Cooperative Architectures and Algorithms for Discovery and Transcoding of Multi-version Content

Claudia Canali
University of Parma
claudia@weblab.ing.unimo.it

Valeria Cardellini
University of Roma “Tor Vergata”
cardellini@ing.uniroma2.it

Michele Colajanni
University of Modena and Reggio Emilia
colajanni@unimo.it

Riccardo Lancellotti
University of Roma “Tor Vergata”
riccardo@weblab.ing.unimo.it

Philip S. Yu
IBM T.J. Watson Research Center
psyu@us.ibm.com

Abstract

A clear trend of the Web is that a variety of new consumer devices with diverse computing powers, display capabilities, and wired/wireless network connections is gaining access to the Internet. Tailoring Web content to match the device characteristics requires functionalities for content transformation, namely *transcoding*, that are typically carried out by the content Web server or by an edge proxy server. In this paper, we explore how to improve the user response time by considering systems of cooperative edge servers which collaborate in discovering, transcoding, and delivering multiple versions of Web objects. The transcoding functionality opens an entirely new space of investigation in the research area of distributed cache cooperation, because it transforms the proxy servers from content repositories along the client-server path into pro-active network elements providing computation and adaptive delivery. We propose and investigate different algorithms for cooperative discovery, delivery, and transcoding in the context of edge servers organized in hierarchical and flat peer-to-peer topologies. We compare the performance of the proposed schemes through ColTrES (Collaborative Transcoder Edge Services), a flexible prototype testbed that implements all considered mechanisms.

1 Introduction

The Web is rapidly evolving towards a highly heterogeneous accessed environment, due to the variety of new consumer devices that are increasingly gaining access to the Internet. The emerging Web-connected devices, such as handheld computers, personal digital assistants (PDAs), mobile phones, and other pervasive computing devices, that will be the predominant fraction of Internet clients in a few years, differ considerably in network connectivity, processing power, storage, display, and format handling capabilities. Hence, there is a growing demand for solutions that enable the transformation of Web content for adapting and delivering it to diverse destination devices.

The process of converting a multimedia resource from one form to another is called *transcoding*. It can be applied to transformations within media types (e.g., reducing the image size, transforming from high-fidelity JPEG to low-fidelity GIF format), across media types (e.g., speech to text, video item to image set) or to both of them. The use of XML and XSL, that allow the same content to be presented on a variety of Web-connected devices through structured XML documents (with the appropriate XSL style-sheets) facilitates the presentation of multiple versions of a Web site content. However, it does not eliminate the need for content transcoding. The existing approaches to deploy Web content adaptation fall into three broad categories depending on the entity that performs the adaptation process [1, 10, 12]: *client-based*, *edge-based* (also called *proxy-based*), and *server-based* adaptation.

In both edge- and server-based adaptation, specific information about the delivery context can be associated with the client request, such as device capabilities, network constraints, user preferences [1]. The server-based approach [14] is designed to add content adaptation by extending the functionalities of a traditional Web server. The edge-based approach [8, 9, 13, 17] uses the proxy server, that is typically required by the client device to access the Web, to analyze and transcode the content on-the-fly, before delivering the result to the user. This component is often called edge server because in the delivery chain between the client device and the content server it is typically located close to the client. The edge server has also another important role. It can cache the results of transcoding, thus avoiding some round-trips to the content server and subsequent transcoding operations when transcodable content can be served from the cache [8, 13, 17]. Moreover, intermediate adaptation can shift load from content-providing Web servers and simplify their design, as it offloads the transcoding task to the intermediary infrastructure. The edge-based approach is also typically used to address dynamic variations in network traffic and to reduce end-to-end response time, especially for mobile devices which have reduced last-hop bandwidths [9]. A transcoding service located at intermediary points of the network infrastructure can also tailor contents coming from different content servers. So far, the edge-based approach to content adaptation is typically carried out by some edge server that directly connects to the clients.

As the transcoding operations may be computationally expensive, the edge solution based on single servers may be subject to a limited scalability [12]. A possible solution is to replace the edge server with a cluster of locally distributed servers [8]. Although this approach may solve the CPU-resource constraint, it does not exploit the potential benefits deriving from an increased cache hit rate and may move the bottleneck from the CPU of the edge server to the interconnection of the cluster. In this paper, we explore a different alternative that considers a distributed system of cooperative edge servers which collaborate in discovering, transcoding, and delivering Web content. The goals are to reduce the user response time and bound its variability. The motivation for a cooperative architecture is that the cost of transcoding can be notably reduced by discovering the desired resource in other servers and also by involving some other servers to perform the (possibly expensive) task of content adaptation.

Although the cooperative transcoding is based on distributed schemes, it is not a simple extension to cooperative caching. The following issues should be addressed in order to achieve a suitable solution.

- The presence of differentiated client device types requires to address various issues related to recognize, discover, and cache the multiple variants of the same resource that are obtained by transcoding operations.
- The transcoding process is expensive in terms of computing resources. Issues related to workload distribution, which are not usually considered in Web caching, can become of fundamental importance in the case of cooperative transcoding.

In this paper, we propose and investigate some architectures and algorithms for cooperative discovery, transcoding, and delivery, including servers organized in hierarchical and flat topologies. All considered schemes can be implemented on the existing Web infrastructure. We compare their performance through a prototype called ColTrES (Collaborative Transcoder Edge Services), which is a flexible testbed based on Squid. The ColTrES prototype implements all considered mechanisms by extending the traditional cooperative caching systems to an environment characterized by heterogeneous client devices. The first extension enhances the capabilities of a traditional cache server, and transforms it into an active intermediary server that not only caches Web objects but also transcodes them and stores the results [13, 17]. The second novel extension allows the cooperation of the active intermediaries. We take advantage of the scalability opportunities provided by cooperation to reduce the response time experienced by the users of a heterogeneous client environment.

We are not aware of any other research work dealing with the study and implementation of cooperative transcoding and caching systems with both hierarchical and flat topologies. The authors have obtained some preliminary results through simulations of a cooperative hierarchical scheme in [4], and some experimental results referring to one cooperative algorithm on flat topologies, which demonstrate the reduction of the user response times achievable with respect to a system of non-cooperative servers [3]. New cooperative transcoding algorithms and architectures are considered here. Through our prototypes, we demonstrate that all proposed algorithms and cooperative architecture are immediately applicable to the Web infrastructure. The real test-beds allow us to evaluate the reduction of the user response time achievable by different cooperative discovery and transcoding schemes. Moreover, we clearly demonstrate the advantages of cooperative transcoding through flat topologies over hierarchical schemes.

