A New Hybrid Architecture for Cooperative Web Caching

Jinsuk Baek,Gurpreet Kaur and Junghoon Yang

Abstract—An effective solution to the problems caused by the explosive growth of World Wide Web is a web caching that employing an additional server, called proxy cache, between the clients and main server for caching the popular web objects near the clients. However, a single proxy cache can easily become the bottleneck. Deploying groups of cooperative caches provides scalability and robustness by eliminating the limitations caused by a single proxy cache. Two common architectures to implement the cooperative caching are hierarchical and distributed caching systems. Unfortunately, both architectures suffer from performance limitations. We propose an efficient hybrid caching architecture eliminating these limitations by using both the hierarchical and same level caches. Our performance evaluation with our investigated simulator shows that the proposed architecture offers the best of both existing architectures in terms of cache hit rate, the number of query messages from clients, and response time.

Index Terms—Proxy cache, hierarchical architecture, distributed architecture, cooperative caching

1 Introduction

With the increasing popularity of the WWW, web traffic has become one of the most resource consuming applications on the Internet. The reason behind that was the growth of network traffic at a hyper-exponential rate while network infrastructure does not. This increasing use of the web results in increased network bandwidth usage, straining the capacity of the networks on which it runs. Hence, solutions to improve bandwidth utilization of web traffic are currently an important issue. Different directions of research are being investigated in this area, including the use of compression and delta-encoding [8], multicast, server-push and new congestion avoidance algorithms [9].

Two most direct ways of reducing web traffic have been implemented recently. On the one hand, improvements have been made in the Hypertext Transfer Protocol (HTTP). The interaction with the lower level protocol has been modified in HTTP1.1 to support, in particular, persistent connections and pipelining, in order to increase network efficiency. On the other hand, caching at different levels is proposed and implemented: caching at the client side and in the backbone network.

Indeed, according to statistical analysis, some web servers and web documents are much more popular than others. Hence, the request stream exhibits some locality in several clients, especially in the same area, by requesting the same files to the same server during a relatively short time interval. A proxy cache at the client side may therefore avoid superfluous downloads of such files and hence save network bandwidth as well as server capacity.

In a proxy cache, a single host is used for one to several hundred clients. Each client sends its requests through the proxy which keeps copies of objects in the cache. Hence, the involvement of the proxy cache has decreased the user perceived network latency. Proxy caches are placed at different places in the network to serve the clients. These proxy caches 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

• *601 M.L.King Jr Dr, Department of Computer Science, Winston-Salem State University, Winston-Salem, NC 27110, USA.*

This work was supported in part by the 2007 Winston-Salem State University Research Initiation Program (WSSU-RIP 2007) under Grant RIP-121203.

Manuscript received February 15, 2008; revised March 20, 2008.

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 the request is forwarded to the original server. This cooperation between the proxy caches is called the cooperative caching.

There are many issues we should consider to make a cooperative caching system works properly. These issues include caching system architecture, caching contents, proxy cache cooperation, cache resolution and routing, prefetching, cache placement and replacement, and cache coherency. Among these, our focus is on the caching architecture, one of the most important issues of the cooperative caching system, since it provides the paradigm for efficient inter-cooperation between proxy caches.

Two common architectures to implement large scale proxy cache cooperation are hierarchical and distributed caching systems. Hierarchical caching system basically works 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. Although this architecture introduces different levels of cache to increase the overall cache hit rate, it still has some flaws. One of the drawbacks is that it can introduce an additional delay whenever the request is not satisfied in a given level of cache since the request is forwarded to upper levels in hierarchy. Another drawback is that the higher level cache can have a longer queue than the lower level cache resulting in a bottleneck.

In a distributed caching, each object is allowed to be cached only at the lowest level and a cache can obtain an object from the neighboring caches. It effectively tackles many of the drawbacks of hierarchical caching. There are several approaches including Internet Cache Protocol (ICP) [13], Caching Array Routing Protocol (CARP) [12], Summary cache [7], Hint cache [10] and Home cache [14]. Unfortunately, each of the schemes has its own limitation. A comprehensive study [11] 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.

