless* interface used in NFS.

The Xerox DFS [20] uses breakable locks with timeouts, which are superficially similar to leases. However, the timeouts specify a minimum time before which a lock can be broken to avoid an excessive rate of transaction aborts. However, because clients do not use the lock timeout value and they are not reliably notified when a lock is broken, the scheme degenerates to leasing with a term of zero.

Mirage [7] provides a consistent distributed shared memory using infinite-term leases. Mirage augments this with a timer that (in terms of the leasing framework) specifies a minimum time after acquiring a lease before a client will relinquish it. This time can be increased to reduce the amount of thrashing, just as the lock timeout in DFS reduces the frequency of aborted transactions.

Time-based methods resembling leasing have also been used in at least two distributed naming systems. Lampson's global directory service [12] has client caches that discard entries at a server-specified time. Servers are forbidden from modifying an entry before it expires. This condition is equivalent to our policy for leases over installed files. However, no provision is made for either requesting approval of writes or for any extension of the terms.

Name services more commonly use cached data as *hints*, for which consistency need not be guaranteed. In the Internet Domain Name Service [14], for example, a name server specifies a *time-to-live* for the data it returns, and clients cache the data for that period. However, the data may be modified during that interval. Any inconsistency that results must be detected and corrected by other means. Terry [22, 23] discusses in more detail the caching of hints for name interpretation, including the use of on-use and periodic checks as options in maintaining the accuracy of the cache at the desired level.

Finally, the consistency work with caching shared memory systems has ignored the problem of communication and cache failures to date. However, leasing may represent a useful extension to consistency protocols

for large-scale multi-level shared memory multiprocessors [3].

7 Conclusions

Leasing is an efficient, fault-tolerant approach to maintaining file cache consistency in distributed systems. In this paper, we have analyzed its performance and evaluated performance in the context of a real system, examined its fault-tolerance properties, and considered its applicability to other distributed systems, especially large-scale, high-performance systems of the future.

Our simple analytical model estimates the server consistency load and consistency-induced delay to cache requests as a function of the lease term, the ratio of reads to writes, the degree of sharing and message times. This model provides a basis for a file server setting lease terms dynamically based on observed file access characteristics. In particular, it indicates when leases with a non-zero term reduce server load, given that high levels of write-sharing can make file caching ineffective. A trace-driven simulation using data from the V system provides (partial) validation of the analytic model.

A relatively short lease term is close to optimal with file access characteristics expected in Unix-like systems where the dominant file access is for software development and document preparation. In particular, using parameter values from the V system with this model, a lease term of 10 seconds results in a server load that is within 5 percent of that achievable with infinite term. We argued that the V file access characteristics are similar to those observed with various Unix-like systems. Short-term leases have a number of significant advantages over longer leases, including lower write delays resulting from client crashes, lower recovery delay from server crashes and reduced *false sharing*.

Leases appear well-suited to large-scale distributed systems. The improvement in response time that they offer is more significant for the faster processors and higher-delay networks. In this setting, the round-trip time to the client becomes a significant cost and potentially affects the choice of lease term. The lease overhead of handling large numbers of clients can be reduced by distinguishing different classes of files based on access characteristics. In particular, *installed files*—those with a high-degree of sharing and read access but low degree of writing—can be handled efficiently using multicast extensions from the server to extend the leases on directories of these files and delayed update to avoid the overhead of explicit lease invalidation.

Leases provide strict consistency in spite of non-Byzantine failures, including partitions. Failures result only in reduced performance, with their effect minimized by short lease terms. A key assumption is that clocks are reasonably accurate, at least in terms of drift if not mutual synchronization. We have argued that synchronized physical clocks are important in general in a system where files are shared in the manner supported by leases.

There are several limitations to this work. First, we have used a simplified model of file sharing and focused our evaluation on relatively low degrees of sharing. However, low degrees of sharing appear common in most systems. Exceptions that warrant further investigation include distributed transaction processing systems, distributed parallel programming systems and possibly systems that make extensive use of remote execution. Second, our analysis of performance is only approximate, since it ignores important factors such as queueing delays; nonetheless, our easily computed estimates are useful. Finally, there is limited experience with the use of leases in actual system operation. We are presently extending and tuning the file caching service within V, using the measurements of this service to further refine our model of performance and to gain further experience. We also plan to explore adaptive policies that vary the coverage and term of leases in response to system behavior in place of static, administratively set policies.

Leases have other applications besides file cache consistency. In particular, leases may also be applicable to large-scale shared memory multiprocessors. However, the benefits will have to be evaluated relative to the costs of timers on memory and cache lines, and the ability of the software to handle failures.

The lease approach is an example of a communication and coordination mechanism and reasoning based on (real) time, the availability of clocks that measure the passage of time with modest accuracy, and the ability to draw conclusions after a passage of time, possibly in the absence of communication. We are applying this general approach to other areas as well, including a distributed transaction management protocol and a transport protocol. We see this use of time as a fundamental aspect of distributed systems with potential for significant extension beyond that described here.

