A Web Object Management Policy for Cooperative Hybrid Caching Architecture

Jinsuk Baek, Mingyung Kwak, Paul S. Fisher, and Elva J. Jones
Department of Computer Science
Winston-Salem State University
Winston-Salem, NC 27110 USA
e-mail: {baekj, mkwak106, fisherp, jonese}@wssu.edu

Abstract—We discuss a predictive, hybrid caching structure that has eliminated the performance issues of the existing structures and proposed a new policy for discarding web objects that are not likely to be accessed by clients. Our proposed approach is performed based upon the predictive technique using tables of rules derived from actual experience of the web object requests. We discuss the previously proposed schemes with their disadvantages, and then describe an improved approach employing summary tables in each proxy cache and allowing the limited number of executions of the expensive predictions. We provide a simulation of the proposed policy using the NLANR proxy traces to show the impact this proposed policy has on the performance of the Internet with web object traffic.

Keywords- Finite Inductive Sequences, Object Discarding, Hybrid Caching Architecture, Reference Table, Summary Table.

I. INTRODUCTION

With the increasing popularity of the WWW, web traffic has become one of the most resource consuming applications on the Internet. This increasing use of the web results in increased network bandwidth usage, straining the capacity of the supporting networks. Hence, many solutions to improve bandwidth utilization of web traffic have been proposed. These solutions include the use of compression and delta-encoding [1], multicast [2], new congestion avoidance algorithms [3], and so on. Among many proposals, caching the popular web objects at different levels of proxy sites is one of the most efficient approaches promising network bandwidth saving and Internet traffic reduction.

Proxy caches can be placed at different levels in the network to serve the clients, and they also cooperate with one another in case of a cache miss. If a requested web object from a client is not found in a local proxy cache, the proxy cache communicates with its sibling or nearby proxy caches to find that object. In the event that object is not found in the nearby caches only then is the request forwarded to the original server. This cooperation between the proxy caches is called cooperative caching.

Two of the most common architectures to implement large scale proxy cache cooperation are hierarchical [4] and distributed caching systems [5]. A new architecture [6] has recently been proposed by one of the authors, namely a hybrid caching architecture, using both the hierarchical caches as well as the same level caches. Under the architecture, an FI-based object discarding policy [7] has been proposed to show a performance improvement in terms of hit ratio and response time when compared to the existing two architectures. However, we have found the frequent executions of FI systems can lower the performance of the proxy cache.

We propose a new object management policy focused on this issue. With our policy, each lower level proxy cache has the summary table that contains its neighbor proxy caches’ object information. The summary table specifies the location of the requested object when the object is not available in the proxy cache. In addition, in order to boost the performance of the proxy cache, the proposed solution limits the number of executions of FI systems depending upon the current available space of the proxy cache. As a result, our solution provides reduced response time for the requested object by minimizing unnecessary traffic and bandwidth usage between the low level proxy caches and upper level proxy cache.

The outline of the rest of this paper is as follows. In Section 2, we briefly review the existing caching architectures including the hybrid caching architecture. In Section 3, we describe a new object management policy that can be applied to the hybrid caching architecture. We show the performance of the proposed solution in Section 4 and conclude in Section 5.

II. EXISTING SOLUTION

Hierarchical caching systems basically work from the lowest levels toward the highest levels. That is, if a cache miss occurs at a given level of cache, the request is forwarded to a higher level cache. If the requested object is not found in the higher level cache, the forwarding goes on until the requested object is found. In distributed caching, each object is allowed to be cached only at the lowest level, and a cache can obtain an object from neighboring caches. It effectively tackles many of the drawbacks of hierarchical caching. There are several approaches including Internet Cache Protocol (ICP) [8], Caching Array Routing Protocol (CARP) [9], Summary Cache [10], and Home Cache [11]. Unfortunately, each architecture has performance limitations in terms of message overhead, response time and object duplication. Comprehensive studies [12, 13] on both architectures found that hierarchical caching provides shorter connection time than distributed caching while the latter provides shorter transmission time and higher bandwidth usage. To overcome the drawbacks of both architectures, the
hybrid caching architecture [6] was proposed. In order to retrieve the requested object from its lower level proxy caches, it employed one upper level proxy cache, which maintains a reference table containing the location of the proxy cache having the location of the requested web object.

