Piggyback server invalidation for proxy cache coherency

Balachander Krishnamurthy[a] and Craig E. Wills[b]

[a]*AT&T Labs-Research,*
Florham Park, NJ 07932 USA
bala@research.att.com

[b]*Worcester Polytechnic Institute,*
Worcester, MA 01609 USA
cew@cs.wpi.edu

Abstract

We present a piggyback server invalidation (PSI) mechanism for maintaining stronger cache coherency in Web proxy caches while reducing overall costs. The basic idea is for servers to piggyback on a reply to a proxy client, the list of resources that have changed since the last access by the client. The proxy client invalidates cached entries on the list and can extend the lifetime of entries not on the list. This continues our prior work on piggyback cache validation (PCV) where we focused on piggybacking validation requests from the proxy cache to the server. Trace-driven simulation of PSI on two large, independent proxy log data sets, augmented with data from several server logs, shows PSI provides close to strong cache coherency while reducing the request traffic compared to existing cache coherency techniques. The best overall performance is obtained when the PSI and PCV techniques are combined. Compared to the best TTL-based policy, this hybrid policy reduces the average cost (considering response latency, request messages and bandwidth) by 7-9%, reduces the staleness ratio by 82-86%, yielding a staleness ratio of 0.001.

Keywords

Cache coherency; Proxy cache; Server invalidation

1. Introduction

Caching is widely used in the Web at the browser and proxy level. Previously, we studied piggyback cache validation (PCV) [7], a technique to improve cache coherency and reduce the cache validation traffic between proxy caches and servers. We focused on using the information available in the cache and partitioned resources on the basis of originating server. When a server is contacted again, the proxy client piggybacked a list of cached, but potentially stale, resources obtained from that server for validation. The server replied with the subset of resources that are no longer valid. PCV yielded stronger cache coherency and reduced costs by using the validations to extend the expiration time for cached resources and reduce the need for GET If-Modified-Since (IMS) requests.

In this work, we combine resource information available to servers with the piggybacking technique to create a mechanism called piggyback server invalidation (PSI). Servers partition the set of resources at a site into *volumes*, either a single site-wide volume or related subsets of resources and maintain version information for each volume. When a server receives a request from a proxy client containing the client’s last known version of the volume, it piggybacks a list of volume resources modified since the client-supplied version. The proxy client invalidates cached entries on the list and can extend the lifetime of entries not on the list. Servers maintain volume, but no proxy-specific information. While the mechanism could be used by browser clients, we focus our discussion on its use by proxy clients, where there are more cached resources.

The aim of the PCV and PSI mechanisms is the same - use piggybacking to eliminate stale entries from a proxy cache while extending the lifetime of valid entries without introducing additional network traffic. However, the mechanisms differ in their use of resource information and piggybacking. The PCV mechanism uses resource information available only to a proxy while PSI can group resources based on access patterns and modification characteristics known only to a server. The PSI mechanism does require changes to existing Web servers for implementation. The PCV mechanism piggybacks a list of resources in the proxy cache for validation, which can cause validation checks for resources that have not changed. The PSI piggybacks a list invalidated resources, which have changed at the server, but may not be cached at the proxy. For both mechanisms, this unused piggybacked information creates additional bandwidth and latency overhead.

In this work, we study the PSI approach, comparing its performance and overhead to the PCV approach and existing cache coherency techniques. The study was carried out using trace driven simulation on two large independent proxy log data sets. Proxy logs do not have the information regarding when resources change on the server and thus the number of invalidations that would be generated between client accesses. In the absence of such end-to-end logs we studied a number of representative server logs to obtain a measure of invalidations that do occur between successive client accesses and used this measure in the simulation.

The rest of this paper is organized as follows: we begin with a discussion of related work in the field of file systems and the Web in Section 2. Section 3 describes piggyback server invalidation in more detail and discusses approaches for its implementation. Section 4 presents the environment and evaluation criteria for studying various PSI-based cache coherency policies. Section 5 provides the results of this study. Section 6 summarizes the results and discusses alternative approaches.

