An Analytical Study of Opportunistic Lease Renewal

Randal C. Burns, Robert M. Rees
Storage Systems and Software
IBM Almaden Research Center

Darrell D. E. Long
Department of Computer Science
University of California, Santa Cruz

Abstract

We present opportunistic renewal, a lease management protocol designed to keep distributed file systems or distributed shared memories consistent in the presence of a network partition or other computer failures. Our treatment includes an analytical model of the protocol that compares performance with existing lease protocols and quantifies improvements. In addition, this analytical model provides the structure to understand message overhead and availability trade-offs when selecting lease parameters. We include results demonstrating that opportunistic renewal substantially reduces the network overhead associated with lease renewal. As a corollary, opportunistic renewal can reduce the lease length at any given network overhead; e.g., by a factor of 50 at 1% network overhead. Lower overhead makes leasing less intrusive and shorter lease periods allow a system to recover from failure more quickly.

1 Introduction

Fast, efficient, and memory-consistent recovery from failure enables distributed applications to be reliably deployed on a large scale in a fault-prone environment. Internet applications such as transaction processing and data mining are increasingly data driven and require consistency guarantees which current Internet protocols fail to provide. Also, proxy caching benefits from strongly consistent data [3], but is deployed on large scale unreliable networks so that failures are a certainty. While data inconsistency errors are annoying to humans, they can cause distributed applications to produce incorrect results. For large scale applications, efficient recovery from failure and communication errors is essential. Fast recovery makes applications more responsive to users and reduces the costs associated with the unavailability of data. Improving lease efficiency and responsiveness is a key technology when deploying distributed applications on a large scale.

Since their inception in the **V** operating system [4], the semantics of leases have evolved to reduce network overhead and simplify failure and recovery. Distributed systems frequently employ leases to keep memories or file systems consistent in the presence of network or processor failures. In its simplest form, a lease is a fixed length time-out shared between a lease holder (client) and lease manager (server). Having decided when a lease starts, the client and server can use the fixed time-out period to agree upon the lease's expiration. Client and server use expiration as a synchronization point on which to coordinate activities, even if they are no longer able to communicate due to a network failure.

We develop an analytical model that can be used to understand the relative performance of different lease protocols and select appropriate lease intervals. In particular, we study a technique known as opportunistic renewal [2]. Our results show that opportunistic renewal causes lease overhead to become exponentially smaller, $\Theta(2^{-\tau})$, as the lease period τ increases, rather than inversely smaller, $\Theta(1/\tau)$, as is seen with other techniques. Opportunistic renewal can be used to reduce lease periods from the tens of seconds used in the original work [4] and in modern file systems [10] to less than a second without increasing network overhead.

Our analysis also shows that opportunistic renewal makes leases more highly available for networks that exhibit transient failures resulting from factors like routing errors, packet loss, and congestion. This result implies that leasing techniques apply not only to clusters of file systems, but to large scale networks, like the Internet, that exhibit this behavior.

The main contribution of this work is an analytical model that quantifies the benefits of opportunistic renewal and helps select suitable lease parameters. In developing the model, we observe that current lease protocols use a period that can be significantly reduced. With reduced lease periods, systems recover from failures more quickly, present data that is more highly available, and are more responsive to users and applications.

2 The Evolution of Leases

Leases, as introduced in the **V** operating system at Stanford, were originally used as a cache coherency construct [4]. The lease keeps memories consistent across many computers in the presence of network or processor failures. A lease in this system represents a contract entered by a server (the lease provider) and a client (the lease holder) in which the client receives a temporary privilege to read and cache a data object. Each lease has a period for which the server and the remainder of the distributed system agree to respect the holders ownership of a data object. In **V**, the lease period was chosen to be 10 to 20 seconds. The lease cannot be interrupted and a computer that wishes to write a data object must wait until outstanding leases expire before the write completes. While **V** provides a single lease for readers in a write-through caching system, this lease model can be extended for write-back caching by using two types of leases – exclusive for writers and shared for readers.