III. PROPOSED SOLUTION

A. Hybrid Architecture with Reference Table and Summary Table

In order to simplify our discussion, we assume there are two-level hierarchies of caches as depicted in Fig. 1. If the proxy cache does not have the clients requested object, it first refers to its summary table. The summary table specifies the location of the web object by using two fields, namely {Proxy ID} and {Object ID}. {Proxy ID} represents the location of the web object while {Object ID} represents the web object. In case the summary table has the requested object’s information, then the proxy cache simply gets a copy of the requested object from the corresponding proxy cache. In other case, it forwards the request to the proxy cache that is at the next level of the hierarchy. This next level proxy cache tries to locate the object by searching for it within the reference table it maintains. This reference table also contains the different fields specifying the location of the web objects, {Proxy ID} and {Object ID}. Therefore, the upper level proxy cache tries to locate the requested objects using this reference table. However in case the reference table does not have any information about the requested object, then the request is forwarded to the highest level in the hierarchy that is the original server in our example.

When the requested object is retrieved from the original server, then the reference table for each level proxy cache gets updated down through the hierarchy. However, a copy of the web object is not stored in any of the intermediate proxy caches overcoming the problem of duplication of objects found in the hierarchy architecture. Once the proxy cache receives the requested object, it informs its neighbor proxy caches to enable them to update their summary tables. As the reference table keeps an up to date picture of what has been cached in the lowest level proxy caches, False Misses are not introduced. False Hits are possible though, especially when the lowest level proxy caches discard the object using a cache replacement policy, in case the cache is full. In order to avoid False Hits, the proposed policy requires every proxy cache to keep an object with a flag bit. This flag bit is initially set to 1.

Let us assume a proxy cache P1 has an object O1, whose flag bit is set to 1 and another proxy cache P2 requests that object O1. When the proxy cache P2 does not have the object O1, it will refer to its summary table to find out the location of the object O1. In case the location of the object O1 is found from the summary table, the proxy cache P2 sends a request to the proxy cache P1, for the object O1. In the other case, the proxy cache P2 sends a request to the upper level proxy cache. The upper level proxy cache will refer to its reference table to find out the location of the object O1 and will forward that request to the proxy cache P1. After both cases, the proxy cache P2 will now get a copy of the object O1, and its flag bit is set to 1 while the proxy cache P1 will set its flag bit for the O1 to 0, indicating that it can be discarded without informing its upper level proxy cache or neighbor proxy caches. The reference table gets autonomously updated as the object travels from one proxy cache to another. The proxy cache P2 will inform its neighbor proxy caches to refresh their summary tables reflecting the fact that the proxy cache P2 is the new location of the object O1.

The proxy cache P1 can now discard the object O1, because the reference table and summary tables have been updated so they have no information about object O1 cached in the proxy cache P1. By discarding the web object, duplications can be removed efficiently which usually happens when one proxy cache gets a web object from another proxy cache on the same level after referring to the two tables.

However, after observing the users’ access pattern, we have noticed that the possibility of requesting the same object by some of the clients under the same proxy cache is very high. In that case the client has to send a request to a lower level proxy cache (P1 in our example) for the same object, and P1 will have to contact another proxy cache (P2 in our example) after contacting the upper level proxy cache to get the same object that was just discarded from its local proxy cache P1. This simply increases the network traffic in between the lower level proxy and the upper level. It also introduces unnecessary delay in response time.

B. Object Discarding Policy with FI

If we keep the object even after it has been referenced by another proxy cache, there is a waste of space caused by duplicated objects over the some proxy caches located at the same level. This penalty, however, can be compensated for by the reduced network traffic and response time. The most promising solution is to find an efficient way to decide if the proxy cache discards the object or not without introducing a significant penalty. This can be done by analyzing the clients’ object access patterns. In order to achieve this, we consider FI Sequences [14].

