In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, Austin, TX, June 1999.

Proxy Cache Coherency and Replacement—Towards a More Complete Picture

Balachander Krishnamurthy
AT&T Labs–Research
180 Park Ave
Florham Park, NJ 07932 USA
bala@research.att.com

Craig E. Wills
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609 USA
cew@cs.wpi.edu

Abstract

This work studies the interaction of Web proxy cache coherency and replacement policies using trace-driven simulations. We specifically examine the relative importance of each type of policy in affecting the overall costs, the potential of incorporating coherency issues in cache replacement and the inclusion of additional factors such as frequency of resource use in replacement and coherency policies.

The results show that the cache replacement policy in use is the primary cost determinant for relatively small caches, while the cache coherency policy is the determinant for larger caches. Incorporating cache coherency issues in cache replacement policies yields little improvement in overall performance. The use of access frequency in cache replacement, along with temporal locality and size information, results in a simple and better performing policy than found in previously published work. Combining this new replacement policy with the best piggyback-based cache coherency policy results in a 4.5% decrease in costs and 89% reduction in staleness ratio when compared to policy combinations in current use. Preliminary work indicates that cache replacement and coherency policies continue to affect costs in the presence of HTTP protocol enhancements such as persistent connections.

1. Introduction

The performance of Web proxy caching mechanisms is an active area of research. Many studies have examined policies for cache replacement and cache coherency of Web resources. However, these studies have primarily examined them individually and have not taken into account the combined effects of policies for each issue. In this paper we examine a more complete picture of Web proxy performance that explores whether coherency and replacement decisions should affect each other and the impact of these decisions on the overall performance costs (server requests, network latency, bandwidth).

This work builds on previous work we have done on cache coherency. We studied piggyback cache validation (PCV) [10], 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 the origin server. When a server is contacted again to obtain a resource, the proxy client piggybacks a list of cached, but potentially stale, resources obtained from that server for validation. The server replies with the requested resource and the list of cached resources that are no longer valid.

Later we studied an alternate approach, called piggyback server invalidation (PSI) [11], where servers partition the set of resources at a site into volumes, either a single site-wide volume or related subsets of resources. Each volume has its own identifier and current version. 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 volumes, but no proxy-specific information.

Compared to existing policies, both the PCV and PSI mechanisms yielded stronger cache coherency and reduced performance costs, by using the piggybacked validation/invalidation information to extend the expiration time for cached resources and reducing the need for GET If-Modified-Since (IMS) validation requests. The best overall performance was obtained when the PSI and PCV techniques are combined to create a hybrid approach where the choice of the mechanism depends on the time since the proxy last requested invalidations for the volume [11]. If the time is small (e.g. an hour) then the PSI mechanism is used, while for longer gaps the PCV mechanism is used to explicitly validate cache contents. The rationale is that for short gaps, the number of invalidations sent with PSI is relatively small, but as the gap grows larger, the overhead for sending invalidations will be larger than the overhead for requesting validations.

Our prior work on cache coherency assumed the use of a least recently used (LRU) policy for cache replacement. In this work, we build on the cache replacement policy approach of Cao and Irani [4]. Their work proposes a new algorithm called GreedyDual-Size, which combines temporal locality, size, and other cost information. Their algorithm assigns a *cost/size* value to each cached resource. In the simplest case the cost is set to 1 to maximize the hit ratio, but other costs such as latency, network hops and packets are explored. The algorithm takes into account recency for a resource by inflating the *cost/size* value for an accessed resource by the least value of currently cached resources. The algorithm is both simple to implement with a priority queue and yields superior performance relative to previously proposed replacement policies.

In this work, we use these separate research results as a base to study the joint impact of cache coherency and replacement using trace-driven simulations. Our approach is to not use simple performance measures such as cache hit ratio, but to use a cost model [10, 11] that takes into account server requests, response latency, and network bandwidth for combinations of coherency and replacement policies. By also considering the ratio of stale resources served from cache (staleness ratio) we can find policy combinations that provide the most up-to-date resources from cache at the least cost.

The ability of proxy caches to reduce user latency, network bandwidth and server load has been widely studied. Recent results have questioned the effectiveness of caching due to dynamic content, the increased presence of cookies and the impact of persistent connections [2]. However, these results still show cache hit rates of around 35% and recent results show that most responses based on cookies can be reused [19]. In addition, techniques such as delta-encoding [14] and active caches [3] show promise in allowing resources that change frequently, but predictably, to be cached on a cost-effective basis.