Leases have the advantage that they provide uniform operation and synchronized access to data even when computers fail or are isolated. Servers do not need to contact clients to revoke read privileges, because the client agreed ahead of time to a lease period. Uniform operation indicates that there is no special recovery protocol to handle the failure of a lease holder. Leases expire whether or not their holders have failed. Even when clients fail or the client/server communication is lost, caches are coherent.

The **V** lease model has performance, responsiveness, and scalability limitations. A computer must acquire a lease for every data object it accesses and must explicitly renew these leases when they expire. The large number of leases and their need for renewal can prohibitively burden the lease manager and the network. Also, since other computers cannot preempt leases, they must wait up to a entire period for the lease to expire before gaining access to shared data. The **V** lease model cannot scale to large distributed systems or provide timely access to highly shared data.

In order to reduce the overhead of lease renewal, subsequent system designers have used a single lease for each computer, rather than for each data object. The Frangipani file system [10] uses this technique to align the granularity of leases with the granularity of failure; *i.e.*, computers or networks are expected to fail, not individual data objects. This dramatically reduces the total number of leases and therefore reduces overhead.

With a lease per computer, leases alone are not adequate for cache coherency and preemptible cache coherency locks [6] must also be used. This approach employs locks for each data object and any lock is held in the context of a lease. If a lease expires and is not renewed, all locks become invalid. Separating cache coherency (locks) from failure and recovery (leases) allows the system to be more responsive and more efficient: locks can be preempted and need not be renewed. However, uniform operation is lost, because locks do not time out and after a failure they must be *stolen* [1] – locks outstanding on a failed client are labeled invalid and subsequently ignored by the server.

Leases may also be used hierarchically with short leases on individual data objects and long leases on collections of objects. Yin *et al.* [12] describe a system called *volume leases* designed for consistency in large-scale and weakly-connected systems. To access a data object, a computer must hold both a long lease on the object itself and a short lease on the collection that contains the object. The long lease on a single data object reduces the overhead of lease renewal. The short lease makes the system more responsive. The cost of renewing the short lease is amortized over all objects in the collection.

We observe that a lease per computer with cache coherency data locks is a variant of hierarchical leasing. The long lease on a data object corresponds to a cache coherency data lock, because data locks are essentially revocable leases with infinite term. The lease per computer is roughly analogous to a lease on a collection of objects, because it is a single lease protecting consistency on many data objects. However, the lease per computer approach has the advantage that the collection of objects are exactly the data objects being used by the computer at a given moment, rather than an immutable grouping of objects selected *a priori*. Using a lease per computer provides the same consistency guarantees with lower renewal costs than hierarchical leasing.

3 Leases in Storage Tank

We present the opportunistic renewal (OR) lease protocol implemented in the Storage Tank distributed file system [2] as the basis for our analysis. Like Frangipani [10], Storage Tank uses a single lease per computer to align the granularity of leases with that of failure. In addition to leases, Storage Tank uses data locks for cache coherency.

Unlike Frangipani and other systems, the Storage Tank lease protocol introduces two lease management techniques that improve performance: opportunistic renewal and *stateless serving*. Opportunistic renewal al-

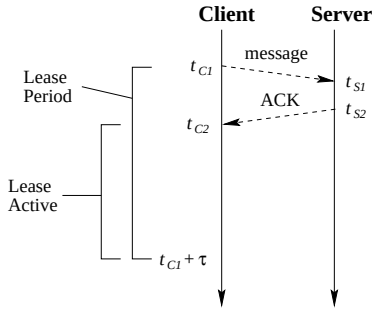

Figure 1: Client lease renewal.

lows a lease holder (client) to renew its lease implicitly on every message that it initiates, avoiding the network overhead of explicit renewal messages. Stateless serving permits the lease manager (server) to retain no state regarding client leases until a communication error occurs. The point of a stateless server is not to save memory at the server (as discussed by Yin *et al.* [12]), but to eliminate the need for the server keep all client leases on timers. This technique reduces both programming complexity and the processor overhead of scheduling and enforcing client lease time-outs.