2. Related work

Previous work on server invalidation [8], explored server invalidation by propagating resource changes to all clients that accessed a resource since its previous modification. While this was to guarantee strong cache coherency. it required the server to maintain a client list for each resource, which could become out-of-date as clients may no longer have previously accessed resources in their cache. In addition, invalidations are sent as separate messages, thus generating more network traffic. In the event of network failures, even the guarantee of strong cache coherency could be lost. The number of invalidations can be reduced by using the concept of ‘‘leases’’ where a server grants a lease whenever a client uses an GET IMS request indicating a subsequent access for the resource. This approach reduces the number of invalidation messages generated by a server at the expense of more GET IMS requests generated by clients.

In file systems, the Andrew File System [5] introduced the concept of volume validation whereby a group of files could be checked for modification. Jeff Mogul’s work on callback-based caching in NFS [10] dealt with revocation (indicating that a cached instance has changed on the disk). He also proposed *volumes* [11] as a way to partition the collection of Web resources on the basis of some shared characteristic: proximal existence on the file system (same directory), identical content types, similar rate of change etc. This proposal suggested piggybacking an invalidation of the entire volume if any volume resource changed. In contrast, our work piggybacks the list of resources that have changed in the volume.

In terms of cache coherency, client validation is another approach to ensure strong cache coherency. In this approach, the proxy treats cached resources as potentially out-of-date on each access and sends a GET IMS request to the server on each access of the resource resulting in many 304 (Not Modified) responses. A common weak consistency approach is for the proxy to adopt an adaptive time-to-live (TTL) expiration time (also called the Alex protocol [1] based on the last modified time [4]). The older the resource, the longer the time period between validations.

3. Piggyback server invalidation

In implementing the PSI mechanism servers group resources into volumes as suggested in [11]. Each volume has a unique identifier and a current version. Whenever a resource changes within a volume, the server updates the volume version and records the resource(s) that changed between the previous and current version.

Each proxy client maintains the current set of server volume identifiers and versions for resources in its cache. When a proxy needs to request a resource from a server, it looks up the current volume identifier and version for the resource and piggybacks this information as part of the request. If the volume identifier is unknown or if the proxy does not have a version for the volume, then it requests such information to be piggybacked in the reply.

In response, the server piggybacks the volume identifier, the current volume version and a list of resources from this volume that have changed between the proxy-supplied and current version. The proxy client updates its volume version, uses the list to invalidate cached entries from this volume and possibly extends the expiration time for volume resources that were not invalidated.

The basic description of the PSI mechanism leads to a number of questions for its implementation, which we explore in this work. Grouping sets of resources into volumes is a central question. The simplest approach is to group all resources at a server site into a single volume. Another approach is to group resources based on the first level prefix of the resource pathname.

Other questions concern how a proxy client should combine server invalidations with a cache validation policy. The client could do no additional validation of cache contents and depend solely on server invalidations to maintain cache coherency or use a TTL-based coherency approach to explicitly validate potentially stale entries that have not been invalidated. Another question is whether a proxy client should treat the lack of an invalidation message from a server as an implicit validation of all cached resources from that volume and extend the expiration time for these resources according to the client validation policy in effect. A proxy client could also combine the PSI mechanism with PCV to create a hybrid approach where the choice of the mechanism depends on the time since the proxy last requested invalidations for the volume and the number of cached resources in the volume.

4. Study

4.1. Proxy cache logs

We studied the relative merits of various PSI policy implementations using a trace-driven simulation with two independent proxy log data sets - one from AT&T Labs-Research [12] and one from Digital Equipment Corporation [2]. Characteristics of these client logs are summarized in Table 1. The *change ratio* value in Table 1 is defined in [3] as the ratio of new instances to total references for a resource (the fraction of references to a resource that yield a changed instance). In our work this value was determined by comparing the last modification time of successive instances of a resource if known or by looking for size changes if the time is unknown.