The rest of this paper is organized as following. Section 2 discusses the requirements and the operation mechanisms of a caching system for cooperative transcoding. Sections 3 and 4 explore different topologies and protocols for cooperative discovery, transcoding, and delivery. Section 5 proposes some transcoding algorithms applicable to flat topologies. Section 6 outlines the main characteristics of the ColTrES prototype. Section 7 describes the workload model that we have used to exercise the prototype. Section 8 presents the experimental results. Section 9 concludes the paper with some final remarks.

2 Edge server cooperative transcoding

In this paper, we consider that cooperation can be established along a vertical direction, namely hierarchical Web caching, descending from the Harvest project, or along an horizontal direction, namely flat or distributed Web caching [15]. In this section we describe the main operations involved in a system of cooperative transcoding and caching servers, while the specific features of the cooperation topologies and protocols are outlined in Sections 3 and 4.

Depending on the cooperation architecture, a server node may have one or more functionalities that is, it can act as a *transcoder*, a *cache* and/or an *edge* server. All nodes have caching functionality and hence are *cache* servers. *Edge* servers receive requests directly from the clients and deliver the requested resources. *Transcoder* servers can perform content adaptation.

The organization we propose in this paper is an extension of the traditional cooperative caching systems for multiple reasons. First of all, multiple variants of the same resource can be present at the same time in the edge servers. Moreover, the server nodes are not only object repositories, but they can also perform the transcoding process that is a typical computationally intensive task.

The features of client devices vary widely in screen size and colors, processing power, storage, user interface, software, and network connections. The client may include the resource data type it can consume as a meta-information in the HTTP request header. Recently, the WAP Forum and the W3C have also proposed the compatible standards CC/PP and UAProf for describing the client capabilities [1]. Hereafter, we will refer to the information describing the capabilities of the requesting client as the *requester-specific capability information* (RCI). An object which has been previously transcoded may be further transcoded to yield a lower quality object. In particular, each version may be transcoded from a subset of the higher quality versions. Different versions of the same object (and the allowed transcoding operations among them) can be represented through a *transcoding relation graph* [4].

In a cooperative transcoding scheme, we can identify three main phases that may require some cooperation among the server nodes, namely *discovery*, *transcoding*, and *delivery* phases. Even the traditional phases differ from the corresponding phases of a standard cooperative caching scheme. We describe the three phases in a reverse order.

Once an exact version of the requested object is found (or generated), the *delivery* phase transfers the resource to the client. Although for some applications it is acceptable to satisfy a request with a lower-quality resource than that specified by the client, we do not consider this possibility in this paper. The final delivery is always carried out by the edge server that is first contacted by the client. Hence, if the resource is found in another node, the delivery phase includes its transmission to the edge server.

The *transcoding* phase is specific to the problem here considered. In some algorithm, it includes the execution of some transcoding operations carried out in a cooperative way. We assume that any server of the considered cooperative system is equipped with software that can perform the transcoding operations required by any type of client device.

During the *discovery* phase, the servers may cooperate to search for the version of the Web object requested by the client. Since multiple versions of the same object typically exist in the caches, in this phase it is necessary to carry out a multiversion lookup process that may require cooperation among the servers. The discovery phase includes a local lookup and may include an external lookup. Once the edge server has determined the client capabilities, it looks for a copy of the requested resource in its cache. The local lookup may generate one of the following three events.

(1) *Local exact hit*: the cache contains the exact version of the requested object, that can be immediately delivered to the client. (2) *Local useful hit*: the edge server cache contains a more detailed and transcodable version of the requested object that can be transformed to obtain a less detailed version that meets the client request. Depending on the transcoding cooperation scheme, the edge server can decide either to perform the transcoding task locally or to activate an external lookup, which is carried out through some cooperative discovery protocol. (3) *Local miss*: the edge server cache does not contain any valid copy of the requested object. This means that no version of the object is found or a less detailed or untranscodable version is found, but it does not satisfy the RCI associated with the client request. The edge server must activate an external lookup to fulfill the request.

When exact and useful hits are both found in the local cache, the former is preferred because it does not require any adaptation task, and no external lookup is necessary. In the case of local miss and sometimes of useful hit, the edge server may activate some cooperative discovery mechanism to locate an acceptable version on other remote servers. The external lookup may provide one of the following results. (1) *Remote exact hit*: a remote server holds the exact version of the requested object, which is transferred to the requesting server during the successive delivery phase. (2) *Remote useful hit*: a remote server cache contains a more detailed and transcodable version of the requested object that can be transformed to meet the client request. Depending on the transcoding cooperation scheme, the cooperating server can decide either to perform the transcoding task locally or to provide the useful version to the requiring server, which will execute the transcoding process. (3) *Remote miss*: no remote server contains any valid copy of the object, that is, a *global cache miss* occurs. The client request needs to be forwarded to the content server.

It is worth to observe that we consider a generic infrastructure that does not involve the content provider in the transcoding process. Hence, we assume that the Web server returns always the original version of the requested resource. We recognize that our architectures opens many novel possibilities for push caching and object replacement, that we do not address in this paper more focused on the cooperative discovery and transcoding phases. For example, we consider that any server that transcodes a resource, stores in its cache both the retrieved and the transcoded versions of the object, and uses LRU as the cache replacement algorithm. More sophisticated policies have been discussed in [6, 17]. Moreover, we do not address issues related to end-to-end content semantics, such as those considered by server-directed transcoding [11]. Cooperative architectures for transcoding and caching can be integrated with content server decisions or not, without altering our performance considerations. Our main conclusions are, then, not affected by the fact that our servers operate autonomously, without being guided by the content server.

3 Cooperation in hierarchical topologies

In a *hierarchical caching* architecture [15, 20] the servers are organized in a hierarchy, where only the bottom level nodes (called *leaf nodes*) are *edge* servers and hence serve client requests directly. Hierarchical architectures follow the idea of hierarchical Internet organization, with local, regional, and international network providers. This is one of the reasons to have typically three levels in the server tree, from leaf nodes to the root node. In this paper, we consider a pure hierarchical architecture where sibling servers do not cooperate.