In order to reduce limitations of the existing solutions, we propose a more efficient hybrid caching architecture using both the hierarchy of caches as well as same level caches. Our hybrid caching basically works as follows. Each cache has multiple sibling caches and one upper level cache to fetch the object. If a client requests a web object to its nearest proxy cache, this cache will firstly look for that object within itself. If the object is not found then it will pass the request to the next level proxy cache. Each level of the proxy cache, except for the lowest level of caches in the hierarchy, creates and maintains their reference tables. This reference table includes some fields representing the location of the proxy cache having the requested web object. The location would be one of its child proxy caches. This cache will now look for that web object within itself, and if the object is not present in the cache it will look in its reference table to find the location of the object. This table contains the information about the contents of its lower level proxy caches. If the table does not include any information about where the requested web object is available in its child proxy caches, then the request will be forwarded to the original web server. Unlike the hierarchical caching, each level of proxy cache will update its reference table rather than keeping the copy of the object.

The major advantages of the proposed architecture are as follows. First, our new approach will overcome the problem with hierarchy caching, because each level of proxy caches keeps the information about the contents of its lower level caches in their reference tables. Hence, it can resolve the problem of long queuing delays and avoid the duplication of the object in intermediated caches. Second, it outperforms the distributed caching architecture, because the lowest level of cache does not broadcast the request to all of its sibling caches when the request is not satisfied from within. Therefore, it can overcome the problem of bandwidth consumption because our approach will use the reference table to find out the exact location of the proxy cache having the requested web object. Finally, our

approach scales well when it is applied to the wide area network (WAN), because each level of caches can handle more first level caches by employing the reference table.

The outline of the rest of this paper is as follows. In Section 2, we summarized the existing solutions. In Section 3, we describe our hybrid caching architecture in more detail. We show the performance of the proposed solution in Section 4. We conclude this thesis and suggest future work in Section 5.

2 Related Work

In this Section, we discuss the existing architectures and their drawbacks. Two well known existing architectures based on which we proposed our architecture are hierarchical and distributed caching architectures. A hierarchical caching [6] was first proposed in the Harvest project [4]. In hierarchical caching architecture, proxy caches exist in several network levels, such as bottom, institutional, regional, and national level. The bottom level of the hierarchy is the client/browser cache. When a request is not satisfied by the client cache then it is forwarded to the institutional level. The request for the web object travels from the lower level to the higher level i.e. from institutional to regional and then to national level. If the web object is not found in all of the levels then the request is forwarded to the original server. As the web object travels down the hierarchy it leaves a copy at each level. In the same way all the requests travel up the hierarchy until they are satisfied either by some level cache or by the original server. A hierarchical architecture is more bandwidth efficient in cases where the cooperating caches do not have fast connectivity. It also has shorter connection times than distributed caching but there are some problems associated with this architecture [3].

1) To set up the hierarchy, one needs to find the key access points in the network, so that a maximum coordination can be maintained between the proxy caches;
2) The hierarchy of caches increases the chances of the web object being found in one of the levels but in case it is not found, then each level introduces an additional delay;
3) Sometimes long queues form at the higher levels with that becomes a bottleneck effects on the performance of the architecture. Sometimes intermediate levels introduce the processing delays as well; and
4) Saving multiple copies at each level of the hierarchy wastes time as well as space.

To overcome the drawbacks of the hierarchical architecture, a new architecture, distributed caching architecture, was introduced. This architecture minimized the delays by taking out the various levels. In this architecture, there is no intermediate cache except for the lowest level caches that were named as institutional level. In order to find out which institutional level has the requested web object, all the levels keep the information about the content of the other institutional level caches. To serve each others misses, caches cooperate with each other using different mechanisms. A query is generated by the lowest level and is broadcasted to its neighboring caches to find out the requested web object.

One strategy used in distributed cooperative caching is *Internet Cache Protocol* (ICP) [13]. ICP is the only communication protocol that does not impose any particular architecture of caches. In this protocol, every sibling cache as well as parent cache is explicitly queried whether they have a particular web object or not. And the sibling cache or parent cache sends back reply message indicating HIT or MISS. When a client requests a certain web object, a query is being sent to clients default proxy cache. If the object is not found in the default cache, the request is forwarded to other caches. Upon getting a HIT message from at least one of the proxy caches, the requested web object is transferred to the default proxy cache. But the drawback for this protocol is that querying every sibling cache increases the workload as well as Internet traffic. As a result, ICP has a *"negative scalability"* in that the more proxy caches added to the network, the more querying required between servers to determine location.