Table 1. Proxy log characteristics

Log	Time period	Requests (mill.)	Requests/hour	Repeat ref. ratio	Change ratio
Digital	Sep 16-22, 1996	6.41	40031	0.69	0.088
AT&T	Nov 8-25, 1996	1.11	2806	0.56	0.125

4.2. Server generated invalidations

For the PSI-based policies, the proxy logs contain information to determine if a resource in the cache is invalidated. However, the proxy logs lack information regarding when resources change on the server and thus the total number of invalidations that would be generated between successive client accesses. To compensate for the unavailable information, we studied representative server logs to obtain the number of resource modifications that occur at a server for each minute interval between accesses by a client. This number is used in the proxy log simulation in conjunction with the time between successive accesses to a server to determine an *expected* number of invalidations that would be piggybacked on the reply. This expected number is used to estimate the bandwidth overhead of the PSI mechanism.

We analyzed the number of server generated invalidations for a variety of relatively recent (all logs include 1997 data) server logs. We obtained server logs from Sun Microsystems, the Apache HTTP Server Project, the National Center for Supercomputer Applications (NCSA) and the Victoria University of Wellington, New Zealand School of Mathematical and Computing Sciences (VUW). Characteristics of the server logs are summarized in Table 2.

Table 2. Server log characteristics

Log (days)	Requests (mill.)	Requests/ hour	Repeat ref. ratio	Change ratio	Invalidations /min. interval	Invocation./ min. interval (top-level prefix)
Sun (9)	1.20	55606	0.23	0.012	2.12	0.09
Apache (49)	2.93	2493	0.37	0.078	0.51	0.07
NCSA (10)	0.38	1568	0.11	0.010	0.01	0.00
VUW (40)	0.36	378	0.53	0.051	0.06	0.01
VUWlmod (40)	0.36	378	0.53	0.077	0.10	0.02

Analysis of the server logs shows a near direct correlation between the number of resources that changed in a server volume and the interval since a client last accessed the volume. The second-to-last column in Table 2 captures this relationship by showing the number of invalidations that occur in a volume relative to the number of minutes between accesses to the volume from the same client. The numbers in this column assume that all resources at a server are treated as a single volume. The last column in the table shows the number of invalidations if the resources at a server are grouped into volumes according to the top-level prefix of the resource pathname. The number of invalidations drops, particularly for the Sun logs.

We used these results to determine a representative number of invalidations per minute of access interval for servers in our simulation study. by considering how resource changes are detected in the server logs. In all but one case of the server logs we obtained, last modification time information was not available. Thus, resource changes for the server logs are detected when the size changes between successive accesses of a resource and this change is not the result of an aborted client connection (when the size returned is a multiple of 8K bytes). Obviously this approach for detecting changes misses the case where a resource is modified, but unchanged in size.

The VUW server logs were augmented to include last modification time information for the files. The results using last modification times to detect resource changes are shown in the last row of Table 2. The change ratio values for the two VUW results show that about two-thirds of the changes detected using the modification times were detected using size-only changes. Moreover, analysis of the AT&T and Digital proxy logs shows that 80% and 95% of the resource changes detected using last modification times also indicate a change in resource size. These comparisons showed that to use only the resource size to detect changes misses up to a third of the actual changes. Further analysis of the VUW site shows that only another 8% of the resources were modified in the file system between the their last entry in the server log and the end of the study.

Based on these results, our simulation uses a base value of 2.0 invalidations per each minute interval for site-wide volumes and 0.1 invalidations per minute for volumes determined by the top-level prefix of a resource. These values are closest to the Sun server logs, but not as high as if we adjusted the Sun values because it uses only size-based changes. As part of our presentation of results we show the sensitivity of these results to the number of generated invalidations.