3.1 The Opportunistic (OR) Renewal Protocol

Before developing the OR protocol, we state the network environment and assumptions required for the protocol to operate correctly. The OR protocol requires clocks at the clients and servers that are rate synchronized with a known error bound δ, *i.e.* an interval of length t when measured on one computer's clock has length that falls within the interval $(t/(1+\delta), t(1+\delta))$ when measured on the clock of another machine. It does not require absolute or relative time synchronization or Lamport clocks [7]. Our protocol operates in a connection-less network environment, where messages are datagrams. However, many messages between a client and server are either acknowledged (ACK) or negatively acknowledged (NACK), and include nonce identifiers for "at most once" delivery semantics.

A lease in this protocol defines a contract between a client and a server in which the server promises to respect the clients cache coherency locks for a specified period. The server respects the contract even when clients are unreachable. A client must have a valid lease on all servers with which it holds locks, and cached data become invalid when a lease expires.

Servers use the lease period to time-out client locks. If a server attempts to send a message that requires an ACK from a client, and the client does not respond, the server assumes the client to be failed. Having decided

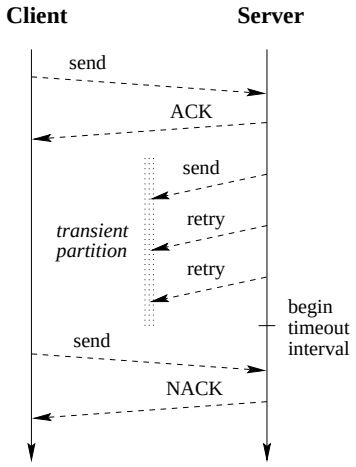

Figure 2: Servers negatively acknowledge messages when timing out clients.

the client is failed, the server starts a timer that expires at time $\tau(1+\delta)$ later, where τ is the contracted lease period. The server knows that $\tau(1+\delta)$ represents a time of at least τ at the client. Once the server waits out this timer, it may revoke the clients locks.

The key feature of the server's protocol is stateless serving – it retains no state about client leases. During normal operation, the server grants locks and ignores leases altogether. No lease-specific operations are performed and no server storage used. Only when a delivery error occurs does the server get involved by starting a lease timer. This passive design simplifies the implementation of the server and limits the performance impact of leases. A server that records client leases must have a local timer for each lease granted and must unschedule this timer and reschedule a new timer whenever a lease is renewed. This necessitates priority scheduling data structures and also frequent interrupt service routines for timers. By making the server protocol stateless during normal operation, we avoid implementation complexity and runtime overhead.

3.2 Renewal and Time-out

In the opportunistic renewal protocol, a client implicitly obtains or renews a lease with a server on every message it initiates (Figure 1). For example, at t_{c1}, the client sends a message to the server. The server receives this message at t_{s1} and acknowledges receipt at t_{s2}. Even without synchronized clocks, the client and server have an absolute ordering on events $t_{c1} \leq t_{S2}$. Upon receiving the ACK at t_{c2}, the client obtains a lease valid for the period $[t_{c1}, t_{c1}+\tau)$. This lease's period starts from when the client initiates the message, not when it receives the

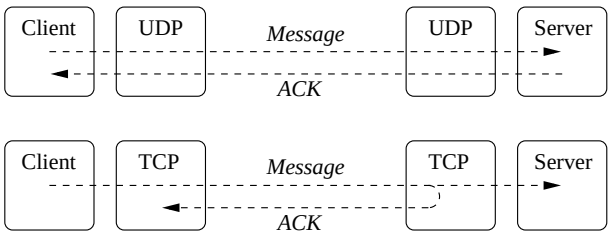

Figure 3: A comparison of application communication using TCP and UDP protocols.