A more recent queryless distributed caching approach has evolved known as *Caching Array Routing Protocol* (CARP) [12]. CARP uses hash-based routing to provide a deterministic "request resolution path" through an array of proxy caches. For any given request, the browser or downstream proxy cache will know exactly where in the proxy array the requested object will be stored, whether it is already cached in any of the proxy caches, or need to contact the original server for information retrieval. Use of hash function completely eliminates the duplication of web objects and efficiently reduces the Internet traffic. In addition, CARP has a *"positive scalability"*. Due to its hash-based routing, and hence, its freedom from peer-to-peer pinging, CARP becomes faster and more efficient as more proxy caches are added. But the drawback for this protocol is that it works well only for Local Area Network (LAN). Therefore, only a fraction of web objects is stored in a cache, which facilitates only the local hit ratio. Moreover, it is not clear how well it performs for wide area cache sharing.

Summary cache [7] has been proposed to reduce bandwidth consumption. In summary cache, each proxy cache keeps the summary of the cache directory of each participating proxy, and checks these summaries for potential hit before sending any request to the original server. Summaries in the caches are updated timely and directory representation uses 8 bits per entry that is also very economical. Whenever an object is not found in the local proxy cache, then the local proxy cache will look into the summary to find out if it is a hit in some other proxy caches. If some other proxy caches have this requested object then the local proxy cache will send query to those particular proxy caches. However, the summary is not accurate all the times. Hence, in case the summary is not updated, the penalty will be a wasted query message while in case the summary does not reflect any information about an object that is cached in some other proxy caches, the penalty is higher miss ratio. Unfortunately, this technique introduces some storage overheads.

A *Hint cache* [10] contains a hint field {objectID, nodeID} pair where nodeID identifies the closest cache that contains the copy of objectID. That is, a hint cache knows where the closest copy of the requested object in a cluster of hint caches is. If an object is not in the local cache, it looks in its hint cache. If any other cache in the cluster has a copy of the requested object, it goes to the closest place. Otherwise, the query message is passed directly to the original server without going through siblings or parent cache. Once the cache has its own copy, it advertises that fact to the all caches in the same cluster. This cross-cache access can be very efficient in terms of cache performance. But the drawback for this technique is that it is complicate to implement and cause message overhead because the hints should be propagated through the entire cluster whenever there are cache misses. Even though hint cache adopts a notion of version through last-modified time and always goes for the most recent know version, it still contains an out of date picture of where the objects are cached especially when the cache content have to be replaced with other objects.

According to the *Home cache* [14], proxy caches can store any object locally in home but there is also a designated home cache for every object. In case of local miss, the cache only checks the home cache for the object. It uses the same concept of hashing function as CARP to associate home caches with the URL, but it can store other objects too. Generally, popular objects are replicated at all the caches but the less popular and large objects are stored only on the home cache. While it is found to be efficient, we need to point out that a) each proxy cache has to perform the hash function whenever the object is not found; and b) it is not applicable to the wide area network as the proposed solution will.

3 Proposed Hybrid Solution

In this Section, we describe a new hybrid caching architecture that can take advantage of both hierarchical and distributed caching. As shown in Fig. 1, we assume there are two-level hierarchies of caches and each level consists of more than one cache. Therefore, when a client requests a web object to its nearest proxy cache,

this proxy cache will try to find out whether it has that requested object or not. If the proxy cache does not have that object, then it will forward the request to the proxy cache that is at the next level of the hierarchy. This next level proxy cache will now look for that object within itself, and if the object is not found then that proxy cache will try to locate the object by searching it in the reference table it has maintained. Each level in the hierarchy except the lowest level maintains the reference table. This reference table contains the different fields specifying the location of the web objects. Two of the major fields are {Proxy ID} and {Object ID}. {Proxy ID} field represents the location of the web object while {Object ID} represents the web object itself. This reference table eventually contains a {Proxy ID} with respect to every {Object ID}. The description of various fields in the reference table is as follows:

- {Object ID}: A unique ID for all the web objects cached at the lower level proxy cache.
- {Proxy ID}: Specifies the proxy cache where a particular object is cached.
- {Date Created}: Specifies the exact date and time when an object is cached on the proxy cache from the original server.
- {Date Modified}: Specifies the recent date when that object is accessed.

Therefore, the second level proxy cache tries to locate the missed web objects using this reference table. However in case the reference table does not have any information about the requested web 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 of each level proxy cache gets updated down the hierarchy.