The rest of this paper begins with a discussion of related work in cache replacement and coherency in Section 2. Section 3 describes the policies we investigate in this work based on different issues. Section 4 presents the environment and evaluation criteria for studying the combined coherency and replacement policies. Section 5 discusses the results of our study. Sections 6 and 7 discuss future directions of the work and summarize our findings to date.

2. Related Work

Much research has been done on cache replacement and cache coherency policies for proxy caches. Cao and Irani [4], as well as other work [1, 15, 18], have studied the effect of cache replacement policies independent of cache coherency. Likewise, prior work on cache coherency, including our own work [10, 11], has not taken into account how these policies interact with cache replacement policies [8, 12, 20].

The interaction between these two issues has been examined in [16], which builds on [15], to account for delays in retrieving a resource and the delay to validate it. The enhanced algorithm is better than LRU and LRU-MIN [1] in terms of savings in delay and in staleness ratio. The results, while encouraging, are not compared with better cache replacement algorithms in [4], are based on a relatively small trace, and the algorithm requires bookkeeping information for each resource and for metadata about resources evicted from the cache.

3. Issues

We examine the interaction of cache replacement and cache coherency policies from a number of perspectives. We use a LRU cache replacement policy and an adaptive TTL (*ttladapt*) cache coherency policy as baseline policies for our work. LRU is commonly used for comparison in prior work on cache replacement. The *ttladapt* policy uses an adaptive TTL based on a fraction of the resource age. Cached resources that exceed this TTL in cache must be validated using a GET IMS request. LRU and *ttladapt* are both used in the popular Squid proxy cache [17]. We can examine a variety of policies using a uniform and comprehensive evaluation model, which takes into account server requests, response latency and bandwidth along with the ratio of stale resources served from cache.

3.1. Relative Costs of Cache Coherency and Replacement

We begin by looking at the relative importance of coherency and replacement for overall costs. Cao and Irani show that the GreedyDual-Size-based policies increase the hit ratio relative to LRU and other previously proposed replacement policies and result in lower costs (as shown later in Section 4.2). Our previous work using piggybacking for cache coherency not only improved staleness results compared to the *ttladapt* policy, but also lowered costs by reducing the number of validation messages sent by a proxy cache. By studying the combination of replacement and coherency policies in a single model, we can answer which of these "better" policies has a greater impact on reducing overall costs.

We consider two cache replacement policies, *lru* and a better policy *gd1* (GreedyDual-Size algorithm with a base value of *1/size*), and two cache coherency policies, *ttladapt* and a better policy *hybrid* (combination of PCV and PSI policies). The four combinations of these policies are tested under a range of cache sizes. While we expect that the better replacement policy paired with the better coherency policy to yield the best performance, it is interesting to compare the combinations with only one of the better policies to see if costs are lowered across the entire range of cache sizes. This comparison will identify which, if any, of the improvements is the primary factor in determining costs.

3.2. Influence of Cache Coherency on Cache Replacement

Rather than treating these issues separately, we now examine if overall performance can be improved by considering coherency issues as part of the cache replacement decision. Normally, user access patterns primarily affect cache replacement decisions while resource characteristics affect cache coherency decisions. However, it is reasonable to consider replacing cached resources that have expired or are close to expiring because their next access will result in a validation message. To improve the staleness ratio, a cache replacement policy could be designed to account for the frequency of change for different types of resources.

Incorporating rate and cost of coherency validation checks was studied in [16]. The Harvest proxy cache [5], which does not have the capability to make GET IMS requests for expired resources, first removes cached resources that have exceeded their expiration time and then uses a LRU policy for unexpired resources. One study using Harvest found that this policy yielded undesirable effects on the cache hit ratio by replacing resources that were accessed again soon in the future [12]. In fact, Squid, a successor of Harvest, does not consider the expiration time in its LRU cache replacement policy [17].

We investigated this issue with two variations of the GreedyDual-Size algorithm that each takes into account coherency information. We broaden the use of cost in the GreedyDual-Size description to one of importance to the cache. Our first policy, *gdlifetime*, uses the remaining lifetime of a cached object in cache as part of the base value (*lifetime/size*) to lower the priority of cached resources about to expire. The second variation uses the observation that different types of resources change at different rates [7]. Specifically, HTML and text resources change more frequently than image resources. The *gdtype* cache replacement policy tests this difference by using the standard value of *1/size* for HTML (text) resources while doubling the value (*2/size*) for other resource types thus increasing the priority for resources that are less likely to change.