ACK, because initiating the message is known to occur before the server's reply. While this lease is valid from $[t_{c1}, t_{c1} + \tau)$, from the client's perspective, it does not activate until t_{c2}, when the client receives an ACK, indicating it is in contact with a server. The client operates under this lease for the period $[t_{c2}, t_{c1} + \tau)$.

Transient failures and communication errors can result in a client, that believes it is operating on a valid lease, communicating with a server that is in the process of running a timer to reclaim client's locks. For correct operation, this asymmetry must be detected and addressed in the lease protocol. For example (Figure 2), a client may experience a transient network partition, miss a message, and recover communication with a server, without knowing that it missed a message. The server knows this client to have missed a message and begins timing out the lease. Having recovered communication, the client sends new requests to the server. The server can neither ACK the message, which would renew the client lease, nor execute a transaction on the client's behalf. Ignoring the client request, while correct, leads to further unnecessary message traffic as the client continues its attempts to renew its lease. Instead, the server sends a negative acknowledgment (NACK) in response to a valid request from a client that is being timed out.

The client interprets the NACK to mean that it has missed a message. It knows its cache to be invalid. The client, aware of its state, foregoes sending messages to acquire a lease, and prepares for recovery from a communication error.

Often, network partitions do not result in the server failing the client. A network partition always causes the client's lease to expire after at the end of the period. However, unless the server tries to send the client a message and fails, the contents of the client cache remains valid. If no communication errors occur, the server is unaware that the client's lease has expired. This is a side-effect of stateless serving. When the network recovers, the client sends its next message to the server and its lease is renewed.

4 An End-to-end Protocol

Our lease protocol requires end-to-end message delivery because the ACK and NACK message are overloaded to signify communication between the server application and the client application. An ACK from the server validates the contents of the clients cache and therefore implicitly renews the clients lease. A NACK instructs the client that its state is inconsistent with that of the server. The server application, not the networking layer, must generate the ACK or NACK for the overloaded semantics to be correct. Saltzer *et al.* [9] present a similar end-to-end argument for ACKs.

The need for end-to-end guarantees encouraged us to implement the OR protocol on UDP rather than TCP. In Figure 3, we observe that an ACK in TCP comes from the network protocol stack. Using TCP an ACK indicates only that the network layer is operational and can be obtained even if the server application has failed. Similarly, the NACK the OR protocol relies on to detect transient partitions cannot be implemented with TCP.

Distributed systems are often implemented with TCP so that they can take advantage of TCP's flow control and windowing, which significantly improve performance when transferring large bodies of data.

However, any file system protocol can be divided into data traffic, which obtains performance benefits from TCP, and control traffic – synchronization and metadata – which consists of small messages that do not utilize TCP's features. This distinction has been made frequently in file systems in order to separate the metadata workload from the data workload so that they do not contend on the same physical resource [8]. Data requests are also separated so that they may be sent to high-performance storage hardware [11, 5]. Most distributed system protocols can be differentiated into data intensive operations and control operations and can choose UDP for the latter. Storage Tank's protocol [2] has a natural division of workload, because data access and control messages are performed on different networks.

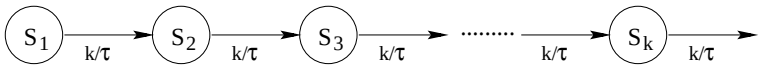

Figure 4: The Erlang-k random variable.

5 Lease Protocol Model

We build an analytical model of the OR lease protocol based on Markovian and steady state assumptions. Our model captures both opportunistic and *explicit* lease renewal, where explicit lease renewal indicates all protocols that send a renewal message when a lease times out [10, 4, 12]. The model describes the lease protocol from the perspective of the client (lease holder) and evaluates performance metrics for single clients only. While we will not consider systems with many lease holders, previous research [4, 12] sums the contribution of each client computer when evaluating many network and server performance metrics. The same technique can be used to extrapolate our results to large systems.