We need to clarify that a copy of the web object is not stored in any of the intermediate proxy caches as with traditional hierarchy architecture. Hence, our approach will overcome the problem of duplication of objects with the hierarchy architecture. In addition, in order to keep the freshness of the reference table, the reference of every web object that travels down the hierarchy from the original server is updated. In this way we can implement our new approach to a LAN. As WAN is a large network that consists of more than one LAN, we can introduce one more proxy cache located in next level hierarchy. This proxy cache will maintain one reference table to trace the requested web objects from the several LANs.

In our approach, when a new object travels down from the original server to the lowest level proxy cache on behalf of the client, the index of the object is stored at the upper level proxy cache. Hence, $False\ Misses$ are not introduced, because the reference table keeps an up to date picture of what has been cached in the lowest level proxy caches. However, $False\ Hits$ are possible especially when the lowest level proxy caches discard the object using a cache replacement policy in case the cache is full. The most straightforward method would be for each lowest level proxy cache to inform its upper level proxy whenever it discards a web object. However, in this case, the number of message exchanged between the proxy caches and the upper level proxy cache highly depends on the cache sizes of the lowest proxy caches. Another solution would be if the lowest level is full, the lowest level cache delivers the object to the upper level proxy before discarding that object. But, this also depends on the cache size of the upper layer proxy.

Our solution to avoid the $False\ Hits$ is for every proxy cache to keep an object with a flag bit. This flag bit is initially set to 1. A proxy cache can discard an object without informing its upper level proxy cache and can set this flag bit to 0. The upper level proxy cache always maintains {Proxy ID} which is the most recent proxy cache associated with {Object ID}.

Let us assume a proxy cache P_1 has an object O_1 whose flag bit is set to 1 and another proxy cache P_2 requests that object O_1. When the proxy cache P_2 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 O_1 and will forward that request to the proxy cache P_1. The proxy cache P_2 will now get a copy of the object O_1, and set its flag bit to 1. Based on this information, the upper level proxy cache updates its reference

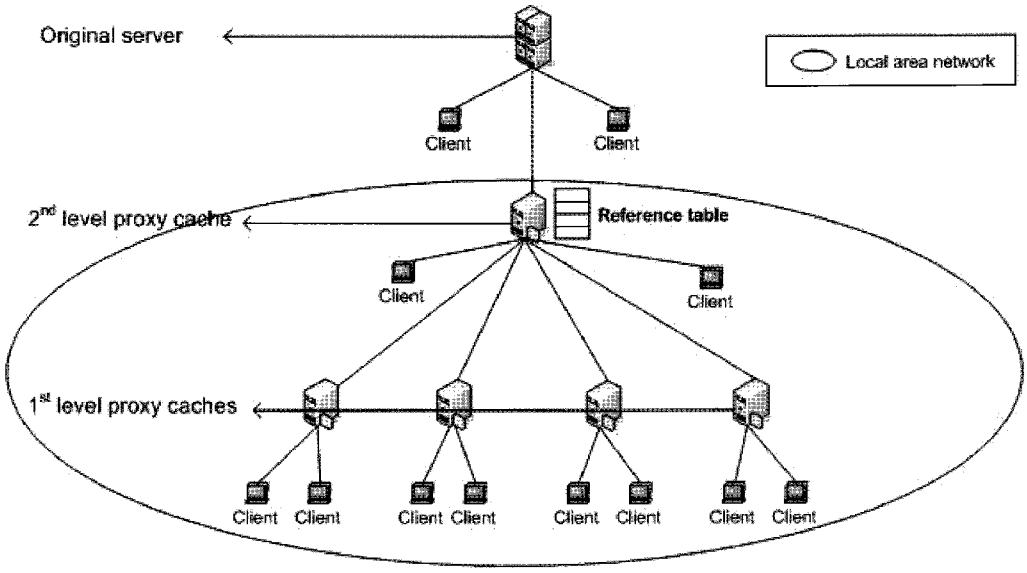

Fig. 1. Example of the hybrid architecture for a LAN

table reflecting that P_2 is the more recent proxy cache keeping the object O_1 than P_1. This will prompt the upper level proxy cache to update its reference table. Therefore this flag bit helps the upper level proxy cache to keep its reference table updated.

