A Scalable Architecture for Cooperative Web Caching

Riccardo Lancellotti[1], Bruno Ciciani[2⋆], and Michele Colajanni[3]

[1] Dip. di Informatica, Sistemi e Produzione, Università di Roma "Tor Vergata"
[2] Dip. di Informatica e Sistemistica, Università di Roma "La Sapienza"
[3] Dip. di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia

Abstract. Cooperative Web caching is the most common solution for augmenting the low cache hit rates due to single proxies. However, both purely hierarchical and flat architectures suffer from scalability problems due to cooperation protocol overheads. We present a new cooperative architecture that organizes cache servers in well connected clusters and implements a novel cooperation model based on a two-tier lookup process. The experimental results carried out on a working prototype show that the proposed architecture is really effective in supporting cooperative Web caching because it guarantees cache hit rates comparable to those of the most performing architectures and it reduces cooperation overhead at a small fraction of that of other protocols.

1 Introduction

Web caching has evolved as the first way to deal with performance and network resource utilization issues related to the growth of popularity of the World Wide Web. The idea is quite simple. Instead of connecting to a "far" and possibly overloaded Web server, the client request reaches a *proxy server*, that hosts resources frequently requested by a set of clients in a cache server "nearer" than the origin server.

The main problem of this approach is that the cache hit rate of one proxy server can be really low. The proposed solutions aim to establish interactions among various proxies. *Global caching* or *cooperative caching* architectures are used by public organizations (e.g., IRCache [1]), Internet Service Providers (e.g., AT&T [2]), third party companies, such as Content Delivery Networks (e.g., Akamai [3], Digital Island [4]). Cooperation among Web caches has been widely studied. For some recent surveys, see [5, 6] or [7, 8]. Cooperation among cache servers can occur for several reasons, but the most important motivations are: cache content lookup (*cooperative lookup*), data placement and document removal. In this paper, we focus on cooperative lookup. The two most popular approaches for cooperative lookup refer to a hierarchy of cooperating caches

⋆ Bruno Ciciani is currently Visiting Professor to the IBM Thomas J. Watson Research Center, NY

(*hierarchical architecture*) or to a flat cooperation topology (*distributed architectures*).

In hierarchical architectures a cache miss will result in looking for the resource to an upper level cache [9]. In distributed architectures every cache is supposed at the same level, and missed resources at one proxy are looked for in all cooperating cache servers. *Hybrid architectures* have been studied as well [6]. In this paper we refer to distributed architectures.

Any cooperation among a set of distributed servers has to decide a protocol and an implementation technique for exchanging some local state information which, in the case of cooperative lookup, basically refers to cache content (although other information can be useful, such as network and/or server load conditions). The two opposite approaches for sharing state information are well defined in literature: *on-demand protocols* in which state information exchanges occur only in response to a client request, and the family of *informed protocols* in which state information is exchanged periodically or when (significant) state modifications occur. ICP [10] and Summary Cache [11] are examples of the former and latter protocols, respectively.

The main drawback of any pure approach to Web caching cooperation is its lack of scalability because lookup latency and/or amount of data exchanges for cooperation augment dramatically for higher numbers of cooperating nodes. For example, *on-demand protocols* require a query/response message to/from each cache server for every local miss, moreover they risk that state information from "far" cooperative cache servers is received well beyond the chosen threshold (e.g., 2 seconds is the default limit for ICP). On the other hand, periodic exchanges of state information among all cache servers are impracticable when the number of cooperative nodes is high or there are many "far" nodes. In these instances, to limit protocol overheads, state refreshments should occur sporadically, but this would affect the nature itself of an *informed protocol* because of high risks of distributing stale state information.

The main claim of this paper, also validated through extensive simulations in [12], is that a better scalability for caching cooperation can be achieved only by architectures that use hybrid protocols. Indeed, each protocol has some tradeoff and its pros can be exploited by taking into consideration the network characteristics of the geographically distributed system. A multi-tier architecture may allow the cooperation of more cache servers with a potential increase of hit rate or permit a better usage of network resources by reducing the lookup time or the bandwidth overheads for coordination among cache servers. A two-tier architecture is the simplest form of a multi-tier architecture, where the basic concept is that cache servers can be distinguished between "near" (that is, well connected nodes) and "far" (that is, nodes connected through multiple network hops and/or through possibly highly loaded links). The two-tier approach aims at being more flexible and scalable than other cooperation mechanisms: first-level lookup cooperation occurs only among limited sets of "near" cache servers with the goal of minimizing lookup latency; second-level lookup cooperation in-

volves only some representative "far" cache servers with the goal of minimizing amount of exchanged information needed for cooperation.