5.1 Modeling a Time-out

The key feature of a lease to be modeled is its duration (period). At the core of a fixed length lease is a structure in the model that describes a time-out.

To accurately model a time-out, we use the sum of a large number of independent and identically distributed (i.i.d.) exponential random variables. The lease itself is the sum of k exponential random variables each with parameter k/τ (Figure 4); the mean of each distribution is τ/k and the variance τ^2/k^2. This construction forms a summation of distributions that takes on values close to τ (the lease period) with high probability. In fact, we choose k large enough so that the expected time to complete the system, which we call $\mathcal{S}_k$ because it is the sum of k i.i.d. variables, differs from the mean by less than a factor of δ (the clock skew) with confidence ϵ: *i.e.*, $\Pr[|\mathcal{S}_k - \tau| \geq \delta\tau] \leq \epsilon$. By choosing a large number of states, we are able to approach this mean with arbitrary precision.

We observe that this construction is identical to the Erlang-k random variable with parameter k/τ, mean τ, and variance τ^2/k. We can use Chebyschev bounds on the Erlang-k variable to evaluate the probability that the variable is close to the mean. Chebyschev bounds state that $\Pr[|x - \mu| \geq a] \leq \sigma^2/a^2$ for random variable x with mean μ, and variance σ^2. Applying this to the OR protocol model, $\Pr[|S_k - \tau| \geq \delta\tau] \leq 1/(k\delta^2) \leq \epsilon$. Based on this bound, we need to choose $k \geq 1/(\delta^2\epsilon)$ to model lease periods within $\delta\tau$ of τ with confidence ϵ.

Tighter bounds can be achieved by treating the system as a sum of exponential variables, rather than a single Erlang-k random variable, and using the central limit theorem (CLT) to choose k. The CLT states

$$\Pr\left[x \leq \frac{\mathcal{S}_k - k\mu}{\sigma\sqrt{k}} \leq y\right] \approx \Phi(y) - \Phi(x), \qquad (1)$$

where $\mathcal{S}_k$ is again the sum of the k i.i.d. exponential random variables and Φ is the normal distribution function. For the OR protocol model, $x = -y = (\delta\tau)/(\sigma\sqrt{k}) = \delta\sqrt{k}$. For ϵ confidence, the CLT dictates that $\Phi(\delta\sqrt{k}) - \Phi(-\delta\sqrt{k}) \geq 1 - \epsilon$.

Having selected values for δ and ϵ, we derive the number of states (k) that are required to satisfy the bounds. For 99% confidence ($\epsilon = 0.01$) that the lease is accurate to within 10% ($\delta = 0.1$), the CLT requires us to model 676 states. Based on Chebyschev bounds, we would require 10,000 states for the same confidence. While 676 is a large number of states, an analytical model of this size can easily be evaluated numerically.

5.2 Analyzing Leases

Based on this technique for modeling a lease, we construct a Markov model of the lease management protocol at a client using opportunistic renewal (Figure 5). The model represents a state machine describing events from the perspective of a lease holder. The states S_1 to S_k describe the lease period while the client holds a valid lease and communication with the server is possible. Similarly, states F_1 to F_k describe the lease period with the client unable to communicate with the server due to a network partition or server failure. States S_x and F_x describe the lease period having expired with or without the interconnect being available respectively.

Transitions among these states represent the passage of time; transitions from S_i to S_{i+1} or from F_i to F_{i+1} occur at rate k/τ. Both the S and F chains with states $1, \ldots, k$ describe the lease time-out using the Erlang-k random variable construction.

