# CONSISTENCY AND SAFE CACHING IN DISTRIBUTED FILE SYSTEM BY USING LEASE BASED MECHANISM

**Prabodh S. Nimat***

**Prof. A.R.Itakikar***

**Abstract**—

In this paper, we argue that weak cache consistency and unsecured caching in distributed file system environment supported by existing mechanism and protocol, and must be augmented by strong consistency mechanisms and protocol to support the growing diversity in application requirements. There are many methods to maintain consistency in the distributed file system but provides weak consistency. But one method provides strong consistency and safe caching as well in the distributed environment is the lease method and it also provide safe caching in distributed environment. During the proxy's lease time from a web server, the web server can notify the modification to the proxy by invalidation or update. In this paper, we have shown some analysis parameter. By using these analyses, we can choose the adaptive lease time and proper protocol (invalidation or update scheme of the modification for each proxy in the distributive file environment). As the number of proxies for web caching increases exponentially, a more efficient method for maintaining consistency needs to be designed.

***Keywords***— leases, strong consistency, cache consistency, safe caching, invalidation, update and locks.

* M.E. 1st Year, Department of Computer Engineering. Sipna COET , SGB Amravati University, India.

## I. Introduction

In distributed file system proxy must maintain the consistency of cached data to preserve them from stale information. It can be achieved by using leases A lease mechanism defines a contract between a client and a server in which the server promises to respect the clients locks for a specified period. The server respects the contract even when clients are unreachable. Leases use the invalidation or update scheme during the lease time for maintaining consistency by notifying about the stale information. In this paper we analyze the notification protocol overhead of maintaining cache consistency during the lease time. Ideally, efficient consistency maintaining schemes should take into account the following factors: lease duration, data access pattern and system parameters (resources overhead such as CPU time, storage cost and communication delay). For example, in general cases the invalidation scheme is efficient when many remote updates occur consecutively [7]. But in a short lease time, an update scheme could be more efficient than invalidation because remaining remote updates are ignored after lease expiration time. It is interesting to analyze how lease duration, modification notice schemes (update scheme and invalidation scheme) affect performance. Thus, in this paper we analyze performance by varying lease duration, modification scheme and local access ratio to remote modification. To reduce overhead, an efficient lease duration and modification notice scheme needs to be adjusted according to data access patterns and system parameters. By analyzing the cost of maintaining consistency according to lease time, shared data access pattern.

In this paper, we argue that finding the optimal notification scheme in lease duration is a critical factor in reducing the message overhead because lease duration affects the efficient notification scheme. There is a tradeoff between data fault and local update. As lease time increases data access fault (data miss) decreases while local update cost increases by data update from the server. On the other hand, as lease time decreases, these cases are reversed. Our work focuses on developing a dynamic notification scheme along with read rates, read and write intensity, and to determine optimal lease duration in a lease-based consistency mechanism.

### *A. Existing Cache Consistency Mechanisms:*

*Benefits and Limitations*

A cache consistency mechanism that always returns the results of the latest write at the server is said to be strongly consistent. Due to the unbounded message delays in the Internet, no cache consistency mechanism can be strongly consistent in this idealized sense. Hence, we relax our definition to the following: a mechanism that returns data that is never outdated by more than t time units with the version on the server is said to be strongly consistent, where t is the server to proxy delay at that instant and 0 < t_1. Mechanisms that do not satisfy this property (i.e., can return stale data) are said to be weakly consistent. Most existing proxies provide only weak consistency by 1) employing a server specified lifetime of an object (referred to as the time-to-live (TTL) value) or 2) periodically polling the server to verify that the cached data is not stale [4], [12], [13]. In either case, a modification to the object before its TTL expire or between two successive polls causes the proxy to return stale data. Strong consistency can be enforced either by server-driven mechanisms or client-driven mechanisms [14]. The former approach, referred to as server-based invalidation, requires the server to notify proxies when the data changes. This approach substantially reduces the number of control messages exchanged between the server and the proxy (since messages are sent only when an object is modified). However, it requires the server to maintain per-object state consisting of a list of all proxies that cache the object; the amount of state maintained can be significant, especially at popular Web servers. Moreover, when a proxy is unreachable due to network failures, the server must either block on a write request until a timeout occurs, or risk violating consistency guarantees. The client-driven approach, also referred to as client polling, requires that proxies poll the server on every read to determine if the data has changed [14]. Frequent polling imposes a large message overhead and also increases the response time (since the proxy must await the result of its poll before responding to a read request). The advantage, though, is that it does not require any state to be maintained at the server, nor does the server ever need to block on a write request (since the onus of maintaining consistency is on the proxy). Server-based invalidation and client polling form two ends of a spectrum. Whereas the former minimizes the number of control messages exchanged, but may require a significant amount of state to be maintained, the latter is stateless, but can impose a large control message overhead. Fig. 1 quantitatively compares these two approaches with respect to 1) the server overhead, 2) the network overhead, and 3) the client response time. Due to their large overheads, neither approach is appealing for Web environments. A strong consistency mechanism