Another advantage of two-tier cooperation derives from the logical distinction maintained between the two levels. In such a way, integration of existing distributed caches (e.g., a group of proxies cooperating with CRISP [13]) into a two-tier infrastructure becomes easier because such a group of cooperating caches can become a first tier cluster with little modifications.

The contribution of this paper is twofold. We propose the architecture of a new prototype of cooperative cache servers where the nodes are organized in clusters (intra-cluster and inter-cluster cooperation protocols has been implemented by modifying the Squid software [14]). Moreover, we present a set of experimental results showing that the proposed prototype offers a good hit rate with a cooperation overhead significatively lower than that of other cooperation mechanisms.

The rest of this paper is organized as following. Section 2 describes the intra-cluster and inter-cluster cooperation protocols that are modified versions of the 2TC protocol proposed in [12]. Section 3 discusses the architecture of the prototype by evidencing main modifications made in Squid. Section 4 describes the experiments and compares the results with those of other cooperative and non-cooperative protocols. Section 5 contains an overview of existing approaches to cooperation. Section 6 summarizes the results of this paper.

2 Two-tier Web caching architecture

The problem of lookup cooperation between proxy servers is essentially a problem of state information exchange within the components of a distributed system where the nodes are connected through heterogeneous links.

Let us consider an initial set S of proxy servers $\{P_1, \ldots, P_n\}$. Let C_i be a subset of S such that $\bigcup_{i=1}^{n} C_i = S$.

The number of cache servers inside C_i is chosen to allow cooperation without incurring in performance penalties caused by scalability issues. Moreover, consider that this clustering is done in a way such that cache servers inside the same cluster are "near", whereas cache servers belonging to different clusters are "far". The distance between nodes can be calculated by considering static (e.g., number of hops) or dynamic information (e.g., Round-Trip Time), although the concept of Internet vicinity is still an open issue. Figure 1 shows an example of clustering, where grouping corresponds to a *partition* of the set S. This is not a requirement for the proposed architecture because some overlapping is acceptable. However, for the sake of simplicity, in this paper we assume that the clusterization phase gives a partition of the available cache servers. The problem of finding an optimal clustering is out of the scope of this paper. It can be easily solved by management motivations (e.g., because the organization can find more convenient to place sets of cache servers in specific regions and none in others), by geographical considerations at different levels of granularity (for example, a coarse grain choice could be to consider four partitions of cache servers: East US,

West US, Asia, and Europe) or by more sophisticated optimization algorithms (e.g., the Goemans-Williamson algorithm [15]).

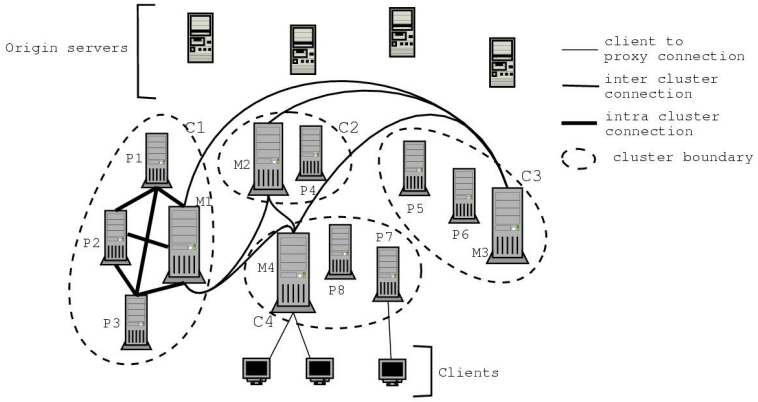


Fig. 1. Clustering of proxy server.

Intra-cluster cooperation is the most used mechanism, so it should be done through a protocol that offers a fast lookup, to keep the latency at acceptable levels. Assuming that the cache servers in the same cluster are well interconnected, the cost of frequent data exchanges occurring "off-line" is affordable. This makes the use of an informed protocol appealing because it offers a lookup extremely fast (if not false hits occur), typically done on some sort of hash table structure kept in RAM.