3.3. Consideration of Resource Access Frequency

The frequency with which a resource is accessed has been investigated in cache replacement policies. Williams, et al. investigated a Least Frequently Used (LFU) policy based on frequency count for cached resources [18], while Scheuermann, et al. calculated the frequency of access as part of a cost model [15]. Considering frequency count alone resulted in mixed performance, but consideration as part of the GreedyDual-Size algorithm may be beneficial; the *gdlfu* policy with *access_count/size* as its base value, does this.

We also consider access frequency counts for cached resources in cache coherency policies. Using the piggyback cache validation policy as a base we consider two variations to reduce the frequency at which less used, but cached resources are validated. The base *pcvadapt* policy uses an adaptive TTL with a relatively small maximum TTL of one hour because the validations are often piggybacked. To reduce the need for validations of new resources, we investigate the *pcvfreq* policy, which uses an adaptive TTL, but initially sets the maximum TTL to one day for all resources. Once a cached resource is re-used, its maximum TTL is reduced to one hour. The goal is to reduce costs while not substantially increasing staleness. The second variation *pcvcount*, has the same goal of reducing costs, but limits the number of piggyback validation checks for a resource to its access count; thus stopping validation checks when the cached resource is not being used.

3.4. Coherency and Replacement Policies

The coherency and replacement policies studied in our work are summarized below along with the parameters used. Coherency policies:

- pcvadapt—piggyback cache validation with an adaptive threshold multiplied by the age of the resource to determine freshness. A threshold of 0.1 is used with a maximum TTL of one hour.
- ttladapt—adaptive TTL policy using a threshold of 0.1 and a maximum TTL of one day.
- hybrid—combination of *pcvadapt* and piggyback server invalidation where the proxy requests invalidations if the elapsed time since the last request to the server is less than one hour and uses *pcvadapt* for larger elapsed times.
- pcvfreq—same as *pcvadapt*, but initially set the maximum TTL to one day for all resources. Once a cached resource is re-used, reduce its maximum TTL to one hour.

- pcvcount—same as *pcvadapt*, but limit the number of piggyback validation checks for a resource to its access count.

Cache replacement policies studied:

- lru—replace the least recently used resource in cache.
- gd1—GreedyDual-Size algorithm using a base value for a resource of *1/size*.
- gdlatency—GreedyDual-Size algorithm using a base value for a resource of *latency/size* where latency is the measured delay for the last retrieval of the resource.
- gdlifetime—GreedyDual-Size algorithm using a base value for a resource of *lifetime/size* where lifetime is the remaining cache lifetime for a resource.
- gdtype—same as *gd1*, but use a numerator of two for non-HTML resources, which typically change less frequently.
- gdlfu—GreedyDual-Size algorithm using a base value for a resource of *access_count/size* where access_count is the number of cache hits for this resource.

4. Study

We constructed a trace-driven simulation to study these policies using two client traces to test all cache coherency/cache replacement policy combinations. We used two sets of client traces—one from AT&T Labs–Research [14] and one from Digital Equipment Corporation [6]. More details about these traces are available in [10]. Both of these traces are distinguished from many proxy logs in the public domain in that they contain last modification time (lmodtime) information so we can determine if server resources actually change. This information is needed to accurately determine cost and staleness performance.

In our earlier work [10, 11], resources that were either generated dynamically through the Common Gateway Interface (CGI) or did not include an lmodtime were treated as cachable with a default expiration time. However, if the last modification time is unknown then a proxy cache such as Squid sets it to the time of the request for purposes of calculating an expiration time. Thus, the cached copy will expire on the next access. If the proxy cache uses a GET IMS for this cached copy with the assumed lmodtime then it is unlikely that the origin server would ever return a 304 response meaning the cached copy is valid. Consequently, in this work we treat such resources as uncachable. The resources account for 23% of the requests in the Digital traces and 27% of the requests in AT&T traces.

4.1. Evaluation Criteria

All combinations of cache coherency and replacement policies are compared using the costs for three evaluation criteria: response latency, bandwidth and number of requests. Consideration of these criteria gives us a more complete evaluation model than previous studies, which used hit rates or weighted hit rates. 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 traces, 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 traces, the average values are 8822 bytes of bandwidth, 2.5 seconds of latency and one request. See [10] for more details.