suitable for the Web must not only reduce client response time, but also balance both network and server overheads.

## II. Literature review

Caching by proxy is necessary to reduce the network overhead at the server. To achieve this, replicating the data object by analyzing read/write patterns and resources [8], [9] can be effective. However, if an object is replicated to many proxies and tries to update the object, it needs the consistency mechanism to reflect the changed data. Since web pages tend to be modified at origin servers, the cached versions of these pages can become inconsistent at the page change from the server. To resolve this problem, numerous schemes to maintain the consistency are proposed. Atomic write transactions are used to preserve the consistency [10]. An efficient consistency system reducing unnecessary delay is performed by using an update ordering model . Lease, invalidation and update schemes are the common methods to supply consistency.

### *A. Invalidation and update*

If an object is modified during the lease time, the server reports the modification to the proxy. There are two methods to notify to clients. One is invalidation and the other is update. The former invalidates the proxy when it receives the modification message Therefore, subsequent read requests make the proxy require the page from the server. When modification of the page occurs frequently in a short time, invalidation is very efficient [15]. But when the object is modified infrequently, the turnaround delay overhead to access the page increases. On the other hand, update doesn't need the turnaround delay because the object or modified part of the object is sent upon each modification [16]. Another update algorithm based on the user's coherency tolerances reduces the load on repositories [17]. This method has the drawback of the network overhead.

### *B. Competitive update*

In write-update protocol, if cache data is loaded, the proxy keeps the cache data regardless of local data accesses. However, there is a problem that the cached data block must be updated many times without local access. To resolve this problem while maintaining the advantage of the write-update, local cache data must be invalidated after the critical update count. Whenever a remote

client requests an update, the update counter is increased. When the counter reaches critical update count, the cache data is invalidated upon remote update [16]. The critical update counter for each cache block have a different value and be changed adaptively.

*C. Adaptive scheme*

To reduce the network bandwidth and latency, the invalidation and update scheme must be applied adaptively by the time varying memory access patterns of an application [17]. An adaptive scheme chooses one of the invalidation protocol and competitive update protocols according to the data access pattern and system parameters.

## III. Cost analysis for the lease-based efficient notification scheme.

If an object is modified during the lease duration, the lease server notifies the proxy of any modification made to the object. The client is not required to poll the server during the lease duration even if read operations occurred consecutively. If a page notification which occurs after read operations are executed by remote write operation in lease duration, notification is processed by three methods, which are invalidation, update and competitive update[1]. To determine which notification method should be used to maintain the consistency is critical to reducing the message overhead. This problem is not easily manageable because it is based on the lease mechanism. Early studies [16] showed analytical comparisons of invalidation, update and competitive update without the lease by using the segment model. We consider the cost of messages as the cost metric for the invalidation, update and competitive model. We assume that particular page P at the proxy cache is accessed by clients. These accesses can be partitioned into segments. A segment is defined as a sequence of remote updates between two consecutive local accesses by a node. A new segment begins with the first access by a client following an update to the page by the server. Segments are defined from the point of view of each node.