4.3. Baseline policies and evaluation criteria

As part of the study, four baseline cache coherency policies were established for comparison with PSI-based policies. The pcvadapt policy implements piggyback cache validation where on a request to a server, a proxy client piggybacks a list (maximum size of 50 used in study) of cached, but potentially stale resources from that server for validation [7]. The pcvadapt policy uses an adaptive TTL expiration time based on a fraction (0.1 used in study) of the age of the resource with a maximum expiration time of one hour. The ttladapt policy uses an adaptive time-to-live expiration time based on resource age with no piggybacking. The upper bound for a resource expiration time is fixed at one day. The alwaysvalidate policy generates a GET IMS request for every access of a cached resource ensuring strong coherency, but causes a request to the server for every resource access. The nevervalidate policy has an infinite expiration time; cached resources are never validated. This policy minimizes costs by always using the cached copy of a resource, but results in the most use of stale copies.

These baseline policies are compared with PSI-based policies using the costs for three evaluation criteria: response latency, bandwidth and number of requests [7]. We use a normalized cost model for each of the three evaluation criteria where each of these costs is defined to be 0.0 if a GET request can be retrieved from the proxy cache; 1.0 for an average GET request to a server with full resource reply (status 200). In the Digital logs, the average values are 12279 bytes of bandwidth usage (includes contents and headers), 3.5 seconds of latency and one request for an average retrieval. In the AT&T logs, the average values are 8822 bytes of bandwidth, 2.5 seconds of latency and one request.

The cost of a Validate (GET IMS) request, which returns a validation of the current cache copy (304 response), is computed relative to the cost of a full GET request for the log. This request is just as expensive as a full GET request in terms of requests, of intermediate cost (0.36 Digital, 0.12 AT&T) in terms of response latency and of little cost (0.03 Digital, 0.04 AT&T) in terms of bandwidth. A fourth evaluation criterion, the average of the costs for the other three criterion, is used as a composite criterion.

The total cost for each criterion for a simulation run using a cache coherency policy is computed by multiplying each normalized cost by the relative proportion of requests requiring GET, Validate, and cache retrieval actions.

In addition to costs, staleness is evaluated as the ratio of the known stale cache hits divided by the number of total requests (both serviced from cache and retrieved from the server). This definition deflates ratios in comparison to the traditional measure of stale in-cache hits, but allows fairer comparison of the policies, which differ in their in-cache ratios.

4.4. PSI implementation costs

The detection of resource changes at the server is a potentially significant cost in implementing the PSI mechanism. One approach is for the server to maintain a list of resources that have been recently accessed, and use an external mechanism, such as COLA [6], to notify the server when any of these resources change. The cost of detection could be controlled by limiting the list to resources accessed over a fixed time such as the last day and tagging dynamically generated resources as uncacheable. The presence of this detection mechanism was assumed in our simulation results. Its overhead is estimated by determining the number of resources on the recently accessed list that change, each of which causes COLA to incur two checks on the file modification time. As a measure of this cost, the Sun server logs indicate a rate of 18 changes per minute or 0.04 checks per request. No direct cost was charged for this overhead in the simulation. An alternate approach for detecting changes is discussed in Section 6.

Charges are incurred in the simulation for the additional piggybacked bytes sent on the network for the PSI mechanism. These charges are for the proxy client to request invalidations by sending a volume identifier and version along with the piggybacked server reply containing the volume identifier, version and list of invalidated resources. In the simulation, 20 bytes is added for each piggybacked field and 25 bytes for each piggybacked resource name. These charges are conservative and techniques such as prefix sharing can reduce them. In addition a byte latency of 0.25 ms, derived from our data, is added to the response latency for each additional byte (this is consistent with [9]).

5. Results

We created scenarios to test the implementation approaches outlined in Section 3 for studying PSI-based cache coherency policies. We used a standard LRU cache replacement policy for all simulations and varied the proxy cache size from 1MB to 8GB. In order to restrict proxy client requests for invalidations to situations in which more useful invalidation information would be received, invalidations are not requested if the volume of the current request has been accessed within the last minute. Results show that this restriction limits invalidation requests and replies to 10-25% (depending on simulation parameters and policy) of total GET requests. Similarly, requests for invalidations are not generated if the last access to a volume has been over 24 hours.