Inter-cluster cooperation occurs among "far" cache servers that are possibly connected through congested and limited bandwidth links. Hence, it is important to use cooperation protocols that limit the waste of network resources. An on-demand protocol could be a good choice because it is activated only consequently to a client request. Unlike an informed protocol, it works "on-line" and exchanges state information only when necessary. The drawback is that this choice tends to increase the latency time of an inter-cluster lookup.

In the following subsections we describe the intra-cluster and inter-cluster protocols implemented in the proposed two-tier Web caching architecture.

2.1 Intra-cluster cooperation

Intra-cluster cooperation is implemented through the Cache Digests protocol. This means that each cache server is informed about the resources stored in any other cache of the cluster.

A local miss in the first contacted cache server activates a lookup process that scans the local information about cooperating caches to check whether another node has the required resource. This cluster lookup phase can result in a *cluster*

hit or in a *cluster miss*. A cluster miss activates the inter-cluster cooperation. A cluster hit allows the retrieval of the object from another cache server of the same cluster. Figure 2 shows this process: the cache lookup on P_2 (step 1) returns a cache digest hit (step 2) and the resource is retrieved from cache P_3 through an HTTP connection (step 3).

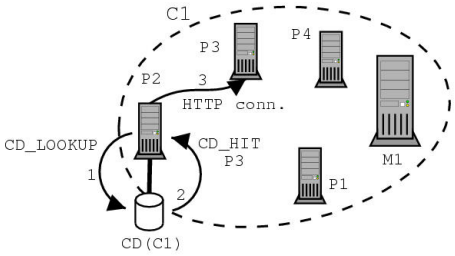


Fig. 2. Two-tier architecture: cluster hit.

In fact, the cluster lookup phase process can lead also to *false hit* or *false miss* because of stale state information or digest inaccuracy. To limit hit rate loss due to false positive or false negative, it is possible to augment the frequency of information exchanged among the caches of the cluster, and the precision (and the size) of exchanged digests. Anyway, the common rule is that the closer is the cooperation, the lesser nodes should be in the cluster.

In the case of false hit, the resource is fetched directly from the origin server to reduce latency time.

2.2 Inter-cluster cooperation

A cluster miss activates the inter-cluster cooperation. An on-demand protocol is used to contact cache servers belonging to other clusters. In the architecture described in this paper, inter-cluster cooperation occurs only among special cache servers, called *master caches*.

The reference architecture for two-tier cooperation is a mesh of peer cache servers: there are no parent nodes with respect to the retrieval of documents. A master cache acts as a parent for on-demand protocols: when a query must be sent to a cache server not belonging to the same cluster, the query is sent to the master cache of the cluster and the duty of contacting other clusters is left to it. This approach allows an easier configuration of the system of cooperative clusters: cache servers inside each cluster need to know only which caches belongs to the same cluster and which of them is the master. The master cache needs only one more information about which are the masters of the other clusters.

