# Deployment of Collaborative Web Caching with Active Networks

Laurent Lefèvre[1], Jean-Marc Pierson[2], and SidAli Guebli[2]

[1] LIP INRIA RESO, Ecole Normale Supérieure de Lyon
46, allée d'Italie, 69364 Lyon Cedex 07, France
laurent.lefevre@inria.fr
[2] LIRIS, INSA de Lyon
7, av. Jean Capelle, 69621 Villeurbanne cedex, FRANCE
jean-marc.pierson@liris.cnrs.fr

**Abstract.** Deploying "distributed intelligence" inside the network through the help of collaborative caches is a difficult task. This paper focus on the design of new collaborative web caches protocols through the help of active networks. These protocols have been implemented within the high performance Tamanoir[6] execution environment. First experiments on local platform are presented.[1]

## 1 Introduction

Nowadays, and for almost a decade now, the World Wide Web generates a huge amount of traffic from the heart of the network down to the home end-users. This amount of information have to be tackled efficiently in order to achieve global good response times. We examine in this contribution two tools to increase the potentiality of the current architectures. On one hand, cache techniques have been proved efficient since a long time to reduce the latency of the network as well as its bandwidth consumption. While the basic technique consists in keeping copies in a cache of the web documents closer to their clients, collaborative caches give the opportunity for different caches to share their content, increasing the global efficiency of the system. Efficient collaborative proxy caches have been deployed worldwide (for instance the well known Squid [12]), and any institution, company or ISP have deployed one (or many) today. They often consist in a specialized host either connected to the communication infrastructure (usually with high speed links) or on-board in the routers or gateways traversed by the web queries. Limited possibilities to change the behavior of the interconnection devices make it difficult to propose and deploy new caches inside the network. On the other hand, active networks allows to dynamically deploy new services in the networks, without interfering with the commonly used protocols. Their programming facilities offer new directions for protocol and network designer to apply new technologies without going through heavy standardization process.

[1] The authors wish to thank J.P. Gelas for his help with Tamanoir platform.

Our goal in this work is to link those aspects, that is to create a collaborative cache framework and to embedded it in an active network through adapted and usable services. The idea is to put the management of the collaborative cache schema in the access layer of the network, and to propose non-intrusive collaborative services. Some of the difficulty lays in the limited resources of the active nodes (in terms of CPU, memory and disk). We have thus to propose an adapted solution taking into account these constraints. The goal of our framework is to add some kind of intelligence in the routing of the web traffic and document caches replication in the edges of the network. We clearly benefit from active networks support by transparently deploying active caches through data path without modifying and re-configuring Web clients and servers. Moreover active caches services can communicate in point to point way through control communication channel between active nodes. We propose an implementation of these web caching active services through the help of the Tamanoir[6] execution environment and present first experimental results.

The remaining of this article is organized as follows : We first analyze in section 2 the work done for collaborative caching and we give the characteristics we want to offer. In section 3 we detail some fundamentals of the active networks and we exhibit the constraints we have to tackle for our framework. In section 4, we propose an adapted framework for collaborative caching in the context of active networks. We give in section 5 some implementation issues as well as results of experimentation on a Tamanoir platform. We finally conclude in section 6.

## 2 Collaborative Caching

Caching techniques have been used in many domains of computer science for several decades, with much work on Web caching [1]. Basically, caches copy data delivered by a server closer to the potential clients to answer data requests later in place of the server. The main expectations are the reduction of the data server load and the improvement of the latency time. A side effect is the reduction of the network load.

In [13], the authors describe the potential impact of the collaborative scheme for the benefit of the hit rate, through real proxy data analysis and an analytic model of web behavior. Several collaboration protocols between caches have been proposed : Hierarchical caches (like Harvest [4] and Squid [12]), Adaptive caches [14,8] and Geographical caches [7]. In this latter technique, the server pushes documents to geographically located proxies that serve groups of users.

Adaptive caching [14,8] explores the idea of creating interleaving meshes with caches. Caches are grouped to answer cooperatively the requests of the users. In this system, the groups dynamically adapt to the changes in the network, and membership of a cache in a group might be revoked if its membership does not give a sufficient benefit.