5.1. PSI with different proxy cache coherency strategies

Our first scenario tested the combination of the PSI mechanism with different proxy cache coherency policies. Three variations were tested (all use a single site-wide server volume):

- nevervalidate-psi - This policy implements piggyback server invalidation with the proxy client taking no other action for cache coherency. A previously cached resource is always used unless it has been invalidated by the PSI mechanism.

- ttladapt-psi - This policy implements PSI to invalidate cached resources, but in addition the proxy cache uses the ttladapt policy to maintain additional coherency.

- ttladapt-psicv - This policy implements PSI to invalidate cached resources and uses the ttladapt policy at the proxy client to maintain additional coherency. In addition, the proxy client treats the lack of an invalidation from a server as an implicit validation of cached resources from that server volume and extends the expiration times according to the ttladapt policy.

Figure 1 shows the average cost for these three policies along with the four baseline policies. As found in [7], the performance differences between policies is primarily a function of the proportion of Validate requests. Graphs for the request, response latency and bandwidth costs (not shown) show similar shapes with more distinction between the policies for request costs and less for the other two costs. As expected, the alwaysvalidate policy performs the worst, while the nevervalidate policies incur the least costs because they never validate the cache contents. The ttladapt-psicv policy incurs the least cost of the PSI-based policies doing proxy validation - comparable to pcvadapt. Figure 2 shows the staleness for the policies under study. The PSI-based policies combined with ttladapt perform the best, degrading a little with a bigger cache as more stale resources reside in cache without the proxy receiving a piggybacked invalidation. The nevervalidate-psi policy reaches a staleness ratio of 0.03 for an 8GB cache. As a benchmark, the nevervalidate policy (not shown) reaches a staleness ratio of 0.07 for an 8GB cache.

The high staleness ratio of the nevervalidate-psi policy indicates the PSI mechanism needs to be augmented with client validation. Ttladapt-psicv and ttladapt-psi are the two best PSI-based policies with the former incurring lower costs and the latter lower staleness. Selecting ttladapt-psicv for comparison, it reduces the average cost by 3% and staleness by 44% in relation to the ttladapt policy, the best existing policy, for an 8GB cache. It has a 1% higher average cost and a 29% higher staleness than the pcvadapt policy.

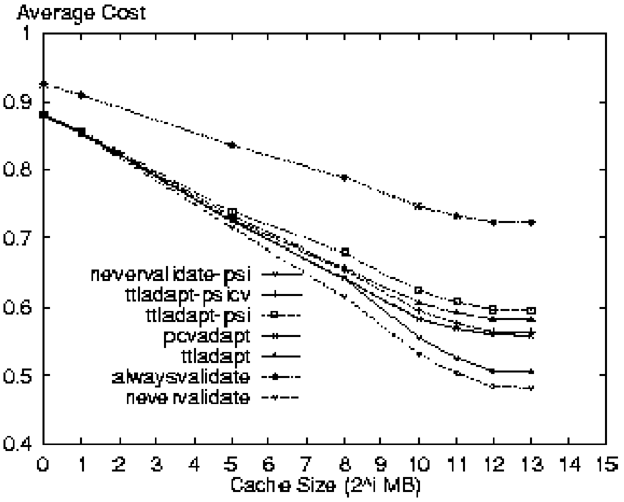

Fig. 1. Average cost versus cache size for AT&T logs using PSI with different proxy cache coherency strategies.

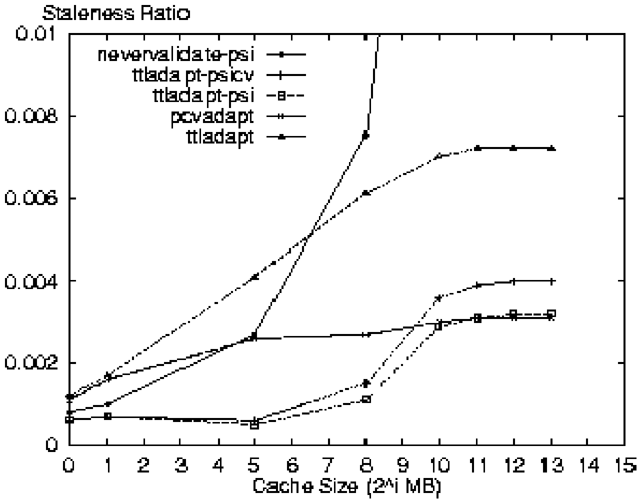

Fig. 2. Staleness ratio versus cache size for AT&T logs using PSI with different proxy cache coherency strategies.