Some approaches for distributing the transcoding load among the servers in a hierarchy have been described in [4]. In this paper we consider two cooperation schemes, called *Hierarchical root* and *Hierarchical leaf*. In root-transcoding, each node is both a *transcoder* and a *cache* server. In the case of local miss, the request is forwarded by the local edge server up the hierarchy, until it is satisfied with either an exact or useful hit. In the case of global miss (that is, no type of hit occurs at any level), the root node retrieves the original resource from the content server, adapts it if needed, and sends the exact version of the object to the lower-level server. Each node experiencing a local exact hit responds by sending the resource to the requesting entity, which can be a client or a lower-level server. In the case of useful hit, the contacted server performs locally the content adaptation before sending the exact version of the resource downwards the hierarchy. A copy of the object is stored in the caches of all the nodes along the request path.

The hierarchical cooperation architecture can be easily implemented once transcoding functionalities are added to the server nodes. For the root transcoding scheme, in the case of local miss, the traditional hierarchical lookup is activated that is, the request is forwarded to the parent node. The root server has no parents. In the case of miss on this node, the requested resource is fetched from the content server and then adapted by the root server itself. As the root node must perform the transcoding service for every global miss and content adaptation may involve computationally expensive operations, there is a great risk of overloading this server. Indeed, different studies have shown that pure hierarchical architectures, even when applied to traditional cooperative caching, may suffer from scalability and coverage problems, especially when the number of nodes is large (e.g., [7, 19]). This situation can dramatically worsen in the case of cooperative transcoding. For this reason, we propose the *leaf-transcoding* scheme, that forces the transcoding task to be performed only by the leaf nodes. In this scheme, the behavior of the leaf nodes is different with respect to the upper level nodes. A leaf node handles exact and useful hits as in the root-transcoding scheme. However, in the case of local miss, the upper level receives a request for the original version of the requested resource, and the content adaptation is then performed locally by the leaf node. Upper levels of the hierarchy act as pure cache servers.

4 Cooperative discovery protocols for flat topologies

An alternative to hierarchical topology is to organize the servers in a flat topology where all nodes are *peers*. Unlike hierarchical schemes, each node of this flat architecture has all three functionalities of being a *transcoder*, a *cache*, and an *edge* server. This flat organization allows us to explore various algorithms for cooperative discovery, which are the topic of this section, and for cooperative transcoding, which are discussed in Section 5.

The discovery phase in a flat distributed system can be based on different protocols. We find it convenient to limit the research space of alternatives to the most interesting and widely used systems. It is important to remark that the cooperation protocols for object discovery and delivery considered in this section differ from traditional policies because multiple versions of the same object may be present in the caches of the cooperative edge servers. Moreover, there are three possible results of the external lookup process: miss, exact hit, and useful hit.

Cooperative lookup among distributed servers requires a protocol to exchange local state information. For cooperative discovery, this information basically refers to the cache content, although when we consider a CPU-bound task such as transcoding, other data can be useful (e.g., server load conditions). Cooperative resource discovery has been studied for a while and many mechanisms have been proposed to address the related issues [15]. Most of those mechanisms can be adapted to the lookup of multiple versions. The two main and opposite approaches for disseminating state information are well defined in the literature on distributed systems: *query-based protocols* in which exchanges of state information occur only in response to an explicit request by a peer, and *directory-based protocols* in which state information is exchanged among the peers in a periodic way or at the occurrence of a significant event, with many possible variants in between. In the following, we consider a query-based protocol and a summary-based protocol (a simplified version of the directory-based protocols).

4.1 Query-based protocols

Query-based protocols are conceptually simple. When an edge server experiences a local miss or even a useful hit (depending on the cooperative transcoding algorithm), it sends a query message to all the peers in order to discover whether one of them caches a copy of the requested resource. In the positive case, the recipient edge server replies with an exact hit message or with a useful hit response, otherwise it may reply with a miss message or not reply at all. In the case of a useful hit, the response message should provide some information about the available version of the resource, to allow its retrieval.

The most important query-based protocol is ICP, used in NetCache and Squid [18]. For this reason, we used ICP as the protocol for our query-based cooperation.

In our prototype we added the support for multi-version lookup into the Squid version of ICP by including the version identifier to the URL contained into the messages. Furthermore, it has been also necessary to introduce a new response code to indicate a useful hit instead of an exact hit.

4.2 Summary-based protocols

Directory-based protocols are conceptually more complex than query-based schemes, especially because they include a large class of alternatives. The two most important ones are the presence of one centralized directory vs. multiple directories disseminated over the peers, and the frequency for communicating a local change to the directory/ies. It is impossible to discuss here all the alternatives that have been the topics of many studies. We consider distributed directory-based schemes because it is a common view that in a geographically distributed system any centralized solution does not scale, the central directory server may represent a bottleneck and a single point of failure, and it does not avoid the query delays during the lookup process.

In a distributed directory-based scheme, each edge server keeps a directory of the resources that are cached in every other peer, and uses the directory as a filter to reduce the number of queries. Distributing the directory among all the cooperating peers avoids the polling of multiple edge servers during the discovery phase, and, in the ideal case, makes object lookup extremely efficient. However, the ideal case is affected by large traffic overheads to keep the directories up-to-date. Hence, real implementations use multiple relaxations, such as compressed directories (namely, *summary*) and less frequent information exchanges for saving memory space and network bandwidth, respectively. Examples of compression used to reduce the dimension of the transmitted messages are the Bloom filters, used by Summary Cache [7] and Cache Digests [16], that offer a form of lossy compression of the cache indexes in which a certain amount of false hits is allowed.

For our experiments, we choose Cache Digests as a representative of the summary-based architectures, because of its popularity and its implementation in the Squid software. Support for caching and discovery of multiple versions has been added to our prototype into the summary-based lookup process through URL-encoding the resource version identifier. Therefore, the basic mechanism of Cache Digests cooperation is preserved. However, the lookup process becomes more expensive because it has to carry out a search for every possible useful version.

5 Cooperative transcoding algorithms for flat topologies

Cooperative transcoding is necessary only when a local or a remote useful hit occurs. Misses and exact hits are handled as described in Section 2, and they are unrelated to the cooperative transcoding algorithms. We can identify two alternatives in the case of local and remote useful hits. For a local useful hit, the edge server can either transcode locally the resource as soon as it retrieves the hit from its cache or it can activate an external lookup process, with the possibility of retrieving a remote exact hit from some peer cache. The latter choice aims to reduce the CPU load on the local edge server (by avoiding some transcoding operation), at the expenses of external lookup operations, which can increase the overall response time, if the remote exact hit rate is low. In the case of remote useful hit, the alternative regards the selection of the server node which performs the adaptation of the transcodable version, being the choice between the edge server which received the client request and the peer that found the useful hit in its cache.