Figure 1. Example of the hybrid caching architecture with summary tables for a LAN
In order to simplify our example, we assume that the numeric values represent object IDs and note that we add the starting symbol ‘s’ to each string in order to shorten the residuals from each string. From the four strings the starting implicates are: s2→2; s6→6; s7→7; but since this is simultaneous factoring, these implicents are contradictory, so they are not suitable. The implicents s2→4; s6→6; s7→7 are consistent. We now provide the table of implicents augmented with their source, for each of the levels in the FI ruling table. These are given in Table 1 with the inductive base held to 2. As can be seen, this table also includes information from the client(s) that generated the rule. The final residuals for each of the strings are those symbols that remain due to inter/intra-string conflicts. C1: s2; C2: s65; C3: s62; and C4: s7.

The ruling table results from the simultaneous factoring of the requested object of the four clients. The issue with this example is that each client examined a different phenomenon rather than having several clients reporting different views of the same phenomenon. Clearly, the more data one has about an event the more accurately one can identify that event. In order to show how rulings work in terms of object access prediction, we now create another table representing consequences with their antecedents. This is shown in Table 2 and the number in the parentheses represents the level where the rule can be found. Therefore, the proxy cache firstly records the previous request history and creates the ruling and antecedent/consequence tables. It also records the current request history. When there is an object request from another proxy cache, it will send the object to the proxy cache. In order to decide if the object is discarded or not, it references the tables to check if the object is uniquely identified by antecedent(s). Based upon this result, the proxy cache can deterministically decide whether an object is to be discarded or not.

For example, if there is a request for object 2 which has eight different antecedents, the proxy caches will check if the current request history includes one of these patterns. If so, it will keep this object. Otherwise, it will not. With our proposed solution, each proxy cache does not always need to discard a requested object to avoid duplication of the object over the same level proxy caches.

Recall the example we discussed in a previous subsection. The proxy cache P1 decides whether the object O3 is discarded or not. As shown in Table 2, this decision depends on whether the object O3 can be uniquely identified by antecedents. When another proxy cache, say P2, requests the same object, the upper level proxy cache requires P1 to contact P2 rather than P3 because the P2’s flag bit will be set to 1. The upper level proxy cache updates the reference table by reflecting that the object O3 is now located in P1, P2, and P3. In addition, for object O3, P1’s flag bit is set to 1 while the P1 and P2 set it to 0. As a result, the workloads for processing the object requests are fairly distributed among the proxy caches which have the object. We also need to mention that we can use a probabilistic/non-deterministic FI system as the object requests from the clients is likely to be non-deterministic but semi-deterministic. This can be done.

An FI sequence S = s1, s2, ..., s8 over a set of symbols P = {P1, P2, ..., Pm} is a pattern in which some preceding subsequence s1, ..., sj uniquely identifies sj+1. If every substring y = s1, ..., sj uniquely identifies sj+1, then the pair, y→sj+1, is called an antecedent, and the substring y is called the antecedent, and sj+1, the consequent. If no subset of y also uniquely identifies sj+1, then y is said to be a reduced form antecedent and y→sj+1 is called a reduced form implicated.

We now assume that every implicent is represented in its reduced form. In these types of sequences, we can store the antecedents and expunge the consequent from the sequence since the antecedent uniquely identifies the consequent. When consequents are eliminated from FI sequences, we produce a new sequence called a residual sequence which may also generate new implicents. The process of generating implicents and residuals from a sequence is called factoring. The factoring process can be repeated until we cannot find any more implicents. The set of all implicents and residuals for a sequence is called a ruling and the maximum length of antecedents in a ruling for a level is called the inductive base for that level and the inductive base ≥ 1. We note that if an FI sequence has inductive base n and an alphabet of k symbols, then kn is an upper bound for the number of its reduced form implicents.