Hierarchical caching [4,12] proposes to create a hierarchy between the caches of the system. The bottom level is composed of caches close to the clients (for instance the caches incorporated in the web browsers). The next level represents institution caches. The upper level is close to the network backbones, managing several institution caches. Although some intermediary levels can be added, Wolman et al. [13] exhibit that using more than two levels do not give extra performances. In this approach, the basic idea is that a request to a document go up in the hierarchy until it retrieves the document. As explained in [11], some features limit their efficient deployment : (1) To grow up a hierarchy, the caches have to be placed at strategic points of the network, leading to a necessary high coordination between each peer of the system. (2) Each level introduces an additional delay. (3) The upper level of the hierarchy might represent a bottleneck. (4) Multiple copies of the same document might be present in the system.

All the collaboration schemas need a communication protocol between caches. The well known ICP (Internet Cache Protocol) [12] focuses on the efficient inter-cache communication module and mechanisms to create complex hierarchies. When a document is requested and not present in a cache, ICP multicasts a query to its neighbors (this neighboring has to be constructed beforehand). The extensibility of the protocol is a problem, since the number of messages increases with the number of collaborative caches in the system. In [5], Fan et al. show that the network traffic can increase up to 90%.

Other promising techniques for communicating information between caches are based on creating and maintaining summaries of the contents of these caches, communicating those, and performing finally a local decision, based on this knowledge of the distributed contents of the caches. Cache digest [10] and Summary cache [5] are such examples. While in Cache digest the caches exchange periodically their summaries (thus generating network load), a push strategy is used in Summary cache (reducing the traffic). The main problem with these approaches lays in the consistency of the summaries as compared to the actual contents of the caches. Indeed, the content of the summary might be partly false due to the dynamic of the documents in each cache. The challenge is to find the right parameters (how often to exchange summaries) to have a good consistency between the summaries and the actual contents of the cache.

One important point in the behaviors of the caches is the replacement strategy of the documents in the cache. While many exist in the literature [3], we will focus in this paper on the schema of collaboration between caches. Thus we will use in the experiment a simple LRU strategy (Least Recently Used) where the oldest document is removed from the cache first [2]. Note that this replacement strategy does not interfere with the remaining of the proposal.

Based on these related works, our proposal will rely on a mix of :

- hierarchical caching (with a limited two levels hierarchy),
- adaptive caching (the community of caches might be dynamic),
- and an adapted communication protocol using summaries.

[2] We also design content-aware strategy[9] where usages and contents of documents are used to determine their time to live in the cache, which is out the scope of this paper.

taking into account the constraints and benefits of using active networks. Thus, we now present briefly the active networks and their key features in the framework of this proposal.

## 3 Using Active Networks for Collaborative Web Caches

In order to efficiently deploy our Collaborative Web cache architecture we need an open network platform easily manageable and deployable. We based our deployment of Collaborative Web Caching infrastructure on the Tamanoir[6] execution environment developed in RESO - LIP laboratory. Its architecture is designed to be an high performance active router able to be deployed around high performance backbone. This approach concerns both a strategic deployment of active network functionalities around backbone in access layer networks and by providing an high performance dedicated architecture.

Tamanoir Active Nodes (TAN) provide persistent active nodes supporting various active services applied to multiple data streams at the same time. The both main transport protocol (TCP/UDP) are supported by the TAN for carrying data. We rely on the user space level of the 4 layers of the Tamanoir architecture (Programmable NIC, Kernel space, User space and Distributed resources) in order to validate and to deploy our active collaborative cache services.

Some of the difficulty lays in the limited resources we want to deploy on the active nodes (in terms of CPU, memory and disk). But, we clearly benefit from active networks support by transparently deploying active caches through data path without modifying and re-configuring Web clients and servers. Collaborative web caches services have been developed in Java (see Fig.2) inside Tamanoir EE. These active cache services can be dynamically modified (parent, child) and communicate in point to point way through control communication channel between active nodes.