Table 1. Parameter for analysis

| Parameters | Description |
|---|---|
| $C_{control}$ | Cost to send the control message |
| $C_{upd}$ | Cost to update a page of the proxy cache |
| $C_{page}$ | Cost to replace a page in the proxy cache |
| $U_{comp}$ | Competitive update count |
| $u_{avg}$ | Average update count per segment |

A segment is the basic unit to analyze the notification scheme (update, invalidate and competitive update). Parameter for analyze the cost are shown in Table 1. The cost of notification protocols in a segment are computed as follows:

Invalidate overhead cost = $(C_{contorl} + C_{page})$ (1)

Update overhead cost = $(u_{avg}C_{upd})$ (2)

Competitive update

$$= \begin{cases} UavgCupd \text{ (when Update Count } \leq 4) \\ UavgCupd + (Ccontorl + Cpage) \text{(if Update Count } > 4) \end{cases} \quad (3)$$

Eqs. (1)–(3) means the cost of each notification scheme. In this analysis, we assume $C_{contor}$= 5, $C_{upd}$ = 10, $C_{page}$ = 35 and $U_{comp}$ = 4. The assumptions are based on web page size (page replacement), frame size (update) and proxy control size (control). We assumed an update count threshold of 4 for the competitive update protocol. When remote write occurs less than the

threshold count, the competitive update protocol notifies the proxy by update and when remote write occurs more than the threshold count, it reports by invalidation**.**

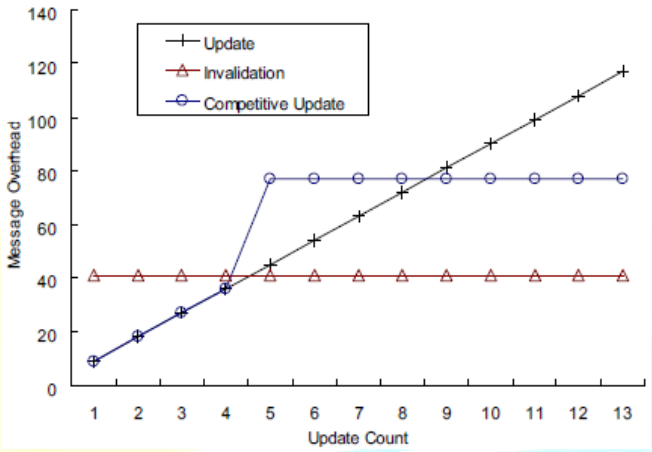


Fig.1. Compare the notification scheme

Update and competitive update protocols are better choices if the update count is smaller than 5. However, invalidation is the best choice when the update count is greater than 5.

*A. General case analysis of the lease-based notification scheme*

Interracial times of read and write requests in a wide distributed area are generally assumed to be random. We evaluate our proposed scheme when interracial times among successive local reads and among successive remote writes follow not only exponential distribution but also Gaussian distribution. Fig. 15 shows that the result of exponential distribution is similar with those of Gaussian distribution. In general, the occurrences of write requests in the web environment are supposed to follow the exponential distribution [12]. Therefore, proposed simulation is performed with the exponential distribution for read and write request at the remaining experiments. In the general case, it is hard to extract the special feature for deciding the lease duration or the notification scheme. To analyze the cost, we divide the general case into two cases. One case is that the lease duration is longer than the segment length and the other case is that the lease duration is shorter than the segment lengthwise the Markov model for cost analysis. Table 2 explains the parameter of the model of the lease scheme.

*B. Lease duration longer than segment length*

In this case we assume that more than one segment is formed

during the lease duration. The parameters to analyze the cost are shown in Table2.

Table 2. Parameters for analysis

| Parameters | Description |
|---|---|
| N | The number of the segment in lease (n > 1) |
| p(k) | Probability to be kth consecutive update |

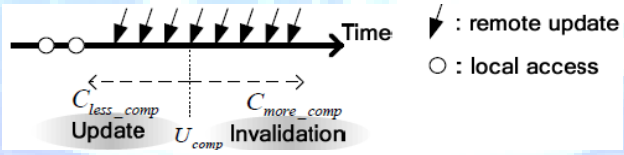


Fig. 2. Conditional cost when using the competitive update ($U_{comp}$ = 4).