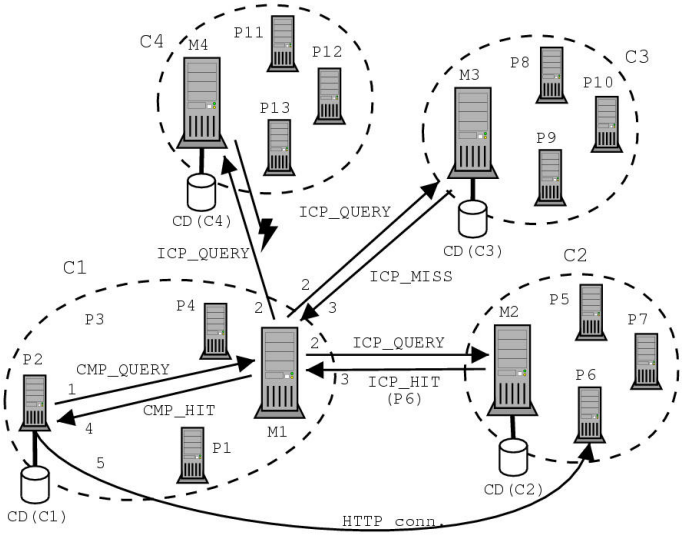


Fig. 3. Master proxy in the 2TC implementation

Figure 3 shows how a master proxy works: P_2 needs to use the second tier of 2TC because lookup in cluster cache digests returned a cluster miss. P_2 knows nothing about other clusters. Its only duty is to send a query to its master M_1 using CMP (Cache-to-Master Protocol, described in section 3.2) as shown in step 1. The master cache sends a query to all other masters through the ICP protocol (step 2). The masters of the other clusters check in their cache digests if the required resource is present in some cache server of its cluster and then returns a response (step 3) of hit (M_2) or miss (M_3). ICP uses a timeout mechanism to detect message loss (as for the one sent by M_4). The information is then sent back to the first proxy using CMP (step 4). In the case of global hit, the resource is retrieved through a normal HTTP connection (step 5).

Each master is informed about the content of the caches in its cluster because of the informed-based protocol used at the first cooperation level, so it can immediately return a message of cluster hit or miss. In the case of cache hit, the master returns also the address of the cache server that holds a valid copy of the requested document (in Figure 3 M_2 returns the address of P_6). It should be noted that the inter-cluster cooperation too is subject to false hit and false miss, because it relies on the informed cooperation protocol at the cluster level.

In summary, for each client request we can have one of the following four scenarios.

– **Local hit** when a valid copy of the requested resource is inside the first contacted cache server. The document is sent to the client and no cooperation is necessary.
– **Cluster hit** when the requested resource is found in the cache server of the same cluster ($P_j \in C_i$) to which the first contacted node belongs to. The intra-cluster cooperation protocol does not require message exchanges among caches during the lookup phase.
– **Global hit** when the resource is retrieved from a proxy $P_k \notin C_i$. As the intra-cluster cooperation determines a cluster miss, the inter-cluster cooperation protocol is activated. The hit is due to a cache server not belonging to the first contacted cluster.
– **Global miss** when the resource must be retrieved from the original server. Both cooperation levels fails in finding a valid copy of the resource in other cooperative cache servers or the first level incurred in a false hit.

3 Prototype implementation of the two-tier architecture

The two-tier cooperation architecture was implemented by modifying Squid 2.4 [14], a well known and widely used proxy server. The main modifications to the Squid software are localized in the modules called *peer selection* and *neighbors* [16]. The *peer selection* module contains the routines that selects a cache that may hold the requested resource and is used in the phase called *cooperation* in Figure 4. This phase is activated by a client request that results in a local miss or in a local hit with a stale file. The peer selection module uses other modules that implement some cooperation protocols, such as ICP and CMP.

The *neighbors* module contains the data-structure definitions and the routines to manage the database of cooperating proxies and the statistics about their state.

We also created a quite new module to support the Cache-to-Master protocol described in Section 3.2.

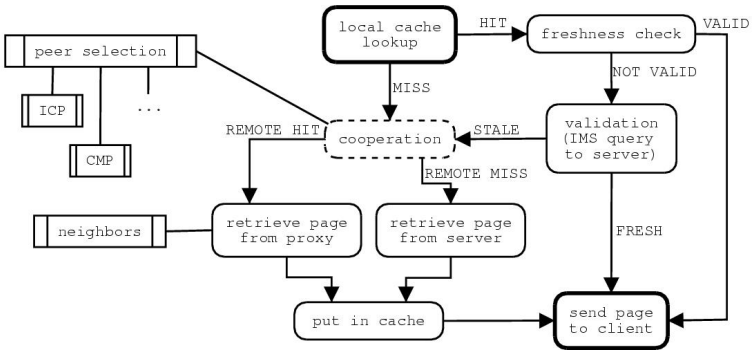


Fig. 4. Service of a client request by Squid.

3.1 Modifications to the peer selection module

The default Squid behavior for the cooperation phase shown in Figure 4 carries out a one-step lookup over the known active cooperating caches. This step can be done through multiple protocols such as ICP [10], HTCP [17], CRISP [18], depending on the compile-time options and on the configuration of the caches.

The main modification of our cooperative Web caching architecture makes this operation a two-step process: if the normal lookup fails (in the case of cluster-miss) a second step is activated, and the CMP protocol is used for inter-cluster lookup.

3.2 Cache-to-Master Protocol

The Cache-to-Master Protocol (CMP) is used to manage the intercommunication between the two levels of the two-tier architecture. CMP is essentially a modified version of ICP, designed to be lighter and simpler than the original ICP protocol.