Let us suppose there are four clients in a subnet and a proxy cache records their object request history for some fixed amount of time. Let the following strings represent the request history for the four clients – namely C1, C2, C3, and C4. The four strings are:

C1: s24466688884243 C2: s666665322187 C3: s66662254542217 C4: s8765442218

In order to simplify our example, we assume that the numeric values represent object IDs and note that we add the starting symbol ‘s’ to each string in order to shorten the residuals from each string. From the four strings the starting implicents are: s2→2; s6→6; s7→7; but since this is simultaneous factoring, these implicents are contradictory, so they are not suitable. The implicents s2→4; s6→6; s7→7 are consistent. We now provide the table of implicents augmented with their source, for each of the levels in the FI ruling table. These are given in Table 1 with the inductive base held to 2. As can be seen, this table also includes information from the client(s) that generated the rule. The final residuals for each of the strings are those symbols that remain due to inter/intra-string conflicts. C1: s2; C2: s65; C3: s62; and C4: s7.

The ruling table results from the simultaneous factoring of the requested object of the four clients. The issue with this example is that each client examined a different phenomenon rather than having several clients reporting different views of the same phenomenon. Clearly, the more data one has about an event the more accurately one can identify that event. In order to show how rulings work in terms of object access prediction, we now create another table representing consequences with their antecedents. This is shown in Table 2 and the number in the parentheses represents the level where the rule can be found. Therefore, the proxy cache firstly records the previous request history and creates the ruling and antecedent/consequence tables. It also records the current request history. When there is an object request from another proxy cache, it will send the object to the proxy cache. In order to decide if the object is discarded or not, it references the tables to check if the object is uniquely identified by antecedent(s). Based upon this result, the proxy cache can deterministically decide whether an object is to be discarded or not.

For example, if there is a request for object 2 which has eight different antecedents, the proxy caches will check if the current request history includes one of these patterns. If so, it will keep this object. Otherwise, it will not. With our proposed solution, each proxy cache does not always need to discard a requested object to avoid duplication of the object over the same level proxy caches.

Recall the example we discussed in a previous subsection. The proxy cache P1 decides whether the object O3 is discarded or not. As shown in Table 2, this decision depends on whether the object O3 can be uniquely identified by antecedents. When another proxy cache, say P2, requests the same object, the upper level proxy cache requires P1 to contact P2 rather than P3 because the P2’s flag bit will be set to 1. The upper level proxy cache updates the reference table by reflecting that the object O3 is now located in P1, P2, and P3. In addition, for object O3, P1’s flag bit is set to 1 while the P1 and P2 set it to 0. As a result, the workloads for processing the object requests are fairly distributed among the proxy caches which have the object. We also need to mention that we can use a probabilistic/non-deterministic FI system as the object requests from the clients is likely to be non-deterministic but semi-deterministic. This can be done.
by considering the frequency of the antecedent/consequent relationships for a specific object.

However, we have found that FI-based discarding policy can lower the performance of the proxy cache because it could impose non-negligible computations if we perform this operation whenever an object is requested by another proxy cache. Therefore, we limit the number of FI executions depending upon the FI criteria. That is, suppose the proxy cache has more than 50% of the maximum cache size space available, it does not perform FI-based object discarding. Instead, it keeps the object. This allows the proxy caches to maintain consistent performance.

IV. PERFORMANCE

To evaluate the performance of our proposed policy, we have developed a trace driven simulation program. In our simulation, we use a one day real log from the National Laboratory for Applied Networking Research (NLANR) [15] that has approximately 30,000 requests. The simulation results eventually show how the hit rate and the response time changes with the size of the proxy caches as well as the number of the lowest level proxy caches. The results also show how the different number of executions of FI systems and the presence of the summary table affect the number of object requests forwarded to the upper level proxy cache.

In our simulation, the clients are assigned to various proxy caches depending upon their IP addresses. Hence, all the clients having a particular range of network address will correspond to the same proxy cache. The average round trip time changes depending upon whether it is a first level proxy hit or a upper level proxy hit. The various fields considered in the web logs are the input parameters to our simulation program. By varying the parameter settings and values we can simulate and compare the performance of the proposed policy with the policy that does not implement a summary table and uses an FI-based discarding policy without any criteria.