On the other hand, proxy cache P_1 will set its flag bit to 0. It can now discard the object O_1, because the reference table has been updated so it has no information about object O_1 cached in proxy cache P_1. 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 same level after referring to the reference table. The reference table gets updated as the object travels from one proxy cache to another.

However, after observing the users' access pattern, we have noticed that the possibility of requesting 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 (P_1 in our example) for the same object, and P_1 will have to contact another proxy cache (P_2 in our example) after contacting the upper level proxy cache to get the same object that was just discarded from its local proxy cache P_1. This simply increases the network traffic in between the lower level proxy and the upper level. It also introduces unnecessary delay in terms of response time.

If we keep the object even after it has been referenced by another proxy cache, there is wastage of space caused by duplicated objects over the some proxy caches in the same level. This penalty, however, can be compensated by the reduced network traffic and response time. The most promising solution is to find a way to decide when the object can be discarded without introducing a significant penalty. This can be done by analyzing how many objects are accessed again from the clients under the same lowest level proxy.

4 PERFORMANCE

To evaluate the performance of our proposed approach, we have developed a trace driven simulation program. This program simulates the various levels cache environments handling the requests generated by the clients at the lower level. It also simulates how the requests are forwarded from the lower level proxy

caches up to the original web server when a cache miss occurs at these levels of proxy caches. The results of the simulation 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. All the simulation results are based on a real log from the *National Laboratory for Applied Networking Research* (NLANR) [15] that has approximately 2.5 million requests.

In the simulation, the clients are assigned to various proxy caches depending upon their IP addresses. Hence, all the clients having a particular network address will correspond to the same first level 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 our architecture with the other architectures that use different protocols. As our architecture is using a reference table for keeping a reference to the various web objects, the change in the size of the reference table will affect the performance of our scheme.

From the traces, we get the average time for a local hit. Based on that time, we calculate two average times that will be used for our simulation. A_1 is the average total elapsed time in the case of a cache miss in the trace. T_1 is the average round trip time between the proxy cache and the original web server while T_2 is the average round trip time between the client and the proxy cache. A_1 is derived by the formula

$$A_1 = T_1 + T_2 \rightarrow \\ \rightarrow T_1 = A_1 - A_2. \quad (1)$$

Hence, we can use T_1 whenever there is both a lower level cache and a reference table miss. We define A_2 as the average total elapsed time between the client and the lower level proxy cache in case where a cache hit occurs in the trace. It will be evaluated by the formula

$$A_2 = T_2. \quad (2)$$

We will be using this time A_2 whenever there is a lower level cache hit. We now show how we can efficiently approximate our response time using the response time in the real trace. Let us define

- P_h as the event that lower level proxy hit occurs.
- P_m as the event that lower level proxy miss occurs.
- UP_h as the upper level proxy hit.
- UP_m as the upper level proxy miss.
- PT_h as the proxy hit in the trace. .
- PT_m as the proxy miss in the trace. .
- R as the response time in our architecture. .
- R_T as the response time in the trace. .
- $RTT(P_i, P_j)$ is the round trip time between any two proxy caches in the group. .
- $RTT(P_i, UP)$ is the round trip time between a proxy cache and its upper level proxy.

In reference to these definitions, we can calculate the response time based on the following six possible cases:

Case 1: If $((P_m \ \& \ UP_m) \ \& \ PT_m)$
$R = R_T$
Case 2: If $((P_m \ \& \ UP_h) \ \& \ PT_m)$
$R = RTT\ (P_i, P_j) + RTT\ (P_i, UP) * A_2$
Case 3: If $((P_m \ \& \ UP_m) \ \& \ PT_h)$
$R = R_T + T_1$
Case 4: If $((P_m \ \& \ UP_h) \ \& \ PT_h)$
$R = RTT\ (P_i, P_j) + RTT\ (P_i, UP) +$
$+ R_T$
Case 5: If $(P_h \ \& \ PT_m)$
$R = A_2$
Case 6: If $(P_h \ \& \ PT_h)$
$R = R_T$

After getting the response time for our approach, we now will be calculating the response time for ICP and then we will compare the results for both the approaches.