Acknowledgements. These ideas have benefited from discussions with many members of the Distributed Systems Group, of whom Joe Pallas has been especially helpful. The comments of the SOSP referees, and especially the more detailed reviews by Marvin Theimer and Doug Terry, have helped to improve the quality of this paper. The name "lease" was suggested by Morry Katz.

References

[1] BACH, M. J., LUPPI, M. W., MELAMED, A. S., AND

YUEH, K. A remote-file cache for RFS. In *Proceedings of the Summer 1987 Usenix Conference* (June 1987), Usenix Association, pp. 273–279.

[2] BURROWS, M. Efficient data sharing. Tech. Rep. No. 153, Computer Laboratory, University of Cambridge, Dec. 1988. The author's PhD thesis.

[3] CHERITON, D., GOOSEN, H., AND BOYLE, P. Multilevel shared caching techniques for scalability in VMP-MC. In *Proc. 16th Int. Symp. on Computer Architecture* (May 1989).

[4] CHERITON, D. R. The V distributed system. *Commun. ACM 31*, 3 (Mar. 1988), 314–333.

[5] CHERITON, D. R., AND DEERING, S. E. Host groups: A multicast extension for datagram internetworks. In *Proc. 9th Data Communications Symposium* (Sept. 1985), ACM/IEEE, pp. 172–179.

[6] CHERITON, D. R., AND ZWAENEPOEL, W. Distributed process groups in the V kernel. *ACM Trans. Comput. Syst. 3*, 2 (May 1985), 77–107.

[7] FLEISCH, B. D., AND POPEK, G. J. Mirage: A coherent distributed shared memory design. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles* (Dec. 1989), ACM.

[8] FLOYD, R. Short-term file reference patterns in a UNIX environment. Tech. Rep. TR 177, University of Rochester, Department of Computer Science, Mar. 1986.

[9] GRAY, C. G. *Performance and Fault-Tolerance in a Cache for Distributed File Service.* PhD thesis, Stanford University, Department of Computer Science, 1989. *In preparation.*

[10] HOWARD, J. H., KAZAR, M. L., MENEES, S. G., NICHOLS, D. A., SATYANARAYANAN, M., SIDEBOTHAM, R. N., AND WEST, M. J. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst. 6*, 1 (Feb. 1988), 51–81.

[11] KAZAR, M. L. Synchronization and caching issues in the Andrew file system. Tech. Rep. CMU-ITC-058, Information Technology Center, Carnegie Mellon University, June 1987.

[12] LAMPSON, B. W. Designing a global name service. In *Proceedings of the Fifth Annual ACM Symposium on the Principles of Distributed Computing* (Aug. 1986), ACM, pp. 1–10.

[13] MANN, T., HISGEN, A., AND SWART, G. An algorithm for data replication. Research Report 46, DEC Systems Research Center, 1989.

[14] MOCKAPETRIS, P. Domain names — concepts and facilities. Request for Comments 1034, Network Information Center, SRI International, Menlo Park, CA, Nov. 1987.

[15] MONTGOMERY, W. Measurements of sharing in MULTICS. In *Proc. of Sixth ACM Symposium on Operating Systems Principles* (1977), ACM, pp. 85–90.

[16] NELSON, M. N., WELCH, B. B., AND OUSTERHOUT, J. K. Caching in the Sprite network file system. *ACM Trans. Comput. Syst. 6*, 1 (Feb. 1988), 134–154.

[17] OUSTERHOUT, J. K., COSTA, H. D., HARRISON, D., KUNZE, J. A., KUPFER, M., AND THOMPSON, J. G. A trace-driven analysis of the UNIX 4.2BSD file system. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles* (Dec. 1985), ACM, pp. 15–24. Published as *Operating Systems Review 19*, 5.

[18] SATYANARAYANAN, M., HOWARD, J. H., NICHOLS, D. A., SIDEBOTHAM, R. N., SPECTOR, A. Z., AND WEST, M. J. The ITC distributed file system: Principles and design. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles* (Dec. 1985), ACM, pp. 35–50. Published as *Operating Systems Review 19*, 5.

[19] SCHROEDER, M. D., GIFFORD, D. K., AND NEEDHAM, R. M. A caching file system for a programmer's workstation. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles* (Dec. 1985), ACM, pp. 25–34. Published as *Operating Systems Review 19*, 5.

[20] STUGIS, H., MITCHELL, J., AND ISRAEL, J. Issues in the design and use of a distributed file system. *Operating Systems Review 14*, 3 (July 1980), 55–69.

[21] SUN MICROSYSTEMS, INC. *SunOS Reference Manual*, 1988.

[22] TERRY, D. B. Distributed name servers: Naming and caching in large distributed computing environments. Tech. Rep. UCB/CSD 85/228, Computer Science Division (EECS), University of California, Mar. 1985. The author's PhD thesis.

[23] TERRY, D. B. Caching hints in distributed systems. *IEEE Trans. Softw. Eng. SE-13*, 1 (Jan. 1987), 48–54.

[24] THOMPSON, J. G. Efficient analysis of caching systems. Tech. Rep. UCB/CSD 87/374, Computer Science Division (EECS), University of California, Oct. 1987. The author's PhD thesis.