The cost of a Validate (GET IMS) request, which returns a validation of the currently cached copy (304 response), is computed relative to the cost of a full GET request for the trace. 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. These validation costs are computed based upon the cost of an average resource retrieval and validation in each trace.

The total cost for each criterion for a simulation run combining a cache coherency and replacement combination is computed by multiplying each normalized cost by the relative proportion of requests requiring GET, Validate, and cache retrieval actions. Each of these costs is important in evaluating various policy combinations. As means to summarize these costs we use the average of the costs as a composite criterion in comparing the various policy combinations.

In addition to costs, the staleness ratio 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.

4.2. Baseline Policies

As background to our study we present the results of using this cost model with cache replacement policies previously proposed. Figure 1 shows the average cost of the *lru.ta* (combination of *lru* and *ttladapt*) combination policy along with two cache replacement policies proposed by Cao and Irani paired with the *ttladapt* coherency policy. Results are for the Digital traces, which are primarily used in our study with results from the AT&T traces given as appropriate.

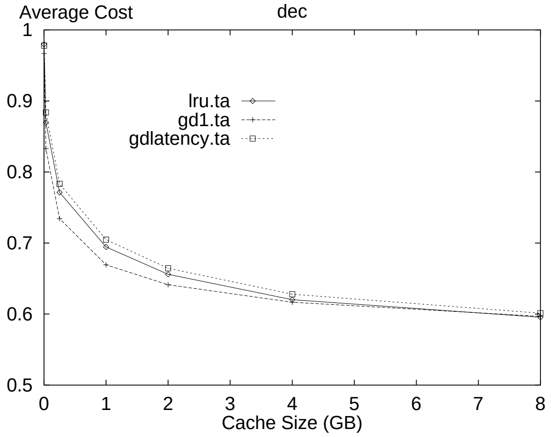

Figure 1. Average Cost versus Cache Size for Digital Traces for Baseline Cache Replacement Policies

The *gd1.ta* combination uses the GreedyDual-Size algorithm with *1/size* as the base value of a resource. The *gdlatency.ta* combination uses the GreedyDual-Size algorithm with *latency/size* as the base resource value. The cost results shown are similar to the hit ratio results found in [4] where the *gd1.ta* combination results in the least average costs. In terms of specific costs, the *gd1.ta* combination has the least cost for all three criteria, ranging from the most reduction in cost for server requests to only a small reduction in cost for bandwidth. As a by-product of keeping more requested resources in the cache, the *gd1.ta* combination has the highest staleness ratio. The *gdlatency.ta* combination has the lowest staleness ratio, but as shown in Figure 1, results in the highest costs (although it is better than *lru.ta*, but not *gd1.ta* for response costs). Cao and Irani found much variation in the latency times for the same resource indicating that it may not be a consistent predictor. Their results did not improve when they used the average latency over successive retrievals. We did not test this policy in our work. Results obtained from the AT&T traces show less differences in the costs, but the same ordering for the policies.

5. Results

Using the cost model introduced in Section 4, the following discussion presents results from studying the policies described in Section 3. The primary focus is on how various combinations of coherency and replacement policies affect costs, particularly average costs. Staleness ratio is also presented when significant in comparing different policies.

5.1. Relative Costs of Cache Coherency and Replacement

Figure 2 shows the average cost of each of the four combinations using the *lru* and *gd1* cache replacement policies combined with the *ttladapt* (*ta*) and *hybrid* coherency policies. The results show that the *lru.ta* combination results in the highest cost while the *gd1.hybrid* combination has the lowest costs. For an 8 GB cache with the Digital traces, the cost savings is 4.0% with an 89% reduction in staleness ratio (ratio is 0.0053 for *lru.ta* and 0.0006 for *gd1.hybrid*) indicating that better replacement and coherency policies in combination can result in both cost and staleness improvements.