In the figure above we have 6 proxy caches and the upper level proxy cache is in the center of all the proxy caches. As we can see, all the proxy caches are connected using the star topology. Using a real trace, we evaluated the actual average round trip time which equals 282ms between the clients and lower level proxy cache. Based on this time, we have

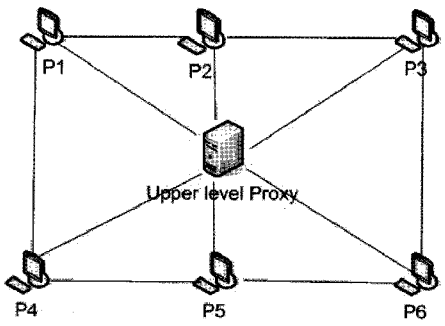

Fig. 2. Star network topology for evaluating response time for ICP

TABLE 1
Results of the local hit rate with varying cache sizes

Proxy Cache	Size of proxy cache in KB				
	20	30	40	50	60
P_1	3.65	4.26	4.97	5.39	5.84
P_2	3.86	4.96	5.91	6.67	7.24
P_3	2.45	3.33	4.00	4.56	5.12
P_4	2.76	3.69	4.39	5.24	5.79
P_5	26.49	31.05	33.56	34.88	37.80
P_6	3.27	3.77	4.58	4.73	5.11

approximately calculated the response time between various proxy caches. We assume the propagation delay between the proxy caches is at least five times longer than that of between the proxy cache and the client. Using all that data, we calculate the round trip time between the neighboring caches and hence the average response time for the ICP protocol. In addition to the parameters defined for our approach, we define the following parameters.

- G_h is the group hit.
- G_m is the group miss.
- $Max(P_i)$ is the maximum wait time for the response for proxy cache i.

Hence, proxy cache i has to wait $Max(P_i)$ time before requesting the object to the original server. Now we have six to calculate the response time for ICP.

Case 1: If $((P_m \ \& \ G_m) \ \& \ PT_m)$
$R = R_T + Max(P_i)$
Case 2: If $((P_m \ \& \ G_m) \ \& \ PT_h)$
$R = R_T + Max(P_i) + T_1$
Case 3: If $((P_m \ \& \ G_h) \ \& \ PT_m)$
$R = A_2 + RTT \ (P_i, P_j)$
Case 4: If $((P_m \ \& \ G_h) \ \& \ PT_h)$
$R = R_T + RTT \ (P_i, P_j)$
Case 5: If $(P_h \ \& \ PT_h)$
$R = R_T$
Case 6: If $(P_h \ \& \ PT_m)$
$R = A_2$

In order to find out the whether the requested object is present in the group or not, we will be using the reference table. We could get the real values for T_1 (= 897 ms), T_2 (= 282 ms), and A_1 (= 1179 ms) from the real trace. These values are plugged into the above formulas. We now show the results for our simulation and compare the results with the ICP. Depending upon the data we have, we calculated the local hit rate for our approach. Local hit rate means how many requested objects are available in the local proxy cache of the client. The results are given in Table 1.

The results for the local hit rate show that the hit rate increases as the cache size increases. As we can see, our approach works well in all the cases owing to our well defined reference table, but in case of ICP if we decrease the cache size the local hit rate will significantly decreases. Hence our approach fares well in case of small cache sizes. However, we need to mention that both approaches eventually show the same global hit rate in all different cache sizes.

When we compare the results for global cache hit rate within the group, we find out that the global hit rate is almost same for both the approaches. However, as we will see, our approach outperforms ICP in terms of response time and the number of query messages generated by each proxy cache. By using the formulas we described earlier, we compare the response time for both approaches and the following graphs are showing the results. As we can see, proxy cache 1 and 6 has 21 and 12 clients respectively and the response time for ICP is either same or more than the response time for our approach, because in case of ICP, each proxy cache i has to wait at least $Max(P_i)$ time before requesting the object to the original server.