Transitions also represent system events. The end-to-end interconnect with the server fails at rate λ and is repaired at rate μ; transitions from network available states to network failed states (S_i to F_i) occur at rate λ and the opposite transition occurs at rate μ. Messages that explicitly renew leases complete at rate σ and transition the model from no lease to a new lease (S_X to S_1). A client initiates messages that can opportunistically renew a lease at rate ρ. Such transitions can take the model

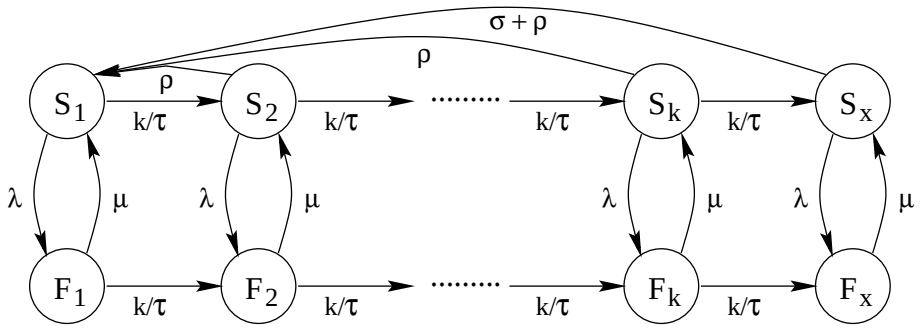

Figure 5: Model of an opportunistically renewed lease.

$$
\begin{aligned}
&(k/\tau + \lambda)\Pr(S_1) = \mu \Pr(F_1) + \rho \sum_{i=2}^{k} \Pr(S_i) + (\sigma + \rho)\Pr(S_x) \\
&(k/\tau + \mu)\Pr(F_1) = \lambda \Pr(S_1) \\
&(\sigma + \rho + \lambda)\Pr(S_x) = k/\tau \Pr(S_k) + \mu \Pr(F_x) \\
&\mu \Pr(F_x) = k/\tau \Pr(F_k) + \lambda \Pr(S_x) \\
&(k/\tau + \lambda + \rho)\Pr(S_i) = k/\tau \Pr(S_{)i-1} + \mu \Pr(F_i), \quad \text{for } i = 2 \ldots k \\
&(k/\tau + \mu)\Pr(F_i) = k/\tau \Pr(F_{i-1}) + \lambda \Pr(S_i), \quad \text{for } i = 2 \ldots k \\
&\sum_{i=1}^{k} (\Pr(S_i) + \Pr(F_i)) + \Pr(S_x) + \Pr(F_x) = 1
\end{aligned}
$$

Figure 6: Balance equations.

from any state where the network is available (S_i) to a new lease (S_1). The variable ρ describes the message rate for all non-lease traffic between a client and server. In the Storage Tank file system, this traffic consists of metadata operations and lock management.

Given the assumption that all random processes are Poisson and the system achieves a steady state, a solution to the model is obtained by solving the balance equations (Figure 6).

This model of a lease can also be used to describe explicit lease renewal as used by other protocols [12, 4, 10]. By setting the variable that describes the rate of opportunistic renewal (ρ) to zero, all leases are renewed explicitly in a transition from state S_x to S_1 at rate σ.

5.3 Renewal Overhead

Solutions to the model show that opportunistic renewal substantially reduces network overhead when compared with explicit renewal. More specifically, we find that the network message overhead decreases exponentially, $\Theta(2^{-\tau})$, as the lease period increases when using opportunistic renewal. This compares to an inverse variation, $\Theta(1/\tau)$, in network message overhead when using explicit renewal.

In Figure 7 we display overhead results comparing opportunistic and explicit lease renewal. The rate at which leases are opportunistically renewed (ρ) is also rate for all messages that are not lease related (with inter-arrival time $1/\rho$). Overhead is unit-less and measured as the rate of messages for explicit renewal divided by the mean message rate, expressed by the quantity $\sigma \Pr(S_x)/\rho$. The lease period is also unitless and describes the lease period scaled as a multiple of the inter-arrival time: *i.e.*, $\tau\rho$.

When the lease period is short – shorter than mean message inter-arrival time – opportunistic renewal does not significantly improve overhead as compared to explicit renewal. With such a short lease period, there are few chances to opportunistically renew, because the lease times out much faster than messages are sent.