Since transcoding a useful hit may be computationally expensive, several load-balancing algorithms can be used. In particular, we distinguish between **load-blind** algorithms that do not take into account any load state information and **local load-aware** algorithms, that use load information about the local server itself for deciding about the node that must perform the transcoding task.

We propose two load-blind algorithms and a local load-aware algorithm and we evaluate their performance with special attention to the response time.

5.1 Load-blind algorithms

We propose two load-blind algorithms, called *blind-lazy* and *blind-active*. The **blind-lazy** algorithm, whose flow diagram is shown in Figure 1(a), tends to limit the computational costs of transcoding by taking most advantage of the cooperative peers. In the case of a local useful hit, the edge server continues the discovery phase by activating an external lookup process to look for an exact version of the requested object in some peer proxy. In the case of a remote useful hit, the edge server always delegates the transcoding task to the peer server that reported the useful hit. The rational behind this approach is to exploit as much as possible the remote exact hits and to distribute in a nearly random way the transcoding process. The price of the external lookup process is worth when the remote exact hit is found and the network links are not saturated; otherwise, a (guaranteed) local useful hit may be preferable to a (possible) remote exact hit.

The **blind-active** algorithm, shown in Figure 1(b), follows an approach opposite to its blind-lazy counterpart. Whenever possible, it saves network usage for the external lookup at the price of local computation. In the case of a local useful hit, the edge server transcodes the useful version found in its cache without continuing the discovery phase. In the case of a remote useful hit, the resource is retrieved from the peer and transcoded locally.

5.2 Load-aware algorithm

The **load-aware** algorithm we propose in this paper is based on local load information, thus not requiring any load information exchange among the peers. When a local or a remote useful hit occurs, the edge server decides whether to perform locally the transcoding operation or to continue the discovery phase on the basis of its current load. Although this algorithm seems somehow naive, we found that it can achieve a performance increase when the distribution of the client requests among the edge servers is unbalanced.

The basic idea of the load-aware algorithm is to follow for each local useful hit one load-blind scheme or the other depending on the CPU load. When the CPU utilization of the edge server surpasses a certain threshold, it behaves in a *lazy* mode, as the lazy approach tends to save local CPU resources. Otherwise, the edge server adopts the *active* approach, because there is enough spare CPU power to perform transcoding. The threshold value can range from 0.0 to 1.0. It is worth to note that the extreme values correspond to the load-blind algorithms. If the load threshold is set to 1.0, the edge server applies the blind-active algorithm; when the threshold is set to 0.0, the server applies the blind-lazy algorithm.

6 Architecture of the ColTrES prototype Testbed

We have implemented a prototype called ColTrES (Collaborative Transcoder Edge Services) that can support all the cooperative transcoding architectures considered in this paper. In this section we describe the features of the prototype that transforms each server from a caching intermediary into an active node that is able to adapt the Web content, to carry out a local lookup of multiple versions of the objects, and to activate an external multi-version lookup. The peculiarities of the cooperation mechanisms for each specific architecture have been described in Sections 3 and 4 for the hierarchical and flat schemes, respectively.

The basic software platform for the prototype is the Squid Web proxy cache, version 2.4 [18]. We chose Squid as a platform not only because of its popularity, robustness and open source characteristics, but also because it supports different cooperation mechanisms. Hence, we could implement the proposed multi-version lookup and cooperative transcoding by following the basic Squid concepts and modifying the existing code.

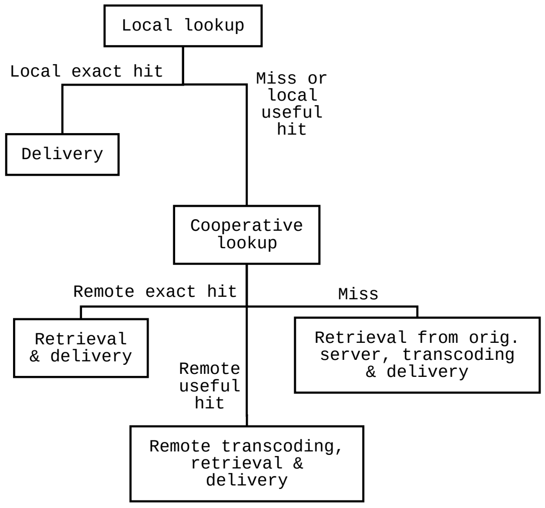
Important modifications to the original Squid software have been introduced to support the management of multiple variants of the same object. Specifically, we alter the URL in some data structures to insert a version identifier into the URL. This modification entails also changes in the Squid lookup algorithm, that has been modified to consider the possibilities of both *exact* and *useful* hits.

As Squid supports various cooperation mechanisms, we have introduced the multi-version lookup in cooperative lookup by extensively modifying the modules responsible for this process. Most of those modifications are related to the use of modified URLs that contain also a version identifier of the resource.

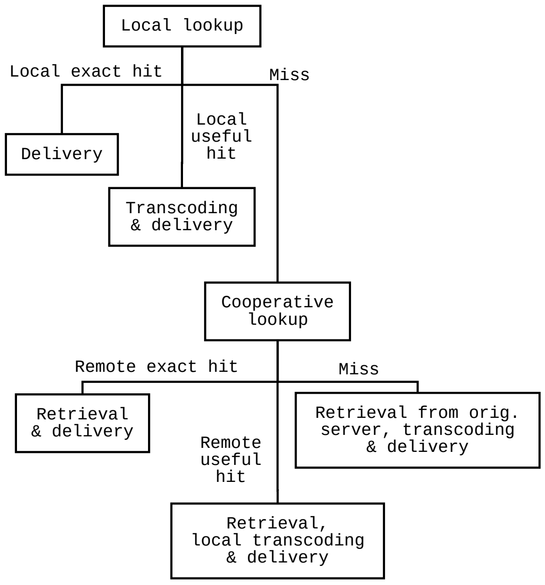
To add the transcoding functionalities, we have implemented a new module that handles the transcoding operations. The transcoding task is performed by an external process, called *transcoder*, and uses the freely available ImageMagick library to adapt the resources to the client specifications. We use an external process to allow multiple transcoding operations to be performed in parallel. Moreover, the use of multiple instances of the transcoder can increase the performance in SMP systems (in our experiments, the speedup on dual processor systems was up to 110% on 90-percentile of response time with respect to an identical node with only one CPU). For a more detailed description of the modification to the source code of Squid, the reader can refer to [2].