## 4 Framework for Web Caching Services in Active Networks

The goal of our framework is to add some kind of intelligence in the routing of the web traffic in the edges of the network, that is in the active nodes. We define a communication protocol and a cache management schema that is light-weighted (in terms of CPU consumption and memory usage on the active nodes) and that does not ask for heavy data transfer (not to overload the network bandwidth).

We will address in the following two major points of our proposal : the location of the documents cached in the collaborative system, and the delivery of these documents to the end-user. Two questions will have to find solutions : (1) Where is a document in the collaborative caches system ? and (2) How do we exchange documents between caches ? The section 4.2 will address the first question while the section 4.3 will address the latter.

## 4.1 Overview of the Collaborative Caches

The caches are organized in a hierarchy of two levels. On the lower level are the so-called *children-caches*, that play the role of proxies for a community of users. On the upper level are the *parent-caches*, that play the roles of coordinators of a set of children-caches. For the sake of simplicity, we will use in the following either child-cache or child and parent-cache or parent. We are interested in this work in the efficient management of the caches in such a cache community, thus composed of one parent and many children, each child serving a community of end-users. In such a context, the parent of a cache-community is thought to handle web requests for a company or an institution, while the children are basically attached to a department in the company. We work in the framework of the access to the network, i.e. between the end-users and the backbone of the network.

We have not investigated in this work the collaboration between parents. Indeed, we believe that such a collaboration won't add to the efficiency of the system because cache parents are to be used in distinct institutions or companies. The actual price to pay to obtain documents from another cache community will be the same price than to obtain it directly from the original server.

The hierarchy of this architecture is used to share the knowledge of the contents of the documents in the caches of the community. We do not intend to cache the documents at the parent level, but only at the children level. Only the summaries of the content will have to be stored at the parent level.

## 4.2 Summary of the Cache Contents : The Mirror Table

Since we aim at providing the cache service on an active node where memory and disk space might be small, we have to define a compact representation of the contents of the caches. As in [5] and [10], we use the Bloom filter [2] technique to spare space with this representation.

**Bloom Filter.** This method is due to Burton Bloom in 1970 [2]. This probabilistic algorithm aims at testing membership of elements in a set (URL of documents presents in a cache, in our case) using multiple hash functions and a simple bit array. This algorithm consumes a fix small amount of memory regardless of capacity or usage of a cache.

Its principle follows : We first determine a size $m$ for the bit array $F$ (the filter, initially with 0 values) and a number $k$ of independent $h_i$ ($i \in \{1...k\}$) hash functions (those results vary from 0 to $m-1$). Adding a new element $a$ to the filter consists in computing $k$ values $\{h_1(a), h_2(a), ...h_k(a)\}$, representing $k$ positions in the $F$ filter. These positions are then set to 1 in the $F$ filter. Note here that one particular bit of the $F$ filter might be set to 1 by more than one element. Note also that whatever number of elements have to be filtered, the size of the bit array does not vary, thus the compact form of representation is assured (and controlled).

To test if an element $a$ is present in the set, we first compute the $k$ values $\{h_1(a), h_2(a), ...h_k(a)\}$. If one of the corresponding bits is set to 0, then $a$ is not in the set (the search is definitely a MISS). If all the positions are set to 1, $a$ is present with a certain probability (this is common when using hash functions). That means the bits might be set to 1 while the element is not in the set. In [5], the authors show that the probability of such an error can be minimized with a value of $k = \frac{m}{n} ln2$, where $n$ is the number of elements in the set.

We use the following four definitions in the rest of the paper :

– True HIT : the filter correctly predicts that an element is in the set.
– False HIT : the filter predicts that an element is in the set, but the element is not actually in the set.
– True MISS : the filter correctly predicts that an element is not in the set.
– False MISS : the filter predicts that an element is not in the set, but the element is actually in the set.

The False MISS should not occur with the Bloom filter technique when the filter and the set are strongly linked (for instance when a new document enters the set, the filter updates its bit array). The False MISS problem occurs when the filter and the set are not updated together, for instance when they are located on different hosts, as we will see later in section 4.4.