When leases are long, we see that renewal overhead decreases exponentially ($\Theta(2^{-\tau})$) with respect to the lease period (linear decrease on a log scale). Regular protocol messages occur much more frequently than a lease times out, and explicit renewals are needed infrequently.

Network overhead for explicit lease renewal declines inverse-linearly as the lease period increases. There is a simple intuition behind this result. Leases are renewed periodically, as they expire, and the overhead is the fre-

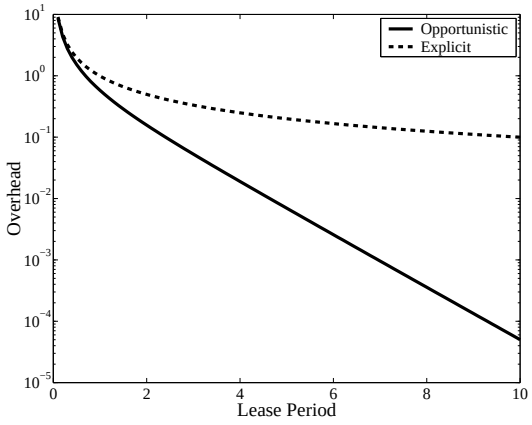

Figure 7: Renewal message overhead.

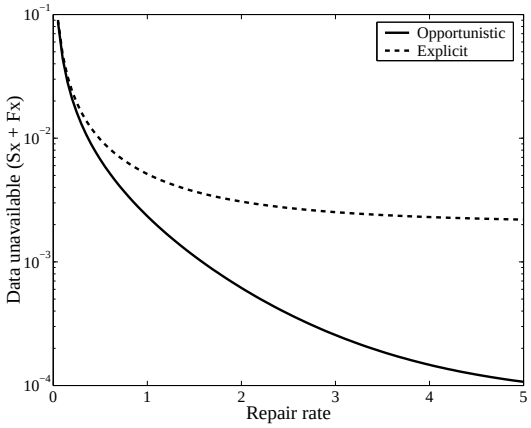

Figure 8: Client data availability.

quency of this renewal compared to the frequency of all other messages: $1/(\tau\rho)$.

Figure 7 confirms our claims to the scaling of opportunistic renewal. This graph shows that the network overhead at any given lease period is reduced; *e.g.*, from 10^{-1} down to 5×10^{-5} at a lease period of $10/\rho$. We also conclude that opportunistic renewal can greatly reduce the lease period for a fixed amount of network overhead. At 100% overhead (10^0, one renewal message for every non-lease related message), opportunistic renewal reduces the lease period only a small amount. However, at 10% network overhead, the lease period can be reduced to $2.4/\rho$ from $10/\rho$, reducing the lease period by over a factor of 4. Extending beyond the chart for explicit renewal, we compare at 1%. For opportunistic renewal, a period of $4.7/\rho$ compared to $100/\rho$, reducing the lease by a factor of more than 10. For 0.1% overhead, opportunistic renewal requires a lease period of $7/\rho$ compared to $1000/\rho$: a factor larger than 140. Opportunistic renewal can reduce the overhead by an arbitrarily large factor by choosing longer lease periods.

While lengthening the period reduces overhead, we note that the length of the lease remains short from a system perspective. Recent benchmark results on local and distributed systems [10] indicate that the file system on any single computer must support over 1000 metadata operations per second to be performance competitive. If we look at workloads of this order distributed over a server cluster of 100 computers, a client communicates with each server 10 times a second. In this environment, a lease period of one half second produces overhead of less than 1%. Comparing this to the lease periods in the tens of seconds used by existing systems [10, 12, 4], we conclude that opportunistic renewal allows a leasing system to respond to failures in a fraction of the time with no additional overhead.

5.4 Data Availability at the Lease Holder

Our analysis also shows that opportunistic renewal can lead to better data availability in the presence of short-lived communication failures. These results indicate the suitability of opportunistic renewal in wide-area or weakly connected environments, like the Internet, where communication errors are more frequent and transient. Losing a lease during an intermittent communication error negatively affects the system, because data are unavailable at the disconnected client.