We now compare the number of query mes-

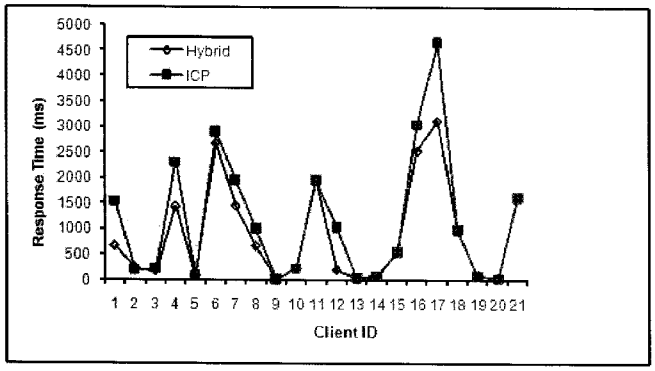

Fig. 3. Response times for the Proxy Cache 1

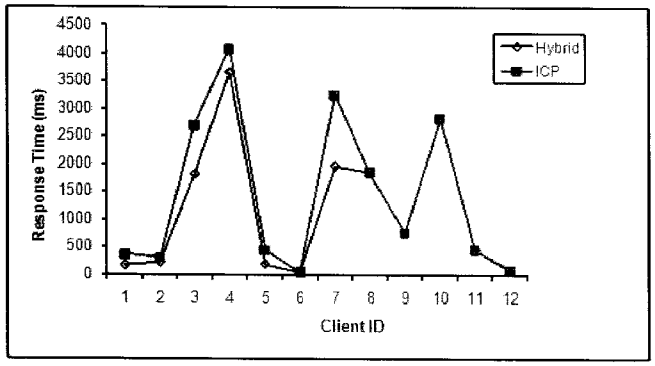

Fig. 4. Response times for the Proxy Cache 6

sages generated in both the approaches. The following table represents that the numbers of query messages generated in ICP are almost 3 times more than the number of query messages generated in the proposed approach.

We need to clarify it is very difficult to numerically compare the proposed solution to the *Hint cache* or *Home cache* because they should be implemented under quite heterogeneous architecture. However, we can technically compare our solution to the Hint cache. As we described in Section 2, *Hint cache* contains the hint field {objectID, nodeID} pair where nodeID identifies the closest cache that contains the copy of objectID. But when compared to our approach this protocol also suffers the same drawback as *Summary cache*. The *Hint cache* also suffers implementation complications and may sometimes contain an out of date picture of where the objects are cached thereby simply resulting in cache miss. Finally, we compare our new approach with the *Home cache*. *Home cache* is the latest and the most efficient approach of all the current protocols. In this protocol every web object has a designated home cache. If an object is not in the local cache, it will definitely be in the home cache. This protocol also uses the hashing technique to associate every object with its home cache. But in comparison to our hybrid caching, this protocol introduces some timing delays whenever the hashing technique is applied, and moreover this protocol does not perform as well in WAN as our new architecture.

5 Conclusion and Future Work

We proposed a new hybrid architecture for cooperative caching that efficiently overcomes the limitations of existing caching architectures. As our approach inherits the distributed and hierarchical architecture, it has reduced querying overhead in comparison to ICP, and eliminated the duplicity in comparison to CARP. Our approach also overcomes the design complexities of *Summary Cache* and *Home Cache* by employing the reference table.

As the performance of our protocol strongly depends on the cache size, reference table size, and replacement policy, we need more simulation to evaluate the performance of our protocol using various cache and reference table sizes. As a future work, we will also observe the performance with various replacement policies including FIFO, LRU, Greedy-Dual size, NFU, and OPT. Finally, we will compare the results with these from different protocols. In addition, in order to optimize on when the object can be discarded from the proxy cache after it is accessed by different proxy caches, we will analyze object access pattern using FI prediction algorithm [5] and how many objects are re-accessed at what intervals. We will also show how our solution can be efficiently extended to WAN environments. In addition, the location of each proxy server in our architecture needs to be strategically positioned for reliable multicast services [1], [2].

TABLE 2
Total number of query messages generated for both approaches

Proxy Cache (The number of messages)	Number of misses for different cache size in KB				
	20	30	40	50	60
P_1 (5248)	5056	5024	4987	4965	4942
P_2 (3208)	3084	3049	3018	2994	2976
P_3 (2193)	2139	2120	2105	2093	2081
P_4 (4707)	4577	4533	4500	4460	4434
P_5 (3746)	2754	2583	2489	2439	2330
P_6 (9681)	9364	9316	9238	9223	9186
Sum of Misses	26975	26625	26338	26175	25949
# of queries in ICP	161850	159752	158026	157049	155692
# of queries in Hybrid	53950	53250	52675	52349	51897

ACKNOWLEDGMENTS

This work was supported in part by the 2007 Winston-Salem State University Research Initiation Program (WSSU-RIP 2007) under Grant RIP-121203.

REFERENCES

[1] J. Baek and M. Kanampiu, *A NAK suppression scheme for group communications considering the spatial locality of packet losses*, International Journal of Computer Science and Network Security, vol. 6, no. 10, pp. 158-167, October 2006.