The False HIT can occur at any time, even with a strong consistency policy. Thus the problem will be fully addressed in section 4.4 and a specific protocol organized to maintain at best cost the consistency.

**Mirror Tables.** The information sent from the children caches to the parent is compacted in a bloom filter used by the parent to localize the documents in the collaborative community.

We define a *mirror table* as a compact structure representing (reflecting) the content of a child-cache. It contains the following fields : an identifier of the cache reflected by this mirror table (an IP address for instance), a bit array (for the bloom filter $F$), the size of the bit array ($m$), the number of hash functions ($k$) and the number of false HIT.

A parent keeps one such mirror table for each of its children. While it is easy to add new elements in the filter, to delete an element is impossible. Indeed, one particular bit may have been set to 1 by more than one element. Thus, if we put a bit to 0 when an element is suppressed, we may make the filter inconsistent. Doing nothing is obviously not good : If the element we intend to suppress was the only one to have set the bit to 1, the cache is inconsistent, too. A solution here should be to use a counter telling us how much time a bit has been set to 1, instead of only a meaningless 1. This solution increases the size of the mirror table but decreases the number of false HIT. Our proposal is to handle modified mirror tables at the children caches, where the bit array is replaced by a counter array of the same size $m$. When a document is added (resp. suppressed), the counters for each position set by the hash functions are increased (resp. decreased) instead of setting a simple 1. From this local table,

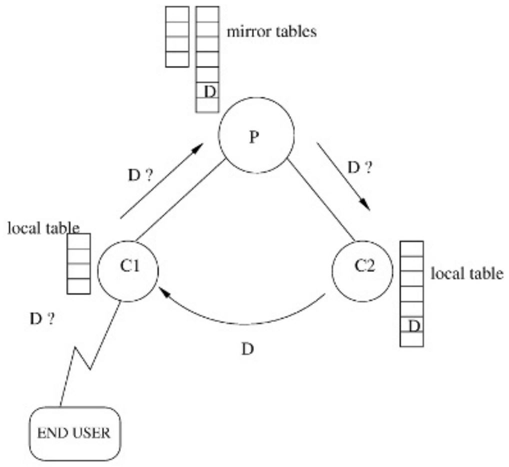


**Fig. 1.** Inter cache protocol, when the requested document D is in the collaborative system

the child simply constructs the filter when needed : When the counter is 1 or more, the corresponding entry in the filter is set to 1.

The parent has copies of the filters present in its children. From the parent point of view, when the filter becomes inconsistent (the number of false HIT becomes very high), this parent asks the corresponding child for a new version of its filter. Conversely, when a child has made a lot of changes in its filter, it takes the local decision to send it to its parent (see section 4.4). Note here that only the filter (the bit array) is sent from one child to its parent. With this mechanisms, only a small amount of information has to be stored at the parent, and the resulting traffic for update is kept reasonable (the bit array, that means $m$ bits). On a child, the consistency of the filter is kept relevant at any time.

### 4.3 Location and Delivery of a Document

Figure 1 illustrates the inter-cache protocol we propose. When contacted by an end-user browser for a document $D$, a child $C1$ examines its own cache. If $D$ is present locally, $C1$ serves directly the end-user. Otherwise, $C1$ forwards the requests to its parent $P$. This one checks its mirror tables (step 1) to detect if one of its other children has the document $D$ (with a certain probability). Two scenarios can occur :

– if none of the brothers of $C1$ owns the document (from the parent knowledge, extracted from the mirror tables), the request is forwarded to the original server. This one sends $D$ back to $P$, who forwards it to $C1$. $C1$ caches $D$, updates its local table and sends $D$ to the end-user;

– if a child $C2$ seems to have $D$ (from $P$ point of view)[3], $P$ sends a control message to $C2$ in order to allow the transfer of $D$ from $C2$ to $C1$. If $C2$ really owns $D$, a peer to peer communication is set up between $C2$ and $C1$. Otherwise, a false HIT occurred : $C2$ informs $P$ of that false HIT. $P$ increases the corresponding field in its mirror table, and generates a new check of its mirror tables (back to step 1), excluding $C2$ from the search. The peer to peer communication between $C1$ and $C2$ permits to decrease the amount of work and number of messages involved at the parent level.