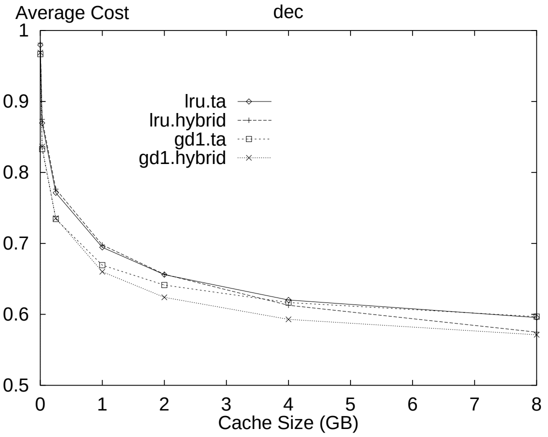

Figure 2. Average Cost versus Cache Size for Digital Traces for Four Policy Combinations

Also of interest are the comparisons between combination policies with a single better policy. Comparing *lru.hybrid* (where the better coherency policy *hybrid* is used) with *gd1.ta* (where the better replacement policy *gd1* is used) we see that for smaller caches the combination with the better cache replacement policy performs best, but for larger caches the combination with the better cache coherency policy yields the least costs. The cross-over point occurs at 3.6 GB in the Digital traces, which is 19% of the maximum cache size needed (19.0 GB). In the AT&T traces, similar results occur with a cross-over point at 0.7 GB, which is 24% of the maximum cache size needed (3.0 GB). These results are reasonably consistent and indicate that for caches where the cache space is less than 15-25% of the total cache needed, the cache replacement policy primarily determines the costs. For caches operating in configurations with larger amounts of cache space, the cache coherency policy in effect is the primary determinant in reducing overall costs.

These results, showing less difference for cache replacement policies for larger cache sizes, are consistent with prior work that shows little difference between policies beyond 5-10% of the total cache needed [4, 18]. The results show that as the cache grows larger, beyond the 15-25% crossover point of the total cache needed, the number of validation requests generated for the large number of cached resources becomes the most significant differentiating cost.

5.2. Influence of Cache Coherency on Cache Replacement

The second issue we investigated was to see if better overall results could be obtained by taking coherency into consideration when making replacement decisions. Two policies were tested—*gdlifetime*, which uses the lifetime of the cached resource in the GreedyDual-Size policy; and *gdtype*, which uses knowledge that HTML and text resources change more frequently than other resources. The average cost results for these two policies and two baseline policies are shown in Figure 3 using the *ttladapt* coherency policy for all.

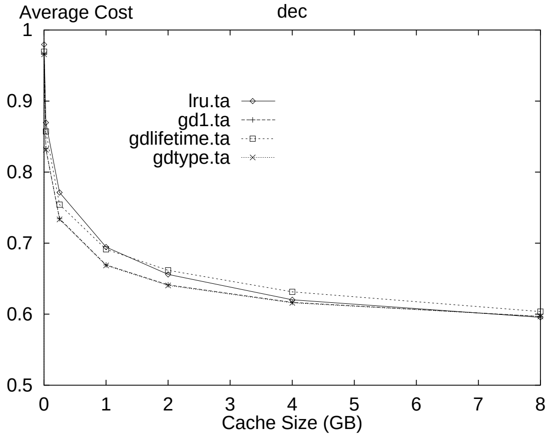

Figure 3. Average Cost versus Cache Size for Digital Traces for Replacement Policies Incorporating Coherency Issues

The results indicate that the *gdlifetime* policy overemphasizes the influence of the remaining expiration time and increases overall costs through more cache misses. The *gdtype* policy performs slightly better than the *gd1* policy on costs and staleness ratio (not shown). The improvement is less than 1% for both measures, but indicates that appropriate inclusion of coherency characteristics can have a small positive effect on performance.

5.3. Consideration of Resource Access Frequency

The next issue we investigated was the potential of using resource access frequency in the cache replacement and coherency decisions. We first examined its impact on cache coherency by including resource count information in the *pcvadapt* policy. The results did not yield improvements. The *pcvcount* policy, which did not piggyback a validation check more times than the access count, was a little better than *pcvadapt* in staleness ratio and much worse in costs (many more non-piggybacked validation checks were needed). The *pcvfreq* policy, which uses a higher maximum TTL for new resources in the cache, showed slightly decreased costs with a comparable increase in staleness ratio.

We next examined the impact of access frequency on cache replacement by using the GreedyDual-Size algorithm with *access_count/size* as the base value to create the *gdlfu* policy. Figure 4 shows the results of this policy, combined with both the *ttladapt* and *hybrid* coherency policies, along with two baseline replacement policies paired with *ttladapt*.