The same tests were performed on the Digital logs with similar results. The nevervalidate policy results in a staleness ratio of 0.04 for an 8GB cache with the nevervalidate-psi policy yielding a much lower staleness ratio (less than 0.01) than for the AT&T logs because of the lower change ratio in the Digital logs. For an 8GB cache, the ttladapt-psicv policy reduces the average cost by 6% and staleness by 75% over ttladapt. The policy has a 1% increase in average cost and a 30% decrease in staleness in comparison to pcvadapt. The policy costs are lower for the Digital logs because fewer Validate requests are generated.

5.2. PSI with different volume strategies

The next scenario tested the PSI mechanism using the top-level prefix to group resources into volumes. Using multiple volumes at a server reduces the number of server invalidations as shown in the last column of Table 2. Two policies were tested:

- ttladapt-psicv-cur - This policy implements PSI to invalidate cached resources in the volume of the *current* resource being requested. Otherwise, the proxy client uses the ttladapt policy with the lack of invalidation for cached resources in the same volume as the current resource treated as an implicit validation to extend the expiration time.

- ttladapt-psicv-cli - This policy implements PSI to invalidate cached resources by including the identifier and version for *all* server volumes currently containing cached resources at the proxy *client*. This policy potentially invalidates all cached resources from the server, but increases the overhead of sending volume identifier and version information.

Figures 3 and 4 show the performance of these policies along with ttladapt-psicv, which uses site-wide volumes, and the baseline policies. The results show that the ttladapt-psicv-cur policy provides comparable performance to ttladapt-psicv in reducing the average cost by 4% and the staleness by 38% when compared to the ttladapt policy for an 8GB cache. The ttladapt-psicv-cli policy performs worse as the increased overhead for volume invalidations does not result in significantly better staleness results. The Digital logs contain anonymous entries and cannot be used to test alternate volume strategies.

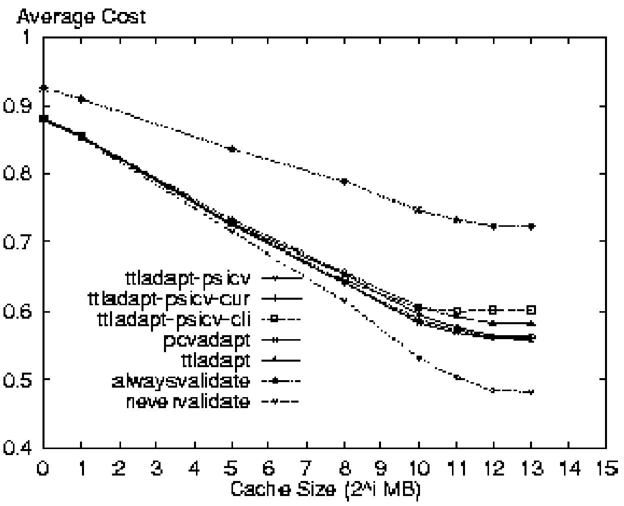

Fig. 3. Average cost versus cache size for AT&T logs using PSI with different volume strategies.