### 4.4 Consistency of the Information

The size of caches are limited, so a replacement schema is used to control them and delete documents. This behavior, inherent to the techniques of caching, involves in our case an inconsistency between the mirror tables at the parent and the actual contents of the caches at the child. This will generate some false HIT in the system, leading to unnecessary communications between parents and children.

To minimize this problem, we introduce two mechanisms : at the parent level, we maintain for each child the number of false-HIT generated during the communications with it. When this counter reaches a given validating threshold $\alpha$, the parent asks the child for a new version of its filter, considering its mirror table as obsolete. This solution is acceptable in the general case, when the parent often communicates with its children.

However, when a child is less contacted by its parent, the content of its cache might evolves locally. In this scenario, the content of the mirror table at the parent is not consistent. The child must anticipate some possible false MISS and false HIT and send a new version of its filter to its parent, when the number of changes in the local table becomes greater than a validating threshold $\beta$.

Note also that the problem of the false HIT is inherent to the Bloom filter technique : it is the price to pay to compact the information. Only a good choice in its parameters $k$ and $m$ (see section 4.2) can decrease the number of false HIT.

We propose therefore two protocols (and two parameters $\alpha$ and $\beta$) to handle the consistency of the information at the parent level based on both pull and push strategies. When either the parent or the child observes a high probability of dysfunction in the system future, it initiates the refreshing of the filter.

## 5 Implementation Issues and Experiments

### 5.1 Functional Architecture of a Collaborative Cache Active Node

Design and implementations of cache techniques have been made trough the implementation of active services All web browser clients are configured to send

[3] Note that more than one child-cache might have $D$; a LRU (Least Frequently Used) algorithm is used to load balance the queries among the candidates

their HTTP requests through a children cache of our system. This cache deploys active HTTPHandlerS service to handle this request.

Collaborative caches services are dynamically deployed on Tamanoir active node depending on cache configuration (parent, child). Figure 2 describes features embedded in active node to support cache collaborations.

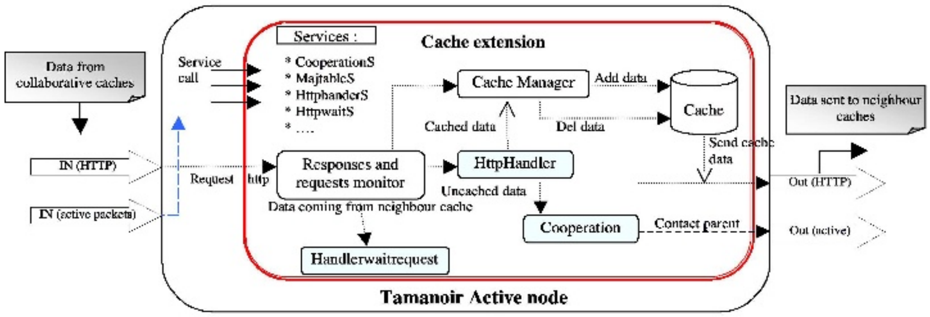


**Fig. 2.** Active Web cache architecture

## 5.2 Construction of the Collaborative Community

In this section we address the different methods to create the collaborative cache systems. The active nodes dynamically decide if they run a parent-cache or a child-cache.

In a fully static method, each node in the network willing to join the system must have some information about it. For instance, a node willing to become a new child must know one parent and use its *JoinparentS* service to contact this parent. Additionally, a parent (which is an active node) can ask another node to act as a child dynamically, by sending an active packet (thus sending a *JoinparentS* service this node can use to join the system). This situation can occur if the parent receives/sends some web traffic from/to another active node actually not in the system. This dynamic inscription is made possible by the active networks characteristics.