[2] J. Baek and J. F. Paris, *A heuristic buffer management and retransmission control scheme for tree-based reliable multicast*, ETRI Journal, vol. 27, no. 1, pp. 1-12, February 2005.

[3] G. Barish and K. Obraczka, *World Wide Web caching: Trends and techniques*, IEEE Communications Magazine, vol. 38, no. 5, pp. 178-185, May 2000.

[4] C. Bowman, P. Danzig, D. Hardy, U. Manber, and M. Schwartz, *The Harvest information discovery and access system*, Computer Networks and ISDN Systems, vol. 28, no. 1-2, pp. 119-125, December 1995.

[5] J. H. Case and P. S. Fisher, *Long term memory modules*, Bulletin of Mathematical Biology, vol. 46, no. 2, pp. 295-326, March 1984.

[6] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F.Schwartz, and K. J. Worrel, *A hierarchical Internet object cache*, Proceedings of the USENIX Annual Technical Conference, pp. 153-164, January 1996.

[7] L. Fan, P. Cao, J. Almeida, and A. Broder, *Summary cache: A scalable wide-area web cache sharing protocol*, IEEE/ACM Transactions on Networking, vol. 8, no. 3, pp. 281-293, June 2000.

[8] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, *Potential benefits of delta-encoding and data compression for http*, Proceedings of the ACM SIGCOMM, pp. 181-193, July 1997.

[9] N. Niclausse, Z. Liu, and P. Nain, *A new efficient caching policy for the World Wide Web*, Proceedings of the Workshop on Internet Server Performance, pp. 119-128, June 1998.

[10] R. Tewari, M. Dahlin, H. Vin, and J. Kay, *Design considerations for distributed caching on the internet*, Proceedings of the 19th International Conference on Distributed Computing Systems, pp. 273-284, May 1999.

[11] J. Wang, *A survey of web caching schemes for the Internet*, ACM Computer Communications Review, vol. 29, no. 5, pp. 36-46, October 1999.

[12] V. Valloppillil and K. Ross, *Cache array routing protocol v1.0*, Internet Draft, February 1998.

[13] D. Wessels, and K. Claffy, *ICP and the squid web cache*, IEEE Journal on Selected Areas in Communication, vol. 16, no. 3, pp. 345-357, April 1998.

[14] M. Zu and J. Subhlok, *Home Based Cooperative Web Caching*, Proceedings of the Seventh Multi-Conference on SCI, July 2003.

[15] *National lab of applied network research*, http://ircache.nlanr.net/. (Accessed January 2006).

Jinsuk Baek is Assistant Professor of Computer Science at the Winston-Salem State University (WSSU), Winston-Salem, NC. He is the director of Network Protocols Group at the WSSU. He received his B.S. and M.S. degrees in Computer Science and Engineering from Hankuk University of Foreign Studies (HUFS), Korea, in 1996 and 1998, respectively and his Ph.D. in Computer Science from the University of Houston (UH) in 2004. Dr. Baek was a post doctorate research associate of the Distributed Multimedia Research Group at the UH. He acted as a consulting expert on behalf of Apple Computer, Inc in connection with Rong and Gabello Law Firm which serves as legal counsel to Apple computer. His research interests include scalable reliable multicast protocols, mobile computing, network security protocols, proxy caching systems, and formal verification of communication protocols. He is a member of the IEEE.

Gurpreet Kaur was born in Punjab, India. She did her B.T. degree in Computer Science and Engineering from the Guru Nanak Dev Engineering College, India, in 2004. She was an international student at the Winston Salem State University (WSSU) and finished her M.S. degree in Computer Science in 2007. She is currently working as Software Consultant in Florida, USA. Her areas of concentration are computer networks, and data-warehousing.

Junghoon Yang was born in Seoul, South Korea. He was an exchange student at the Winston-Salem State University (WSSU) through the 7+1 program conducted by Hankuk University of Foreign Studies (HUFS). He worked as a software developer at the Interweb Software Inc., a system integration company in the area of e-procurement, for three and a half years. He was a member of the Intelligent Information Systems Group at the WSSU and received his B.E. degree in Computer Science and Engineering from the HUFS in 2008. He is currently working as a staff at the KPMG Korea. His areas of concentration are computer networks, network security, and programming languages.