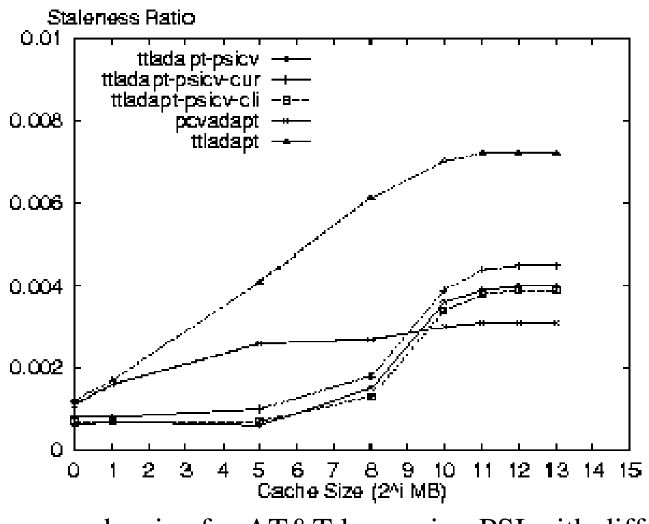

Fig. 4. Staleness ratio versus cache size for AT&T logs using PSI with different volume strategies.

5.3. PSI with piggyback cache validation

The third scenario we investigated looks at the potential of combining server invalidation with proxy client validation to maintain proxy cache coherency. Different variations were tested using site-wide volumes with the best variation as follows:

- pcvadapt-psicv-hy - This policy is a hybrid of PSI and PCV policies where the proxy client requests invalidations to be sent if the time since the last access to a volume is less than one hour (maximum resource expiration time), otherwise the proxy client uses the pcvadapt policy to do client validation of cached resources.

The simulation results show that the hybrid policy reduces the average cost by 3% and the staleness ratio by 68% in comparison to the pcvadapt policy for an 8GB cache using the AT&T logs. In absolute terms, the staleness ratio is 0.001 - the policy is close to strongly coherent. In comparison to ttladapt, average costs are reduced 7% and the staleness ratio by 86%. The Digital logs show similar results with a 2% average cost and 50% staleness reduction compared to pcvadapt, and a 9% cost and 82% staleness reduction compared to the ttladapt policy. Again the staleness ratio for an 8GB cache is 0.001.

5.4. Sensitivity analysis

The final scenario we tested was the sensitivity of the results to changes in the number of invalidations per minute interval. The base values we used were 2.0 invalidations for each minute interval with site-wide volumes and 0.1 invalidations per minute for volumes based on the top-level prefix.

For an 8GB cache with the AT&T logs, the costs for the ttladapt-psicv policy increase by 2% if the number of invalidations doubles to 4.0 for each minute interval and increase by 6% if the number doubles again to 8.0. In contrast, the average cost of the pcvadapt-psicv-hy policy increases by less than 1% when the number of invalidations is increased. Similar analysis for top-level volumes also yielded virtually no change for generated invalidations values of 0.2 and 0.4. The insensitivity of the latter two results suggests PSI is a cost-stable mechanism when there is a tight limit on the interval between invalidation requests or related resources are grouped together.

6. Summary and future work

The results of this work show that the piggyback server invalidation mechanism can be combined with a proxy TTL-based coherency policy to yield close to strong cache coherency at a lower cost than current cache coherency policies. Cache coherency is improved by invalidating stale resources and costs are reduced by extending the expiration time of cached resources to reduce the number of GET IMS requests. In the simplest implementation, all the resources at a site are treated as a single site-wide volume, but this approach can generate a large number of piggybacked invalidations from sites with frequently changing resources. This sensitivity is sharply reduced when resources are grouped into volumes, such as using the top-level prefix of a resource pathname. The best results using the PSI mechanism come when it is combined with piggyback cache validation. This hybrid policy provides nearly strong cache coherency with a staleness ratio of 0.001 and a 6-9% reduction in overall costs in comparison to the best TTL-based policy. In contrast to the PCV mechanism, the PSI mechanism does require changes to existing Web servers for implementation.

Our work thus far on the PSI mechanism suggests a number of directions for future work. One issue with the PSI mechanism is how servers detect changes to resources. An interesting alternate approach for detecting changes is for a server to store the last modification time for each resource on its list of resources that have recently been accessed. Rather than use an external mechanism for detection, it could compare the last modification times of successive resource requests to detect changes. This approach introduces potential delay between when a resource change occurs and the server discovers it, but avoids an external mechanism.