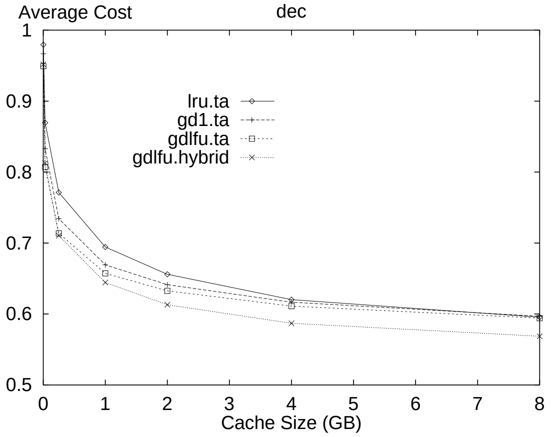

Figure 4. Average Cost versus Cache Size for Digital Traces Using Frequency of Use in Cache Replacement

The *gdlfu.ta* combination results in lower costs than *gd1.ta*. For example, for a 1 GB cache (5% of cache needed), the *gdlfu.ta* combination results in 1.7% reduced costs compared to *gd1.ta* and 5.3% savings when compared to *lru.ta*. Comparing cache hit rate, a traditional measure of performance, *lru.ta* has a 35.8% hit rate, *gd1.ta* a 42.6% hit rate and *gdlfu.ta* a 44.1% hit rate. When combined with the *hybrid* coherency policy, the *gdlfu.hybrid* combination reduces costs by 4.5% and staleness ratio by 89% when compared to the base combination of *lru.ta* for an 8 GB cache.

The effect is the same, but less pronounced in the AT&T traces. Using a 0.25 GB cache (10% of cache needed), the *gdlfu.ta* combination results in 0.7% reduced costs compared to *gd1.ta* and 2.0% savings when compared to *lru.ta*. Cache hit rates for this cache size are 31.5% for *lru.ta*, 36.0% for *gd1.ta* and 36.7% for *gdlfu.ta*. When combined with the *hybrid* coherency policy, the *gdlfu.hybrid* combination reduces costs by 2.8% and staleness ratio by 86% when compared to the base combination of *lru.ta* for a 2 GB cache. These improvements indicate that consideration of the frequency of use in a simple manner using the GreedyDual-Size algorithm results in better performance than previously proposed cache replacement policies.

6. Future Work

The primary directions for future work are to examine the impact of new developments in the Web on cache coherency and replacement. The use of multi-layered proxy hierarchies [5] have potential implications for the piggybacked approach to cache coherency; we need to examine how piggybacked information is propagated up and down the hierarchy.

The availability of trailer and pipelining mechanisms in HTTP/1.1 [9] also allows us to explore reductions in the latency and bandwidth overhead of piggybacking in servicing a request. In addition, the persistent connection feature of HTTP/1.1 is expected to have a major impact on the performance of the World Wide Web by reducing network overhead and latency. The importance of cache replacement and coherency, even content caching itself [2], need to be considered in light of this new development. We have made a preliminary study on the performance of replacement and coherency policies in light of persistent connections. Using an approach similar to Manley and Seltzer [13], we assume that a network connection between a proxy cache and a server persists for a window of time beyond the last request from the proxy to the server. While this assumption may not hold for all client/server combinations, it serves as a baseline to measure "connection cost," the ratio of requests that require a new network connection to be created. Using a short 5 second time out window for the Digital traces, the maximum number of connections open at one time for an 8 GB cache is 91.

The results show that approximately 70% of requests can be retrieved on a persistent connection even with a small cache and the ratio stays relatively constant even with a much larger cache. However, the results show that the policy combination does have some effect on the number of connections. For an 8GB cache, the *gdlfu.hybrid* combination results in a new connection for 24.5% of the requests in comparison ot 26.4% for *lru.ta*. These initial results indicate that persistent connections have an impact on overall performance, but that coherency and replacement policies are still important to consider in this environment.

7. Summary

The outcome of this work is a better understanding of how cache replacement and coherency policies interact with each other. Through the use of a trace-driven simulation that allows different replacement and coherency policies to be combined in the context of a uniform cost model, we have been able to evaluate the relative cost improvements due to each type of policy, the impact of incorporating coherency issues in cache replacement, and the inclusion of other factors in these policies.

The major results of this study are:

- Cache replacement and coherency are both important in reducing the costs for a proxy cache. Better cache replacement policies, based on the GreedyDual-Size algorithm, were shown to have the primary impact on costs for cache sizes of less than 15-25% of the total cache needed. Better cache coherency policies, based on the piggybacked validation mechanism, were shown to have the primary impact for larger cache sizes.