7 Workload model

In this section we describe the client and workload models used to test the performance of the cooperation mechanisms and algorithms. We consider a classification of the client devices on the basis of their capabilities of displaying different objects and connecting to the assigned edge server [4, 6]. The classes of devices range from high-end workstations/PCs which can consume every object in its original form, to cellular phones with very limited bandwidth and display capabilities. We introduced six classes of clients each with its own



(a) Blind-lazy algorithm.



(b) Blind-active algorithm.

Figure 1: Load-blind algorithms.

characteristics. The full description of the devices capabilities and the values of their popularity that has been used in our experiments can be found in [2]. In this paper, we consider that most transcoding operations are applied to image objects (GIF, JPEG, and BMP formats), as more than 70% of the files requested in the Web still belong to this class [5]. However, in our experiments we also consider scenarios where the transcoding operations can have higher costs, such as the ones that can be found in a near future of the Web characterized by larger percentage of multimedia resources.

Our experiments aims at comparing different cooperative architectures and algorithms, and in each set of experiments we used different workloads.

The first workload, namely **light trans-load**, aims at capturing a realistic Web scenario with a reduced transcoding load. The set of resources used in this workload are based on proxy traces belonging to the nodes of the IRCache infrastructure. We downloaded the resources from their content servers and placed them on Web servers. We performed some characterization on the images in the light workload, such as file size, JPEG quality factor, and number of colors of GIF images, and found that they are very close to the results reported in [5]. The measured costs of transcoding operations required by this set of resources on the machines used for our experiments gave the following results: 0.04 and 0.22 seconds for the median and the 90-percentile service time, respectively (the cumulative distribution is shown in Figure 2).

We also consider a second workload model (called **heavy trans-load**) that aims at denoting a scenario where the transcoding process has a major cost. As the trend of the Web is towards a growing demand for multimedia resources, this workload can represent a situation with a large amount of multimedia objects, such as video and audio. In this scenario, the costs for transcoding operations are 0.27 and 1.72 seconds for the median and the 90-percentile service time, respectively (as for the previous working set, the cumulative distribution of the transcoding time in shown in Figure 2).

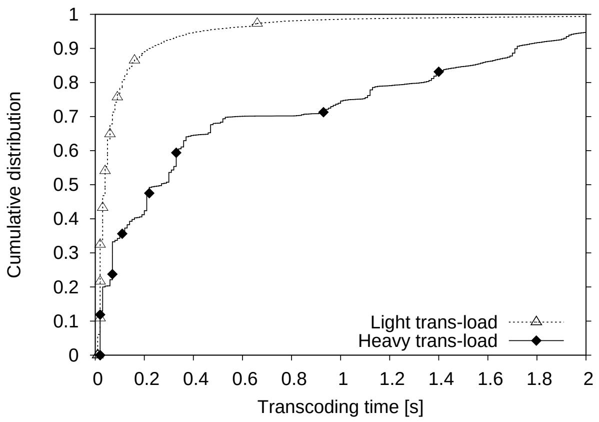


Figure 2: Cumulative distributions of transcoding time for the light and heavy trans-load scenarios.

The size of the working set (including only the original resources) for the light and heavy trans-load workloads is similar and corresponds to 10% more than the sum of the cache size used in our experiments. On the other hand, the mean file size differs considerably, depending on the used set of resources. Hence, the light workload determines higher cache hit rates than the heavy one. We have also introduced a popularity resource distribution by defining a set of hot resources (corresponding to 1% of the working set): 10% of the total number of requests refers to this hot set.

From the file list of each workload model, we obtained 80 different traces that were used in parallel during the experiments. Each trace consists of 1000 requests with a random delay that elapses between two consecutive requests. For both workloads, the number of traces assigned to the edge servers is the same, so that each node receives the same number of requests. However, the types of requests in each trace can differ substantially, because the file size follows a heavy-tailed distribution, especially for the light trans-load working set.

To study the performance of the transcoding algorithms, it has been necessary to consider a scenario with systems under heavy stress. To this purpose, we created two workload models (called **uniform** and **bimodal**) which are based on the previously described heavy trans-load workload, but are characterized by different client request distributions. In the uniform scenario the request load distribution is even, with each node receiving the same number of client requests. In the bimodal scenario, instead, the distribution of client requests among the edge servers is unbalanced, with 50% of the servers receiving 90% of the requests and the remaining half of the nodes handling only 10% of the traffic.

8 Experimental results

We use three main metrics to evaluate the performance of the proposed schemes for cooperative discovery and transcoding: the *CPU utilization* of the servers, the *cache hit rates* (local, global, exact, useful), and the *system response time* that corresponds to the interval between the instant in which the client sends a request to the edge server and the instant in which the client receives all the response.

As our main target is to enable heterogeneous devices to access Web content, the servers transcode the object to best fit the client capabilities, while we do not explore object compression to reduce transmission time as done in [9]. We also consider only complete transcoding relation graphs, where each version can be created starting from any higher quality version [4]. In our experiments we set up a system of 16 servers. The servers are equipped with ColTrES and configured to cooperate through different architectures and discovery protocols.

8.1 Comparison of the architectures

In this section we compare the performance of the hierarchical and flat architectures of servers that collaborate in discovering, transcoding, and delivering Web objects. We set up a scenario where all servers are well connected among them and with the clients. The content servers are placed in a remote location, connected through a geographic link with 14 hops in between, a mean round-trip time of 60 ms, and a maximum bandwidth of 2Mb/sec. We verified that in this scenario the network path to the content servers (reached in case of global miss) was one of the possible system bottleneck. Hence, the global cache hit rate may impact on the response time.

We consider the **Hierarchical leaf** and **Hierarchical root** schemes for the hierarchical architecture, and the query-based protocol (**ICP**), and the summary-based protocol (**Cache Digests**) schemes for the flat architecture. For a fair comparison, in this set of experiments flat schemes use the blind-active algorithm.