When each segment begins with a first read, the proxy cache needs to obtain the page by sending a control message. If n segments are included during the lease, overhead cost for lease invalidation case is as in Eq. (4):

Lease invalidate overhead cost = $n(C_{control} + C_{page})$. (4)

In the update case, page replacement cost is requested at the read of the first segment, and every update requirement needs the update cost at Eq. (5).

Lease update overhead cost = $n(u_{avg}C_{upd}) + C_{page}$. (5)

This gives an idea about overhead cost of each notification scheme while using leases in distributed network.

## IV. Lease Based Safe Caching

A safety protocol that protects the consistency of data and guarantees cache coherency must 1) protect the contents of a file by preventing isolated clients from writing data to disk after their locks are stolen, and 2) allows isolated clients to write the dirty contents of its cache to shared storage before its locks are stolen. Fencing fails to address the second point and clients behave incorrectly, giving stale data to local processes and losing written data. Leasing improves the semantics of failure and recovery when clients become isolated. For client failures or failures in distributed file system leasing offers improvements over fencing. Proposed lease-based safety protocol has performance advantages when compared to other leasing systems. These include optimizations that allow leases to be renewed opportunistically, eliminating message traffic during normal operation, and a design that allows for a "passive" server that participates in the protocol only when an error occurs. Before developing the protocol, we state the network environment and assumptions required for the protocol to operate correctly. The goal of the protocol is to ensure data integrity and cache coherency among clients accessing network attached storage devices. To this end, proposed protocol addresses arbitrary partitions in the control network, including asymmetric partitions.

Proposed protocol requires clocks at the clients and servers that are rate synchronized with a known error bound _, *i.e.* an interval of length t when measured on one computers clock has length that falls within the interval _))when measured on the clock of another machine[2]. It does not require absolute or relative time synchronization, or Lamport clocks [18]. The protocol operates in a connection-less network environment, where messages are datagram. However, many messages between the Storage Tank client and server are either acknowledged (ACK) or negatively acknowledged (NACK), and include version numbers 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' 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 leases 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. Clients that are isolated instead of

failed have missed a message. They either have stale metadata or have not properly participated in a locking protocol. However, this does not yet result in a violation of file system semantics. Missing a metadata message is tolerable; because file systems only guarantee that metadata are weakly consistent1. Missing a lock protocol message is also acceptable, because locks and locked data are protected until the lease expires. Having decided the client is failed, the server starts a timer that goes off at a time _(1 + _) later, where _ is the contracted lease period. The server knows that _ represents a time of at least _ at the client. Once the server waits out this time, it may steal the client's locks. The client is responsible for ensuring that all dirty data are written out by this time, and that the data and metadata it caches with respect to this server are invalid. The key feature of the server's protocol is that it retains no state about client leases. During normal operation, the server merely grants locks and ignores leasing 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 leasing.

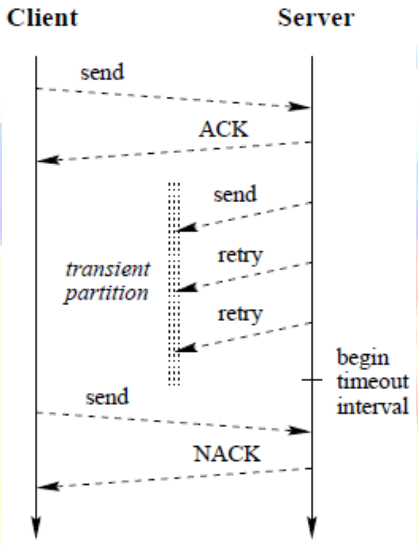


Figure 3. Servers negatively acknowledge messages when timing out clients.

Transient failures and communication errors can result in a client that believes it is operating on valid leases, communicating with a server that is in the process of running a timer to steal the client's locks. For correct operation, this asymmetry must be detected and addressed in the lease protocol.