In our system, at least one node needs to be configured as a parent-cache.A full automatic deployment of this system would require that the parent also might be dynamic or change dynamically in a collaborative community : Indeed, the first parent-cache node deployed (statically or dynamically) may be a poor candidate for such a role and another node (better connectivity, more memory, more CPU) might take this role. This future work will have to be investigated.

## 5.3 Test Architecture

We experiment our collaborative cache solutions on an active networks platform based on Tamanoir Execution Environment. Our experiments are based on log files extracted from a real proxy server[4]. Different logs are used with different caches.

```
Time    elapsed    remotehost    code/status    bytes
1035849606.566 394 252.183.145.92 TCP_CLIENT_REFRESH_MISS/200 1160

method   Ur    rfc931    peerstatus/peerhost    type
GET http://br.yimg.com/i/br/cat.gif - DIRECT/200.185.15.91 image/gif
```

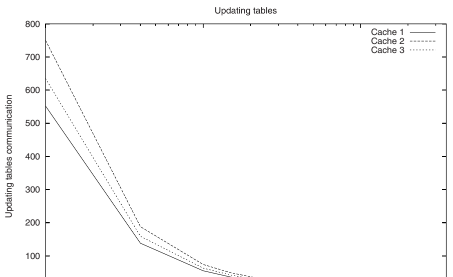
**Fig. 3.** Log example

## 5.4 Results

We define *Quasi Hit* operations in order to evaluate improvement provided by caching collaborative actions. A quasi hit occurs when a cache is able to download requested documents from one of its neighbor.

In our following experimental evaluations, we fix the mirror size for all caches. We choose 4 hash functions for the bloom filter. Filter size is also fixed at the optimal value of 8000 bits (as explained in section 4.2.1, filter size is derived from the number of hash functions and the number of documents in the set (1766 in our experiments)). False hit probability is then 0.11.

We first consider caches with unlimited size (they can cache all data requested during the experiment). Figure 4 shows the impact of validating thresholds ($\alpha$ and $\beta$) value on updating table operations and communication. $\alpha$ and $\beta$ follow the same value. Performances increase with threshold value but consistency between tables is not supported.



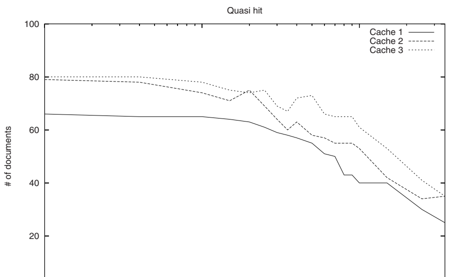


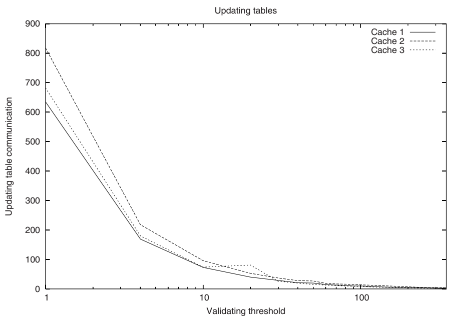
**Fig. 4.** Updating tables - unlimited cache **Fig. 5.** Quasi hit with unlimited cache

We illustrate the impact in terms of table coherence through the Quasi Hit effect (Figure 5). When the validating threshold increases, quasi hit rapidly decreases and we lose the benefits of collaborative caches due to a weak consistency of the tables. This experiment is done on 3000 requests on 1766 different documents. Local Hit of each cache is constant (cache1=439, cache2=221,

[4] Trace logs available on http://www.ircache.net

cache3=339). Local hit represents the number of requests to document in the cache. Note here that the total number of document in the cache might be larger than this, since some documents might be put once in the cache and never accessed later on.

Within experiments presented in figures 6 and 7, we deploy collaborative caches with limited resources (10% of needed storage). We observe cache replacement effect through a simple LRU policy. Local Hit decreases compared to unlimited caches experiments but remains constant (cache1=355, cache2=149, cache3=291). Quasi hit increases compared to unlimited cache experiment. But we show that quasi hit results remain low compared to the small available amount of cache. Due to the limited size of the cache, the local hit decreases as compared to unlimited caches : Indeed some requested documents , that had been already requested (thus potentially in the cache), might have been replaced by the replacement strategy. The combination of addition and suppression of documents in the cache allows in the limited cache experiment to update more often the mirror table than in the unlimited case, leading to more accurate consistency between parent and children mirror tables.