- Direct inclusion of cache coherency issues in cache replacement policies yields little improvement in overall performance in our study. While the consideration of cache lifetime for a resource appears desirable to eliminate validation requests, enough of this resources must be retrieved again for future requests to raise the overall costs. Using the observation that HTML/text resources change more often results in small improvements in performance.

- Consideration of the frequency of use in cache coherency policies does not yield better performance, but its inclusion in the GreedyDual-Size algorithm results in a better cache replacement policy than found in previously published work. Combined with the best piggyback-base cache coherency policy results in a 4.5% decrease in costs and 89% reduction in staleness ratio when compared to existing policy combinations in current use.

Future work needs to examine the impact of new developments in the Web on cache coherency and replacement. Preliminary work suggests that better replacement and coherency policies continue to provide lower costs in light of new features such as persistent connections.

8. Acknowledgments

We thank Digital Equipment Corporation for making their proxy traces available to us. We thank Anja Feldmann for help with the AT&T trace. Thanks to Jennifer Rexford, Mikhail Mikhailov and the anonymous reviewers for making comments on draft versions of the paper.

References

[1] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: Limitations and potentials. In *Proceedings of the Fourth International World Wide Web Conference*, December 1995.

[2] Ramon Caceres, Fred Douglis, Anja Feldmann, Gideon Glass, and Michael Rabinovich. Web proxy caching: the devil is in the details. In *Workshop on Internet Server Performance*, Madison, Wisconsin USA, June 1998.

[3] P. Cao, J. Zhang, and K. Beach. Active cache: Caching dynamic contents (objects) on the web. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, The Lake District, England, September 1998.

[4] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Symposium on Internet Technology and Systems*. USENIX Association, December 1997.

[5] Anawat Chankhunthod, Peter Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical internet object cache. In *Proceedings of the 1996 USENIX Annual Technical Conference*, San Diego, California, USA, January 1996. USENIX Association.

[6] Digital Equipment Corporation. Proxy cache log traces, September 1996.

[7] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey 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.

[8] James Gwertzman and Margo Seltzer. World-wide web cache consistency. In *Proceedings of the USENIX Technical Conference*, pages 141–152. USENIX Association, January 1996.

[9] Balachander Krishnamurthy, Jeffrey C. Mogul, and David M. Kristol. Key differences between HTTP/1.0 and HTTP/1.1. In *Eighth International World Wide Web Conference*, Toronto, Canada, May 1999.

[10] Balachander Krishnamurthy and Craig 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.

[11] Balachander Krishnamurthy and Craig E. Wills. Piggyback server invalidation for proxy cache coherency. In *Seventh International World Wide Web Conference*, pages 185–193, Brisbane, Australia, April 1998.

[12] Chengjie Liu and Pei Cao. Maintaining strong cache consistency in the world-wide web. In *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems*, May 1997.

[13] Stephen Manley and Margo Seltzer. Web facts and fantasy. In *Symposium on Internet Technology and Systems*. USENIX Association, December 1997.

[14] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *ACM SIGCOMM'97 Conference*, September 1997.

[15] Peter Scheuermann, Junho Shim, and Radek Vingralek. A case for delay-conscious caching of web documents. In *Sixth International World Wide Web Conference*, Santa Clara, California, USA, April 1997.

[16] Junho Shim, Peter Scheuermann, and Radek Vingralek. Proxy cache design: Algorithms, implementation and performance. *IEEE Transactions on Knowledge and Data Engineering*, 1999.

[17] Squid internet object cache. `http://squid.nlanr.net/Squid`.

[18] Stephen Williams, Marc Abrams, Charles R. Standbridge, Ghaleb Abdulla, and Edward A. Fox. Removal policies in network caches for world-wide web documents. In *Proceedings of the ACM SIGCOMM Conference*, pages 293–305, August 1996.

[19] Craig E. Wills and Mikhail Mikhailov. Towards a better understanding of web resources and server responses for improved caching. In *Eighth International World Wide Web Conference*, Toronto, Canada, May 1999.

[20] Jian Yin, Lorenzo Alvisi, Michael Dahlin, and Calvin Lin. Using leases to support server-driven consistency in large-scale systems. In *Proceedings of the 18th International Conference on Distributed Systems*. IEEE, May 1998.