Both ICP and CMP have a *sender address* field. However, its use is quite different. In ICP this field is considered untrusted, and the information is usually taken from the lower level protocols. In our prototype architecture, a master proxy can report a hit referred to another cache. Hence, in both CMP and ICP responses the sender address contains the IP address of the cache holding the requested document. This approach was preferred to that based on the ICP_OP_MISS_POINTER opcode. There are two motivations for this choice. The opcode is still experimental, and our choice kept our prototype simpler without disrupting its ability to cooperate with existing proxy servers. For example, hit referring to other caches are discarded by the standard ICP implementation as malformed messages, thus preserving the original ICP behavior.

The use of a master proxy to communicate with other clusters is a two-way process: both queries and replies pass through the master. This is necessary to avoid the risk of *cache poisoning*: cache hit in unknown proxies are trusted only if signalled by the cluster master, which is trusted.

3.3 Modifications to the neighbors module

The implementation of CMP required a modification of the *neighbors* module. For security reasons, Squid does not fetch resources from unknown proxies, but this would not allow inter-cluster cooperation.

Each proxy knows only the caches belonging to its cluster. With the modified module, when a CMP_HIT reply pointing to a previously unknown proxy is received, the module that manages this protocol calls a function in the neighbors module that dynamically adds a new entry to the list of known proxies. In such a way, the Web objects are always retrieved from known peers even if they come from caches of other clusters.

4 Experimental results

4.1 System configuration

The prototype of the cooperative Web caching architecture was tested on a cluster of nine PCs running Linux. Eight PCs hosted a proxy server and an HTTP workload generator. The nineth PC hosted an HTTP server. We used Web-Polygraph version 2.5 [19] as a workload generator. The results were collected from the Squid logs and do not refer to latency times. The proxies were configured in order to implement four cooperation mechanisms, namely *no cooperation*, Cache Digests, ICP and 2TC. The first was used as a comparison for the other scenarios. For Cache Digests and ICP the proxies were configured to let each cache cooperate with each other, while for the last architecture we organized the cache servers in two clusters each composed of four nodes.

To emulate a steady-state initial cache population, all experiments were done twice and collected information referred only to the last one.

4.2 Workload model

The workload model was based on that used in the second cache-off promoted by IRCache. We used a mix of content types made as following: images 65%, HTML documents 15%, binary data 0.5%, others 19.5%. Table 1 reports minimum, maximum and mean size for each type of object.

10% of requests were referred to *hot* resources. The hot set was 1% of the working set. Only 50% of the requests of each client was taken from a public set of pages, common to all clients. Each client was configured to visit more than once only 80% of the URLs. HTML resources could contain embedded objects.

Each proxy served requests coming from 5 clients. The working set of each cache was 47MB, corresponding to about 4250 URLs. It changed over time during the experiment, so that the global working set was 157 MB for each cache, corresponding to nearly 14200 URLs. The whole system of cooperating caches dealed with a 180.5 MB of documents, that generated over time a global working set of 601.8 MB. The caches were configured to hold no more than 30 MB of data each, so that each cache server could keep 5% of the global working set.

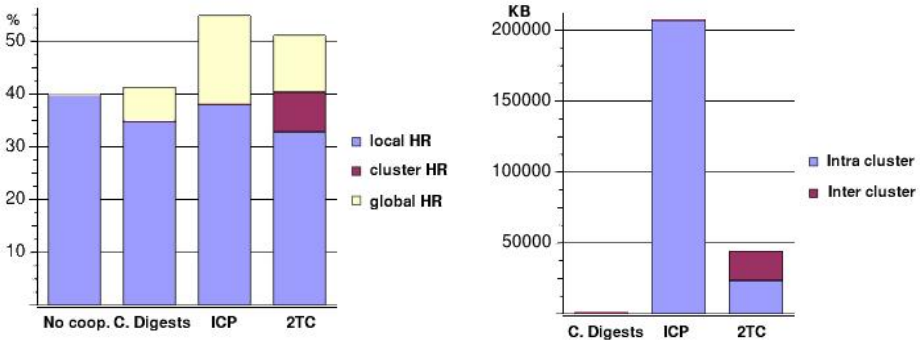
Type	Min size (KB)	Max size (KB)	Mean size (KB)
images	0.5	49	4.5
html	0.5	77.5	8.5
binary data	24	1577	300
other	7.5	89	25

Table 1. Hit Rate

4.3 Performance results

The experiments were focused on obtaining two performance indices, that is: *object hit rate*, and *protocol overheads* due to cooperation. It is worth to observe that no latency measure was collected, hence it was not a problem to use cache servers belonging to the same local network.