Both hierarchical schemes are configured on the basis of a three-level hierarchy with 12 leaves, 3 intermediate servers (with a nodal out-degree of 4), and one root node. The client are redistributed to let only the leaves receive their requests. The workload size and the number of requests are not changed. The configuration for ICP and Cache Digests is based on a flat cooperation scheme, where all edge servers have sibling relationships among them.

In these experiments we use both light trans-load and heavy trans-load workloads. First, we evaluate the cache hit rates of the various cooperation schemes. We then focus on the response time, which is the crucial performance metric to the end users.

Tables 1 and 2 show the cache hit rates for light trans-load and heavy trans-load workloads, respectively. For each cooperation scheme, we report the local exact and useful hit rates (columns 2 and 3, respectively) as well as the remote hit rates (columns 4 and 5). The last column shows the global hit rate, which is the sum of the various hit rates. For the hierarchical leaf scheme, we do not report the remote useful hits, because the requests to the parent nodes refer only to the original version of the resources.

We will start describing Table 1, using data of Table 2 as a comparison to confirm our observations or to identify differences. From the last column of Tables 1 we can observe that there are some significant differences in the global hit rates, depending on the used cooperation mechanism. In particular, ICP provides the best results, while Cache Digests turns out to be less effective in finding hits. Cache Digests becomes imprecise (i.e., the accuracy of the exchanged cache digests decreases) and its remote hit rates diminish. This is particularly evident for the heavy trans-load workload (but it can be also observed for the light one): the presence of larger objects causes faster changes in the cache contents, having as a con-

Table 1: Cache hit rates (*light trans-load*).

	Local exact HR	**Local useful HR**	**Remote exact HR**	**Remote useful HR**	**Global HR**
ICP	19.4%	16.9%	13.8%	19.3%	69.5%
Cache Digests	21.2%	11.9%	11.5%	11.5%	56.4%
Hierarchical root	17.9%	6.8%	7.1%	7.7%	39.7%
Hierarchical leaf	10.2%	8.2%	19.6%	n/a	38.2%

Table 2: Cache hit rates (*heavy trans-load*).

	Local exact HR	**Local useful HR**	**Remote exact HR**	**Remote useful HR**	**Global HR**
ICP	5.1%	4.7%	20.3%	22.1%	52.4%
Cache Digests	5.3%	4.6%	10.3%	8.9%	29.2%
Hierarchical root	6.3%	4.7%	5.2 %	4.4 %	20.7%
Hierarchical leaf	6.1%	4.3%	11.6%	n/a	22.1%

sequence a reduction of the accuracy of exchanged digests. Columns 4 and 5 in Table 1 show that the reduction in the global hit rate is caused by a reduction of the remote hit rate.

The two hierarchical schemes achieve similar global hit rates (last column of Tables 1). However, their global hit rates are lower than those of flat architectures. The most evident and expected result observed from comparing Tables 1 and 2 is the higher hit rates obtained under the light trans-load workload, because the object sizes in the heavy trans-load workload are larger than those in the light one. The lower hit rate of the heavy trans-load workload increases the replacement activity, thus reducing the hit rate of Cache Digests. For this reason, the reduction in remote hit rate of this protocol observed for the light trans-load workload is even more evident from columns 4 and 5 of Table 2.

We now pass to consider the response time. Figures 3 and 4 show the cumulative distribution of system response time for the considered schemes under the light trans-load and heavy trans-load workloads, respectively. Most of the curves show several steps or jumps as a consequence of the different kinds of cache hit (i.e., remote vs local, and useful vs exact) at the discovery phase. All the cooperation schemes present a first step (located on the left side of each graph) due to local exact hits that are nearly instantaneous with respect to other hits and misses. Useful local hits have longer response times, which are typically comparable to the ones of remote exact hits. Remote useful hits are even longer response times, but, due to the variance in response time, do not generate apparent steps on the response time curve. Misses generate the highest response times, hence are typically located in the right side of each curve.

The two figures also show that the hierarchical leaf scheme clearly outperforms the hierarchical root one. However, none of the hierarchical schemes can compete with flat architectures (ICP and Cache Digests). The expected bad performance of the hierarchical root scheme is due to its poor load balance. We observed that the higher levels of the hierarchy are often overloaded because they have to handle all misses from the lower levels. Measurements on the CPU load show that the mean load of the root node is nearly 0.90 and 0.99 for light trans-load and heavy trans-load workloads, respectively, as this node has to process every miss occurred in the lower levels. On the other hand, leaf edge servers are often idle (the corresponding mean CPU load is less than 0.02 for both workloads), waiting for upper level nodes to process their requests.

The hierarchical leaf scheme achieves better performance: the response times in Figures 3 and 4 are much lower than those obtained by the hierarchical root. However, even the hierarchical leaf scheme is penalized with respect to the flat schemes. There are two reasons for this result. In hierarchical leaf scheme, the upper hierarchy levels can only act as pure cache servers (in our testbed prototypes, 4 nodes over 16 do not contribute in transcoding operations). Moreover, as shown in the Tables 1 and 2, the flat cooperation schemes achieve the highest cache hit rates.

Flat architectures offer the best results. A preliminary performance comparison between ICP and Cache Digests is reported in [3]. With the experiments carried out in this paper we confirm the previous observations: the higher global hit rates of ICP tend to reduce the response time of the found resources. On the other hand, due to the faster lookup mechanism of Cache Digests, remote hits are typically serviced faster than those of ICP. For this reason, it seems interesting to analyze the cumulative distribution of the response time. Table 3 provides a summary of data in Figures 3 and 4. It shows the median (50-percentile) and 90-percentile of the response time for each cooperation scheme and both workload models.

Figure 3 shows that the difference between the two curves of Cache Digests and ICP is slight, with ICP only slightly superior to Cache Digests on the right side of the graph, even

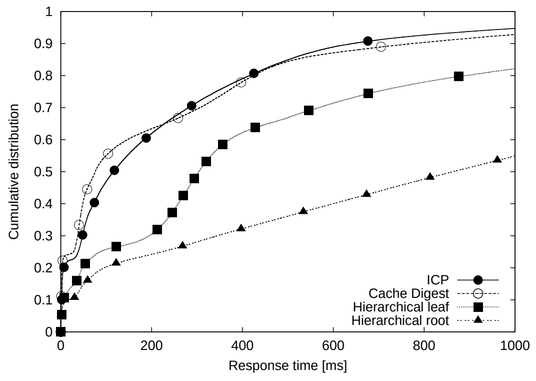


Figure 3: Cumulative distributions of system response times (*light trans-load*).