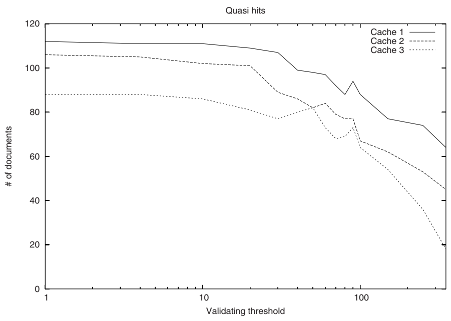


**Fig. 6.** Updating tables with limited cache

**Fig. 7.** Quasi hit with limited cache

We can note on figure 7 that the amount of quasi hit increases compared to unlimited cache experiments. This result benefits from the updating of mirror tables after document removal (these updating operations are generated through depending on the $\beta$ threshold).

## 6 Conclusion

Collaborative web caches allow to quickly find a requested document in a community of distributed caches while avoiding that parent caches keep a copy of documents. This solution greatly improves performance of document localization compared to hierarchical caches. In this paper, we present our first step towards the design of new collaborative web caches protocols. Deploying and maintaining collaborative caches is pretty difficult in IP networks. With the help of active networks approach, these caches can be easily managed and the deployment of cache policy can be dynamically broadcasted.

Caching documents inside a network requires high level and efficient intelligence support inside network equipments with limited resources. Moreover we needed an high performance network execution environment able to support multiple services dealing with web streams based on TCP transport layer. Through the help in terms of dynamicity and easiness of implementing new services, we benefit from the high performance Tamanoir active node framework. This paper presents our first experiments in evaluating our collaborative cache protocols. We are currently more evaluating performance aspects of our cache collaboration.

## References

1. G. Barish and K. Obraczka. World wide web caching: Trends and techniques. *IEEE Communications Magazine Internet Technology Series*, 38(5):178–184, May 2000.
2. B. Bloom. Space/time trade-offs in hash coding with allowable errors. *CACM*, 13(7):422–426, July 1970.
3. L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the INFOCOM '99 conference*, March 1999.
4. A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *USENIX Annual Technical Conference*, pages 153–164, 1996.
5. L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.
6. Jean-Patrick Gelas, Saad El Hadri, and Laurent Lefèvre. Towards the design of an high performance active node. *Parallel Processing Letters*, 13(2), jun 2003.
7. J. S. Gwertzman and M. Seltzer. The case for geographical push-caching. In IEEE, editor, *Proceedings Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, pages 51–55, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1995. IEEE Computer Society Press.
8. T. Lanmbrecht, P. Backx, B. Duysburgh, L. Peters, B. Dhoedt, and P. Demeester. Adaptive distributed caching on an active network. In *IWAN 01*, Philadelphia, USA, 2001.
9. J.M. Pierson, L. Brunie, and D. Coquil. Semantic collaborative web caching. In *Web Information Systems Engineering 2002 (ACM/IEEE WISE 2002)*, pages 30,39, Singapore, dec 2002. IEEE CS Press.
10. A. Rousskov and D. Wessels. Cache digests. *Computer Networks and ISDN Systems*, 30(22–23):2155–2168, November 1998.
11. J. Wang. A survey of Web caching schemes for the Internet. *ACM Computer Communication Review*, 25(9):36–46, October 1999.
12. D. Wessels and K Claffy. ICP and the Squid Web cache. *IEEE Journal on Selected Areas in Communication*, 16(3):345–357, April 1998.
13. A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *Symposium on Operating Systems Principles*, pages 16–31, 1999.
14. H. Zhang, H. Qin, and G. Chen. Adaptive control of chaotic systems with uncertainties. *Int. J. of Bifurcation and Chaos*, 8(10):2041–2046, 1998.