Figure 5(a) reports same indices contained in Table 2. The *object hit rate* (HR) was divided into *local HR* (clients requests that were served without any cooperation), *cluster HR* (when a cache server belonging to the same cluster provided the object), and *global HR* (when a cache server belonging to other cluster provided the object). Of course, cluster HR has no meaning for ICP and Cache Digests architectures because their cache servers are not organized in clusters. We found that the differences in the local hit rate were induced by changes in the access locality caused by the cooperation.



(a) Hit Rate

(b) Global traffic for Coordination

Fig. 5. Experimental results

Coordination	Local HR	Cluster HR	Global HR
No Cooperation	39.84	n/a	**39.84**
Cache Digests	35.57	n/a	**42.09**
ICP	37.99	n/a	**54.89**
2TC	35.28	42.83	**53.63**

Table 2. Hit Rate

The protocol overhead was measured both as an absolute value and as a per-request overhead. In Table 3, column 2 and in Figure 5(b), we reported the

traffic generated for coordination purposes. The traffic produced by 2TC was composed of traffic generated by Cache Digests exchange (3331.65 KB), CMP queries and replies (20259.35 KB) and ICP messages (20907.80 KB). In column 3 of Table 3, we compared how many bytes for each request were necessary for cooperation among cache servers. Considering that the request size was between 500 bytes and 1.5 MB, with a mean of 11 KB, the cooperation overhead for ICP seems acceptable in a well connected network environment but can be intolerable for a geographic environment.

Coordination	**KB for coord.**	**Bytes for coordination per request**
No Cooperation	0	0
Cache Digests	790	3
ICP	207064	811
2TC	44499	140

Table 3. Traffic for coordination

Our experiments show that the proposed cooperative Web caching architecture offers high hit rates, slightly lower than that of ICP, which gave the best hit rate in our test-bed scenarios. It is even remarkable that the proposed architecture introduces much less traffic (i.e., less than 20%) for cooperation than that needed by ICP.

These results demonstrate that the proposed cooperation protocol reaches its goal because it offers high cache hit rate with an overhead much lower than that of other approaches. This positive combination is the first step to guarantee scalability of the architecture when the number of cooperative cache servers augments significantly.

5 Related work

The issues related to cooperative cache lookup have been addressed in many ways, but the main philosophy behind them, except for a few exceptions, are two: on-demand protocols and informed-based protocols.

On-demand protocols are activated at lookup-time. They are typically designed to be fast and lightweight, usually relying on UDP messages. The most important of those protocol is ICP [10]. It was proposed as a part of the Harvest project and then adopted in many other proxy servers, such as Squid and NetCache. UDP is not a reliable protocol, so it can happen that a message is lost. ICP uses a timeout mechanism to detect packet loss and not well connected proxies.

ICP scales poorly: increasing the number of cooperating proxies leads to a quadratic increment of coordination traffic and increases the probability of packet loss (proportional to $1 - (1 - P_{err1})^N$ [20]), thus increasing the latency

time because of more requests being served only after the timeout has expired. Additionally, ICP does not support HTTP/1.1 caching directive semantics, so it is subject to false hit and false miss due to different freshness parameters among caches.

This last ICP problem was addressed by the HTCP protocol [17], which is more expressive, although more complicated than its predecessor. Scalability issues remains similarl to those that affect ICP.

Informed-based protocols uses a completely different approach to cooperation: information exchange occurs *before* the lookup phase, that in such a way becomes much faster than that of on-demand protocols. Information exchange can be periodic or synchronous. This latter form requires some message exchange at the occurrence of any new event and guarantees strong consistency. Nevertheless, it introduces big overheads, so that state information is usually exchanged in an asynchronous way, through some form of compression, even if this form of cooperation may cause stale state information. Particularly useful is a lossy compression called Bloom filters [21] used by Summary Cache [11] and Cache Digests [22]. There is a well known trade-off between cooperation effectiveness and scalability of informed-based protocols: to increase the first it is necessary to exchange state information more frequently and to use more accurate description of cache contents. This increases network resource usage and leads to scalability problems.