Other techniques to construct volumes could also be studied. These techniques include looking at the rate of change of resources and grouping resources that change with similar frequency into a volume, thus maximizing the usefulness of the invalidation information. From related work [3], we know that there is correlation between content type and rate of change of pages; for example, image types change less often than text pages. This correlation argues for a grouping based on content type. Construction of volumes with tighter relationships between the resource elements could lead to exploration of invalidating entire volumes rather than selective resources within such volumes as suggested in [11]. This approach reduces the amount of piggybacked traffic, but could result in more GET IMS requests from a client if valid cached resources are marked as potentially stale.

Acknowledgments

We thank Jeff Mogul for discussions regarding volumes early on and for comments on a draft version of the paper. Jennifer Rexford's detailed comments helped with the presentation. We also thank Anja Feldmann, Pawan Goyal and Misha Rabinovich for reviewing a draft version of the paper. This work would be impossible but for the various logs provided by Digital Equipment Corporation, Sun Microsystems, AT&T, Victoria University of Wellington, the National Center for Supercomputer Applications and the Apache Group. We thank them all.

References

[1] V. Cate, Alex - a global filesystem, in: *Proceedings of the USENIX File System Workshop,* USENIX Association, May 1992, pp. 1-12.

[2] Digital Equipment Corporation, Proxy cache log traces, September 1996, `ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html`.

[3] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul, Rate of change and other metrics: a live study of the World Wide Web, in: *Symposium on Internet Technology and Systems*, USENIX Association, December 1997, `http://www.usenix.org/publications/library/proceedings/usits97/douglis_rate.html`

[4] J. Gwertzman and M. Seltzer, World-Wide Web cache consistency, in: *Proceedings of the USENIX Technical Conference*, USENIX Association, January 1996, pp. 141-152, `http://www.usenix.org/publications/library/proceedings/sd96/seltzer.html`.

[5] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, and R.N. Sidebotham, Scale and performance in a distributed file system, *ACM Transactions on Computer Systems*, 6(1): 55-81, February 1988.

[6] E. Krell and B. Krishnamurthy, COLA: customized overlaying, in: *USENIX San Francisco Winter 1992 Conference Proceedings*, 1992, pp. 3-7.

[7] B. Krishnamurthy and C.E. Wills, Study of piggyback cache validation for proxy caches in the World Wide Web, in: *Symposium on Internet Technology and Systems*, USENIX Association, December 1997, http://www.usenix.org/publications/library/proceedings/usits97/krishnamurthy.htm

[8] C. Liu and P. Cao, Maintaining strong cache consistency in the World-Wide Web, in: *Proc. of the 17th IEEE International Conference on Distributed Computing Systems*, May 1997, http://www.cs.wisc.edu/~cao/papers/icache.html

[9] S. Manley and M. Seltzer, Web facts and fantasy, in: *Symposium on Internet Technology and Systems*, USENIX Association, December 1997, http://www.usenix.org/publications/library/proceedings/usits97/manley.html.

[10] J. Mogul, Recovery in spritely NFS, *Computing Systems*, 7(2): 201-262, Spring 1994, http://www.research.digital.com/wrl/techreports/abstracts/93.2.html.

[11] J. Mogul, An alternative to explicit revocation? January 1996, http://weeble.lut.ac.uk/lists/http-caching/0045.html.

[12] J.C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, Potential benefits of delta-encoding and data compression for HTTP, in: *ACM SIGCOMM'97 Conference*, September 1997, http://www.acm.org/sigcomm/sigcomm97/papers/p156.html.

Vitae

Balachander Krishnamurthy is a Senior Member of Technical Staff at AT&T Labs-Research in Florham Park, New Jersey, USA.

Craig E. Wills is an associate professor in the Computer Science Department at Worcester Polytechnic Institute. His research interests include distributed computing, operating systems, networking and user interfaces.