We consider there are six proxy caches and one upper level proxy cache. All six proxy caches are connected using the star topology where the upper level proxy cache is located at the center of all the proxy caches. After allocating each client to their proxy caches based on the network address, we could observe that each proxy cache has from 8 to 21 different clients and from 2193 to 9681 different or repeated requests. We note that we have six proxy caches but the real trace data were generated by one proxy cache having multiple clients, we need to adjust the response times that will be varied in different scenarios in our solution. For example, with our simulation, a client will be assigned to one of the six different proxy caches. Even though we assume all six proxy caches have the same cache size, the cache size is different from that of the real proxy cache where the traces were generated. Hence, even though the real trace shows that the requested web object is available in its proxy cache, this is not always true in our environment. If our proxy cache and the real proxy cache show the same hit/miss result for a given object request, we can use the real response time in the trace. Otherwise, our simulator needs to approximate the response time based on the real response time. We now show how we approximate the response time between the upper level proxy and original web server in case a cache miss has occurred. From the traces, we get the average response time for a local hit and miss. We define $A_{\text{client, server}}$ as the average total elapsed time in the case of a cache miss while $A_{\text{client, proxy}}$ is the average round trip time in the case of a cache hit in the trace. If we assume $A_{\text{proxy, server}}$ is the average round trip time between the proxy cache and the original web server, the $A_{\text{proxy, server}}$ can be given by taking $A_{\text{client, proxy}} - A_{\text{client, server}}$. We could get the real values for $A_{\text{proxy, server}} (=1115$ ms), $A_{\text{client, proxy}} (=28$ ms), and $A_{\text{client, server}} (=1179$ ms) from the real trace. Hence, we use $A_{\text{proxy, server}}$. Whenever the requested object is not available in both a lower level proxy cache and an upper level proxy cache. Based on the $A_{\text{client, proxy}}$, we can also estimate the response time between various proxy caches. We also assume the propagation delay between the proxy caches is at least three times longer than the time between the client and the lower level proxy cache.

TABLE III. HIT RATE FOR PROXY CACHES WITH FIVE DIFFERENT CACHE REPLACEMENT POLICIES WHEN THE FI CRITERIA IS SET TO 30%

<table>
<thead>
<tr>
<th>Replacement</th>
<th>FIFO</th>
<th>LIFO</th>
<th>Size</th>
<th>Frequency</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary table</td>
<td>$W$</td>
<td>$WO$</td>
<td>$W$</td>
<td>$WO$</td>
<td>$W$</td>
</tr>
<tr>
<td>$P_1$</td>
<td>54.6</td>
<td>54.6</td>
<td>52.1</td>
<td>51.8</td>
<td>54.6</td>
</tr>
<tr>
<td>$P_2$</td>
<td>45.1</td>
<td>45.1</td>
<td>41.0</td>
<td>40.8</td>
<td>46.2</td>
</tr>
<tr>
<td>$P_3$</td>
<td>31.7</td>
<td>31.7</td>
<td>29.7</td>
<td>29.7</td>
<td>31.3</td>
</tr>
<tr>
<td>$P_4$</td>
<td>60.3</td>
<td>60.4</td>
<td>59.0</td>
<td>60.3</td>
<td>62.5</td>
</tr>
<tr>
<td>$P_5$</td>
<td>37.3</td>
<td>37.4</td>
<td>33.3</td>
<td>34.8</td>
<td>37.5</td>
</tr>
<tr>
<td>$P_6$</td>
<td>46.7</td>
<td>46.8</td>
<td>45.6</td>
<td>46.1</td>
<td>49.2</td>
</tr>
</tbody>
</table>

TABLE IV. AVERAGE RESPONSE TIME WITH FIVE DIFFERENT CACHE REPLACEMENT POLICIES WHEN THE FI CRITERIA IS SET TO 30% (MILLISECOND)