Pure versions of on-demand and informed-based protocols have one trait in common: both them performs the cooperation as a *one-step* process. A different scheme is proposed by CARP [23, 24] that uses an implicit cooperation that does not need any message exchange among the cache servers. The main drawback of CARP is its static nature, that makes this protocol not suitable when cache servers are geographically distributed and network status is subject to variations, as it is the typical case of Internet.

A hybrid cooperation protocol that is more related to the 2TC protocol is CRISP by Chase et al. [13, 18]. Similarly to 2TC, CRISP combines an informed protocol with a query approach to build a scalable cooperation mechanism. Unlike the cooperative Web caching architecture discussed in this paper, CRISP relies on a centralized directory that makes this architecture not scalable in a geographic network environment, as observed by the same authors [25].

6 Conclusions

This paper presents a novel architecture for cooperative Web caching based on a two-tier cluster-based lookup process. The prototype has been implemented by modifying the Squid proxy server. Experiments carried out through an artificial workload based on Polygraph show that the proposed architecture guarantees better scalability because its object hit rates are comparable to those of ICP, the best performing protocol (about 50%), but its overhead due to cooperation is much lower (140 bytes per request vs. 811 bytes).

References

[1] IRCache: Ircache project (1995) – http://www.ircache.net.
[2] AT&T: At&t (2002) – http://www.att.com.
[3] Akamai: Akamai inc. (2002) – http://www.akamai.com.
[4] DigitalIsland: Digital island inc. (2002) – http://www.digitalisland.com.
[5] Wang, J.: A survey of web caching schemes for the internet. ACM Computer Communication Review **29** (1999)
[6] Rodriguez, P., Spanner, C., Biersack, E.: Web caching architectures: hierarchical and distributed caching. In: Proc. of Web Caching Workshop (WCW'99). (1999)
[7] Rabinovich, M., Spatscheck, O.: Web Caching and Replication. Addison Wesley (2002)
[8] Wessels, D.: Web Caching. O'Reilly (2001)
[9] Yu, P.S., MacNair, E.A.: Performance study of a collaborative method for hierarchical caching in proxy servers. Computer Networks and ISDN Systems (1998) 215–224
[10] Wessels, D., Claffy, K.: Internet cache protocol (icp), version 2 (1997) RFC 2186.
[11] Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: A scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking **8** (2000) 281–293
[12] Santoro, A., Ciciani, B., Colajanni, M., Quaglia, F.: Two-tier cooperation: A scalable protocol for web cache sharing. In: Proc. of IEEE International Symposium on Network Computing and Applications, Cambridge, MA (2002)
[13] Gadde, S., Rabinovich, M., Chase, J.: An approach to building large internet caches. In: Proc. Sixth Workshop on Hot Topics in Operating Systems (HotOS-VI). (1997)
[14] Collins, R., Nordstrom, H., Russkov, A., Wessels, D.: Squid web proxy cache (2002) – http://www.squid-cache.org.
[15] Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. In: Proc. of ACM-SIAM Symposium on Discrete Algorithms. (1992)
[16] Wessels, D.: Squid Programmers Guide. (2002)
[17] Vixie, P., Wessels, D.: Hyper text caching protocol (htcp/0.0) (2000) RFC 2756.
[18] Gadde, S., Chase, J., Rabinovich, M.: A taste of crispy squid. In: Proc. of Workshop on Internet Server Performance (WISP'98). (1998)
[19] Russkov, A., Wessels, D.: Web polygraph (2000) – http://www.web-polygraph.org.
[20] Papoulis, A., Pillai, S.U.: Probability, Random Variables and Stochastic Processes. McGraw-Hill (2001)
[21] Bloom, B.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13** (1970) 422–426
[22] Rousskov, A., Wessels, D.: Cache digests. Computer Networks and ISDN Systems **30** (1998)
[23] Valloppillil, V., Ross, K.: Cache array routing protocol v1.0 (1998)
[24] Ross, K.W.: Hash-routing for collections of shared web caches. IEEE Network Magazine (1997)
[25] Rabinovich, M., Chase, J., Gadde, S.: Not all hits are created equal: Cooperative proxy caching over a wide-area network. In: Proc. of Third International WWW Caching Workshop. (1998)