Figure 8 compares the repair rate of the network to the fraction of time when data are unavailable. For network failure and repair we are concerned with failures attributed to factors like congestion, packet loss, and routing errors, rather than hardware failures that are longer lived. Repair rate is normalized to the frequency of lease expiration; *i.e.* a value of two indicates that the network repairs at twice the rate at which leases expire, $2/\tau$. Data unavailable indicates the amount of time (as a fraction of 1) for which a client has no lease and therefore cannot access data. We observe that as the renewal rate increases, clients using opportunistic renewal lose their lease much less frequently than explicitly renewing clients. Looking at the OR protocol model (Figure 5), this means that opportunistic renewal makes lower valued S and F states more probable than high valued ones. With explicit renewal all S states have equal probability, as do all F states.

Availability results indicate that for weakly connected distributed systems, opportunistic renewal performs well by renewing the lease on every message when the network is available. Therefore, after a network failure, a client has a larger fraction of the lease period before expiration. This effect is more pronounced for higher repair rates where the network can repair before the lease expires. For weakly connected distributed sys-

tems, designers can select the lease period to be several times longer than the expected network repair rate to increase availability.

Increased data availability does not affect recovery time at the server. While it may seem that having the lease valid for longer after a network failure would lead to the server waiting longer before recovering state from an isolated client, we recall (Section 3.1) that the stateless server must wait a whole lease period after a communication error regardless of the renewal protocol.

6 Conclusions

Through the modeling of lease protocols, we are able to quantify trade-offs between lease period and network overhead to aid in parameter selection, determine the benefits of opportunistic lease renewal, and draw conclusions about the impact that leases can have on distributed systems. Opportunistic renewal exponentially reduces the network overhead of a lease protocol. This also means that opportunistic renewal drastically reduces lease periods. We also found that opportunistic renewal increases data availability at clients when networks fail and repair intermittently.

Our results indicate the suitability of leases with opportunistic renewal for providing consistency guarantees in large-scale distributed systems. Lease periods are reduced making systems more responsive when failures occur. Also, at any given lease period, network overhead is reduced and availability increased. Lease protocols are a powerful technique for deploying data-consistent applications in the Internet and opportunistic renewal improves their performance properties in this environment.

Acknowledgements

We would like to thank Wayne Hineman, David Pease, Michael Penner and all the members of the Storage Tank team at the IBM Almaden research center for their contributions to this work. We would also like to our anonymous reviewers for their insightful feedback and corrections.

References

[1] M. L. G. Baker. *Fast Crash Recovery in Distributed File Systems*. Ph.D. dissertation, University of California at Berkeley, 1994.

[2] R. C. Burns, R. M. Rees, and D. D. E. Long. Safe caching in a distributed file system. In *Proceedings of the International Parallel and Distributed Processing Symposium*, May 2000.

[3] P. Cao and C. Liu. Maintaining strong cache consistency in the world wide web. *IEEE Transactions on Computers*, 47(4), April 1998.

[4] C. G. Gray and D. R. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, December 1989.

[5] J. H. Hartman and J. K. Ousterhout. The Zebra-striped network file system. In *Proceedings of the 16th ACM Symposium on Operating System Principles*. ACM, 1993.

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

[7] L. Lamport. Time, clocks and the ordering of events in a distributed systems. *Communications of the ACM*, 21(7), July 1978.

[8] K. Muller and J. Pasquale. A high performance multi-structured file system design. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, 1991.

[9] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transaction on Computer Systems*, 2(4), November 1984.

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

[11] R. W. Watson and R. A. Coyne. The parallel I/O architecture of the high-performance storage system (HPSS). In *Proceedings of the 14th IEEE Symposium on Mass Storage Systems*, 1995.

[12] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Volume leases for consistency in large-scale systems. *IEEE Transactions on Knowledge and Data Engineering*, 11(4), July/August 1999.