<table>
<thead>
<tr>
<th>Replacement</th>
<th>FIFO</th>
<th>LIFO</th>
<th>Size</th>
<th>Frequency</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary table</td>
<td>$W$</td>
<td>$WO$</td>
<td>$W$</td>
<td>$WO$</td>
<td>$W$</td>
</tr>
<tr>
<td>$P_1$</td>
<td>143</td>
<td>562</td>
<td>126</td>
<td>564</td>
<td>119</td>
</tr>
<tr>
<td>$P_2$</td>
<td>185</td>
<td>662</td>
<td>185</td>
<td>664</td>
<td>184</td>
</tr>
<tr>
<td>$P_3$</td>
<td>39</td>
<td>926</td>
<td>59</td>
<td>927</td>
<td>59</td>
</tr>
<tr>
<td>$P_4$</td>
<td>115</td>
<td>1022</td>
<td>134</td>
<td>1023</td>
<td>132</td>
</tr>
<tr>
<td>$P_5$</td>
<td>62</td>
<td>386</td>
<td>63</td>
<td>387</td>
<td>62</td>
</tr>
<tr>
<td>$P_6$</td>
<td>118</td>
<td>622</td>
<td>118</td>
<td>622</td>
<td>117</td>
</tr>
</tbody>
</table>
The result of the hit rate is shown in the Table 3. As shown, the hit rate of the proposed solution is equal to or slightly less than that of the hybrid architecture without the summary table from -4% to 0.5%. This small penalty is compensated for because of the reduced number of forwarded requests to the upper level proxy cache and the reduced number of executions of FI Systems. The results given in Table 5 and 6 show our proposed solution has less FI systems executions and less requests that are forwarded to the upper level proxy cache. In Table 4, we show our proposed solution numerically outperforms the previous solution from 72% to 94% in terms of response time. These results indicate that our proposed solution helps the proxy caches to have better performance and to have less network traffic between the upper level proxy cache and the proxy caches. Fig. 2 shows the response time for each client of proxy cache $P_j$. It shows client 3 will have much shorter response time when the summary table is applied.

### V. Conclusion

We have proposed a new object management policy that can be applied in the hybrid architecture for cooperative caching that efficiently overcomes the limitations of existing caching architectures. As the hybrid approach inherits the distributed and hierarchical architecture, it has reduced querying overhead and overcomes the design complexities of both architectures by employing the upper level proxy cache having a reference table. Moreover, our new policy has reduced the response time by employing the summary table in each proxy cache and improved the performance of the proxy cache by limiting the executions of FI systems. Our solution will minimize unnecessary network traffic when it is deployed in a LAN environment. Also, it can be applied to the WAN environment in a straightforward manner. More research is yet to be done in order to determine the various summary and reference table sizes.

### Table V. The Number of Executions of FI Systems With/Without Summary Table

<table>
<thead>
<tr>
<th>Summary table</th>
<th>With FI criteria</th>
<th>With 30%</th>
<th>With 40%</th>
<th>With 50%</th>
<th>With 60%</th>
<th>With 70%</th>
<th>Without</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number of executions</td>
<td></td>
<td>929</td>
<td>963</td>
<td>981</td>
<td>996</td>
<td>1007</td>
<td>1021</td>
</tr>
</tbody>
</table>

### Table VI. The Number of Object Requests Forwarded to the Upper Level Proxy Cache With/Without Presence of the Summary Table

<table>
<thead>
<tr>
<th>Summary table</th>
<th>With FI criteria</th>
<th>With 30%</th>
<th>With 40%</th>
<th>With 50%</th>
<th>With 60%</th>
<th>With 70%</th>
<th>Without</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number of requests</td>
<td></td>
<td>12723</td>
<td>12731</td>
<td>12731</td>
<td>12731</td>
<td>12731</td>
<td>29783</td>
</tr>
</tbody>
</table>

### Acknowledgment

This research was supported in part by the Winston-Salem State University Research Initiation Program (WSSU-RIP) under grant RIP-121203. A preliminary version of this paper was presented at 2008 ACMSE Conference, Auburn, AL, March 2008. This present paper is a revised version which has been extended to include comprehensive simulation results.

### References