For example, a client that experiences a transient network partition misses a message and recovers communication with a server, without knowing it missed the message. The server knows this client to have missed messages and begins timing out the lease, with the knowledge that its cache is invalid. Having recovered communication, the client sends new requests to a server (Figure 3).

This works in three phases, each phase corresponds to a legal lease holder. The server can neither acknowledge 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 when the client attempts to renew its lease in phase 2. Instead, the server sends a negative acknowledgment (NACK) in response to a valid request from a suspect client. The client interprets the NACK to mean that it has missed a message. It knows its cache to be invalid and enters phase 3 of the lease interval directly. The client, aware of its state, forgoes sending messages to acquire a lease, and prepares for recovery from a communication error.

## V. Conclusion

In this paper we analyze the notification protocol overhead to maintain cache consistency during lease time. By implementing a data safety protocol in a distributed system using leases, protects the consistency of data at little or no runtime cost. We can see two consistency properties from the experiment. One consistency properties is that the update scheme shows better performance by reducing the message overhead as remote writes happen infrequently. The other property is related with the lease scheme. As read request occurs frequently, the lease scheme shows better performance than other consistency maintenance schemes without lease. When read requests occur more frequently than write requests, the lease-based scheme is very efficient.

**References**

[1] Byoung-Hoon Lee , Sung-Hwa Lim, Jai-Hoon Kim, Geoffrey C. Fox. In Lease-based consistency schemes in the web environment. Future Generation Computer Systems 25 (2009) 8–19.

[2] Randal C. Burns, Robert M. Rees, Darrell D. E. Long. In IEEE 0-7695-574- 0/2000.

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

[4] Venkata Duvvuri, Prashant Shenoy and Renu Tewari. Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web.In IEEE transactions on knowledge and data engineering, vol. 15, no. 5, september/october 2003.

[5] G. Barish, K. Obraczka, World wide web caching: Trends and techniques, IEEE Communications Magazine 38 (5) (2000) 178–184.

[6] Mustafa Mat Deris, Jemal H. Abawajy, Ali Mamat, An efficient replicated data access approach for large-scale distributed systems, Future Generation Computer Systems 24 (1) (2008) 1–9.

[7] P. Cao, C. Liu, Maintaining strong cache consistency in the world wide web, in: Proc. of Third International Conference on Web Caching,1998.

[8] S Wu, Y. Chang, An active replication scheme for mobile data management, in: Proc. of the 6th International Conference on Database System for Advanced Applications, April 1999, pp. 143–150.

[9] A. Beloued, et al., Dynamic data replication and consistency in mobile environments, in: Proc. 2nd Int'l Doctoral Symp. Middleware,MDS05, vol. 114, No. 1, ACM Press, 2005, pp. 1–5.

[10] Wanlei Zhou, Li Wang, Weijia Jia, An analysis of update ordering in distributed replication systems, Future Generation Computer Systems 20 (4) (2004) 565–590.

[11] O. Babaoglu, R. Davoli, L. A. Giachini, and M. G. Baker. RELACS: A communications infrastructure for constructing reliable applications in large-scale distributed systems. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, 1995.

[12] J. Gwertzman and M. Seltzer, "World-Wide Web Cache Consistency," Proc. 1996 USENIX Technical Conf., Jan. 1996.

[13] Squid Internet Object Cache Users Guide, available online at http://squid.nlanr.net, 1997.

[14] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, "Volume Leases for Consistency in Large-Scale Systems," IEEE Trans. Knowledge and Data Eng., Jan. 1999.

[15] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, R. Tewari, Scalable consistency maintenance in content distribution networks using cooperative leases, IEEE Transactions on Knowledge and Data Engineering (2003).

[16] Hakan Grahn, Per Stenstrom, Michel Dubois, Implementation and evaluation of update-based cache protocols under relaxed memory consistency models,Future Generation Computer Systems 11 (3) (1995).

[17] S. Shah, et al., Maintaining coherency of dynamic data in cooperating repositories, in: Proc. of the 28th International Conference on Very Large Data Base, 2002.

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