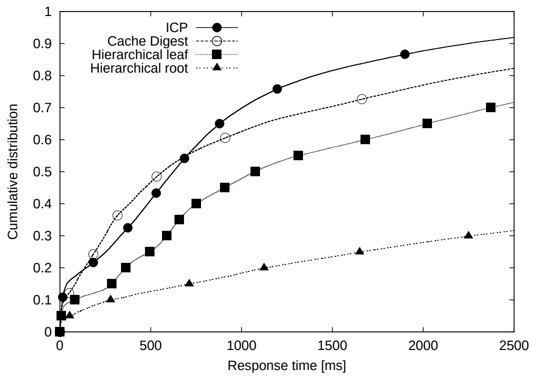


Figure 4: Cumulative distributions of system response times (*heavy trans-load*).

Table 3: Median and 90-percentile of system response times.

Cooperation scheme	**Light trans-load**		**Heavy trans-load**	
	median [sec]	**90-percentile [sec]**	**median [sec]**	**90-percentile [sec]**
ICP	0.11	0.64	0.62	2.24
Cache Digests	0.07	0.78	0.56	3.76
Hierarchical root	0.86	2.82	5.52	14.57
Hierarchical leaf	0.30	1.74	1.07	5.11

if the global hit rate differs significantly (69.5% vs. 56.4%). Moreover, if we analyze the median response time, we can see that Cache Digests is faster than ICP (also shown in column 2 of Table 3). This can be explained by the high lookup time required by ICP. On the other hand, under the heavy trans-load workload (Figure 4) the curves of response times are more differentiated with ICP outperforming Cache Digests, due to the higher difference in cache hit rate (52.4% vs. 29.2%) that cannot be compensated by the faster lookup. Even in this case, however, the median response time is lower for Cache Digests.

Table 3 summarizes the results that can be get from the figures: the hierarchical root scheme is the slowest; flat architectures outperform hierarchical schemes; ICP is the fastest scheme to serve the large majority of the requests, even if Cache Digests can be faster than ICP to serve half of the requests for both workloads.

8.2 Cooperative transcoding algorithms

In this section we compare the performance of the cooperative transcoding algorithms for flat architectures described in Section 5. For this set of experiments we choose the ICP cooperation scheme, because it typically offers the highest cache hit rates and lowest response times.

ICP performs well with a wide range of workloads, at least until the system is under heavy stress, as noted in [3]. Under low and medium load, the difference between the various transcoding algorithms is very small. Therefore, it is more interesting to explore the performance gain achievable with the proposed cooperative transcoding algorithms when server CPUs are nearly always busy due to transcoding operations. To this purpose, we used the bimodal and uniform workloads described in Section 7.

Figure 5 shows the cumulative distribution of response time for the load-blind and load-aware algorithms with the bimodal workload, while Figure 6 refers to the uniform workload. It is worth to note that the load-aware algorithm is a typical threshold-based policy that uses the CPU utilization as activation parameter. We performed experiments with different thresholds ranging from 0.1 to 0.9 and found that for bimodal workload the typical common-sense value of 0.66 for the threshold offers the most stable performance in terms of 90-percentile of response time. Therefore, Figure 5 shows only the curve related to this threshold value. On the other hand, for the uniform workload we found that no "best" threshold value exists: the 90-percentile of the response time grows monotonically as the threshold value decreases from 0.9 to 0.1. In Figure 6 the curve of the load-aware algorithm corresponds to the same load threshold value (0.66) used in Figure 5. For the best threshold value (0.9) for this workload, the curve is in between the one of the blind active algorithm and the curve for threshold equal to 0.66.

Response time curves achieved by blind-active and load-aware algorithms are similar, with the load-aware algorithm providing better response times in the case of bimodal workload and the blind-active algorithm being faster in the case of

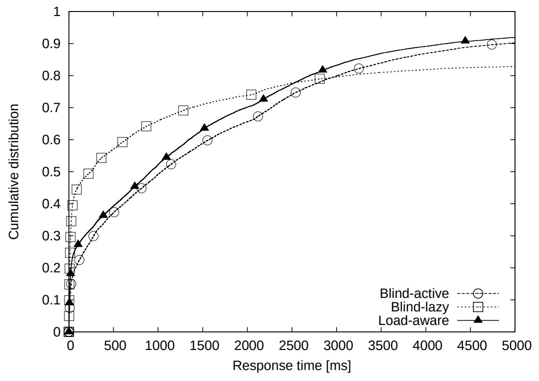


Figure 5: Cumulative distribution of system response times (*bimodal workload*).

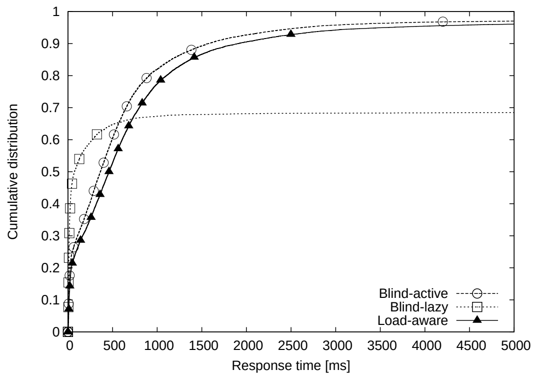


Figure 6: Cumulative distribution of system response times (*uniform workload*).

Table 4: Response times for load-blind and load-aware algorithms.

	Bimodal workload		**Uniform workload**	
Algorithm	**median [sec]**	**90-percentile [sec]**	**median [sec]**	**90-percentile [sec]**
Blind-active	1.03	4.88	0.36	1.56
Blind-lazy	0.25	121.12	0.07	239.79
Load-aware	0.89	3.98	0.46	1.90

uniform workload. On the other hand, the blind-lazy algorithm shows a different behavior. It is able to reduce the response time for most requests, but it becomes unacceptably slow for up 30% of the requests, depending on the workload.

To better present the performance differences, in Table 4 we report the median and the 90-percentile of the response time.

The blind-lazy algorithm bets on finding either a useful hit or an exact remote hit on a less loaded peer. If it succeeds, it can reduce the response time. However, if no remote hit is found or, even worse, if the peer having a remote useful hit is overloaded, the response time can increase significantly. This explain why the blind-lazy algorithm can successfully reduce the median response time (as shown in columns 2 and 4 of Table 4), but it tends to have poor performance when considering 90-percentile due to the high number of pathological cases (columns 3 and 5 of Table 4). The problem is more evident when the load is evenly distributed (i.e., uniform workload, column 5 of Table 4), because in this case there is a higher probability of finding a peer with a heavier load.

On the other hand, the blind-active algorithm seems to offer better performance because the transcoding load is only related to the client request being served and not to the request directed to other peers. The response time has a more definite upper bound, thus reducing the 90-percentile of the response time. On the other hand, the median response is higher than that of the blind-lazy algorithm.

The load-aware algorithm offers some performance gain when the load is unevenly distributed (bimodal workload), because it can act smarter than the load-blind algorithms. In particular, it reduces of about 22% the 90-percentile (as shown in column 3 of Table 4) and about 14% the median response time (column 2 of Table 4) with respect to the blind-active algorithm. On the other hand, in the case of uniform workload, the load aware algorithm is ineffective in reducing the response time, and there is a performance loss on both 90-percentile and median response time. Indeed, when the skewness of the workload is low, we need a more sophisticate algorithm, possibly based on information on the load of a large (maybe whole) set of edge servers.

9 Conclusions

In this paper, we have investigated practical schemes for cooperative caching and transcoding that can be implemented in the existing Web infrastructure and have compared their performance through ColTrES, a real prototype testbed based on Squid.

We evaluated different cooperation schemes that use both hierarchical and flat architecture and found that hierarchical schemes are always surclassed by flat topologies, due to the risk of bottlenecks in the higher levels of the hierarchy as well as reduced hit rates. Among the flat cooperation schemes, we evaluated multi-version lookup extensions of Cache Digests and ICP and found that, due to the lower hit rate of Cache Digests, ICP tends to have better performances.

As a further experimental contribution of this paper, we verified that the proposed load-aware algorithm can achieve a performance gain in case of unevenly distributed request load. On the other hand, in case of uniform load distribution, the load-aware algorithm fails to reduce response time.

A plethora of research issues can be studied for the topic of active edge server systems that cooperate in multi-version content caching, discovery, and transcoding. A limited number of issues have been investigated in this work, that to the best of our knowledge represents the first implementation of cooperative transcoding and caching systems for both hierarchical and flat topologies. There are many research issues that this paper opens up, such as cooperative cache replacement policies that take into account multi-version content, transcoding policies based on global load information and network available bandwidths, and the integration with server-direct transcoding to preserve the end-to-end content semantics.

References

[1] M. Butler, F. Giannetti, R. Gimson, and T. Wiley. Device independence and the Web. *IEEE Internet Computing*, 6(5):81–86, Sept./Oct. 2002.

[2] C. Canali, V. Cardellini, and R. Lancellotti. Squid-based proxy server for content adaptation. Technical Report TR-2003-03, Dept. of Computer Engineering, Univ. of Roma “Tor Vergata”, Jan. 2003. `http://weblab.ing.unimo.it/research/trans_caching.shtml`.

[3] V. Cardellini, M. Colajanni, R. Lancellotti, and P. S. Yu. A distributed architecture of edge proxy servers for cooperative transcoding. In *Proc. of 3rd IEEE Workshop on Internet Applications*, June 2003.

[4] V. Cardellini, P. S. Yu, and Y. W. Huang. Collaborative proxy system for distributed Web content transcoding. In *Proc. of 9th ACM Int’l Conf. on Information and Knowledge Management*, pages 520–527, Nov. 2000.

[5] S. Chandra, A. Gehani, C. S. Ellis, and A. Vahdat. Transcoding characteristics of Web images. In *Proc. of Multimedia Computing and Networking Conf.*, Jan. 2001.

[6] C.-Y. Chang and M.-S. Chen. Exploring aggregate effect with weigthed transcoding graphs for effi cient cache replacement in transcoding proxies. In *Proc. of IEEE 18th Int’l Conf. on Data Engineering*, Feb. 2002.

[7] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. *IEEE/ACM Trans. on Networking*, 8(3):281–293, June 2000.

[8] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Proc. of 16th ACM Symp. on Operating Systems Principles*, pages 78–91, Oct. 1997.

[9] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas. Dynamic adaptation in an image transcoding proxy for mobile Web browsing. *IEEE Personal Communications*, 5(6):8–17, Dec. 1998.

[10] A. Joshi. On proxy agents, mobility, and Web access. *Mobile Networks and Applications*, 5(4):233–241, 2000.

[11] B. Knutsson, H. Lu, and J. Mogul. Architectures and pragmatics of server-directed transcoding. In *Proc. of 7th Int’l Workshop on Web Content Caching and Distribution*, Aug. 2002.

[12] W. Y. Lum and F. C. M. Lau. On balancing between transcoding overhead and spatial consumption in content adaptation. In *Proc. of ACM Mobicom 2002*, pages 239–250, Sept. 2002.

[13] A. Maheshwari, A. Sharma, K. Ramamritham, and P. Shenoy. TransSquid: Transcoding and caching proxy for heterogeneous e-commerce environments. In *Proc. of 12th IEEE Int’l Workshop on Research Issues in Data Engineering*, pages 50–59, Feb. 2002.

[14] R. Mohan, J. R. Smith, and C.-S. Li. Adapting multimedia Internet content for universal access. *IEEE Trans. on Multimedia*, 1(1):104–114, Mar. 1999.

[15] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison Wesley, 2002.

[16] A. Rousskov and D. Wessels. Cache Digests. *Computer Networks*, 30(22-23):2155–2168, 1998.

[17] A. Singh, A. Trivedi, K. Ramamritham, and P. Shenoy. PTC: Proxies that transcode and cache in heterogeneous Web client environments. In *Proc. of 3rd Int’l Conf. on Web Information Systems Engineering*, Dec. 2002.

[18] Squid Internet Object Cache. `http://www.squid-cache.org`.

[19] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative Web proxy caching. In *Proc. of 17th ACM Symp. On Operating Systems Principles*, Dec. 1999.

[20] P. Yu and E. A. MacNair. Performance study of a collaborative method for hierarchical caching in proxy servers. *Computer Networks*, 30(1-7):215–224, 1998.