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
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 processing powers, display capabilities, and 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 provider or by some proxy server at the edge. In this paper, we propose an alternative solution consisting of an intermediate infrastructure of distributed servers which collaborate in discovering, transcoding, and delivering multiple versions of Web resources to the clients. We investigate different algorithms for cooperative discovery and transcoding in the context of this intermediate infrastructure where the servers are organized in hierarchical and flat peer-to-peer topologies. We compare the performance of the proposed schemes through a flexible prototype that implements all proposed mechanisms.

1 Introduction

The Web is rapidly evolving towards a highly heterogeneous accessed environment, due to the variety of consumer devices that are increasingly gaining access to the Internet. The emerging Web-connected devices, such as handheld computers, PDAs, mobile phones, 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 these diverse destination devices.

The content adaptation mechanism, called *transcoding*, can be applied to transformations within media types (e.g., reducing the color depth of an image), across media types (e.g., video clip to image set) or to both of them. The existing approaches to deploy Web content adaptation fall into three broad categories depending on the entity that performs the adaptation process [1, 12]: *client-based*, *edge-based* (also called *proxy-based*), and *server-based* adaptation. A comprehensive analysis on related work is in Section 2. The edge-based approach uses the proxy server to analyze and adapt the content on-the-fly, before delivering the result to the user. This component is often called *edge server* as in the delivery chain between the client device and the content server it is generally located close to the client. So far, the edge-based approach has been typically carried out by some edge server that directly connects to the clients. In this paper, we explore a different alternative that considers a distributed system of cooperative servers which collaborate in discovering, transcoding, and delivering Web content. Minimizing the user response time and bounding its variability are the main goals of this distributed cooperative architecture. Indeed, the computational 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 task of content adaptation. Although the cooperative transcoding is based on distributed schemes, it is not a simple extension to cooperative caching because two main issues have to be addressed to achieve a suitable solution. First, the presence of diverse consumer devices requires to recognize, discover, and cache the multiple variants of the same resource obtained by transcoding operations. Moreover, the transcoding process is expensive in terms of server 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. We compare their performance through a prototype called ColTrES (Collaborative Transcoder Edge Services), which is a flexible testbed based on Squid [19]. ColTrES implements all considered mechanisms by extending the traditional cooperative caching systems to an environment characterized by heterogeneous client devices. The first extension transforms a traditional cache server into an active intermediary, which not only caches Web objects but also transcodes them, stores the results, and allows multi-version lookup operations [7, 18]. The second novel extension allows the cooperation of the active intermediaries for caching and transcoding. 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. Through our prototypes, we demonstrate that all proposed algorithms and cooperative architecture are immediately applicable to the Web infrastructure. The real testbed allows 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 follows. Section 2 analyzes related work. Section 3 discusses the main features of a distributed system for cooperative transcoding. Sections 4 and 5 explore different topologies and protocols for cooperative discovery, transcoding, and delivery. Section 6 describes the workload model used to exercise the prototype. Section 7 presents the experimental results. Section 8 concludes the paper with some final remarks.

2 Related work

The client-based approach to content adaptation seems not suitable for all the cases in which clients offer limited processing power and connection bandwidth. The server-based approach, that adds content adaptation services to traditional Web server functionalities [14], increases the complexity of the server platform and software, but remains a valid alternative. However, in this paper we will focus on content adaptation carried out by an intermediate architecture of distributed servers. The advantages of proxy-based content adaptation have been explored by many recent studies [4, 3, 5, 7, 9, 10, 13, 18]. This scheme can use one [10, 13, 18] or more [4, 3, 9] servers to analyze and adapt the content on-the-fly, up to fully distributed peer-to-peer networks as in [17] (although the study of Shi *et al.* is more focused on personalized contents). An intermediate adaptation can shift load from content-providing servers and simplify their design. Moreover, this solution is also viable because the large majority of consumer devices requires some proxy to access the Web. A transcoding service located at intermediary points of the network can also tailor resources coming from different content servers. The intermediate server plays also another important role that is, it can cache the results of content adaptation, thus avoiding some round-trips to the content server and costly transcoding operations when resources can be served from the cache [9, 18].

Most research has focused on handling the variations in client bandwidth and display capabilities (e.g., [5, 9, 10]), without focusing on caching aspects. In these proposals, the edge server that directly connects to the clients typically reduces the object size (thus reducing bandwidth consumption), apart from providing a version that fits the client device capabilities. A limited number of recent proposals have also exploited techniques to combine both adaptation and caching to reduce the resource usage at the edge server [7, 13, 18].

The large majority of research efforts have been devoted to the investigation of solutions in which the adaptation and caching functionalities are provided on stand-alone edge servers that do not cooperate among them. The main motivation that leaded us to study distributed architectures for intermediate services of caching and adaptation is the limited scalability of a single proxy-based approach because of significant computational costs of adaptation operations [12]. Fox *et al.* [9] address this scalability issue by proposing a cluster of locally distributed edge servers. This approach may solve the CPU-resource constraint, but it tends to move the system bottleneck from the server CPU to the interconnection of the cluster. On the other hand, the proposes infrastructure is designed to be distributed over a wide area network thus preventing network bottlenecks.

In recent works we have started to examine how to extend traditional caching architectures to the active support of cooperative transcoding. In [4] the authors have obtained some preliminary simulation results that demonstrated the importance of distributing the computational load of transcoding in a cooperative hierarchical scheme. Some preliminary experimental results on flat topologies have been presented in [3], in which the authors demonstrate that a cooperative distributed system consistently outperforms in terms of user response times a system of non-cooperative servers.

3 Main features of the intermediate infrastructure

The servers of a distributed system can be organized and cooperate through a large set of alternatives. Each node of the intermediate infrastructure may have one or more functionalities that is, it can act as a *transcoder*, a *cache* or an *edge* server. *Edge* servers receive requests directly from the clients and deliver the requested resources. *Cache* servers provide caching functionalities for both original and transcoded resources. *Transcoder* servers perform content adaptation. In this paper we consider hierarchical and flat topologies. In flat topologies all nodes are peers and provide all functions, while in hierarchical topologies the nodes may provide different functionalities. In this section we outline the main common operations that characterize an intermediate infrastructure for cooperative transcoding, while the specific features of hierarchical and flat organizations are described in Sections 4 and 5, respectively.

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

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

The *transcoding* phase is specific to the problem here considered. We assume that any server of the cooperative system is equipped with software that can perform the transcoding operations required by any type of client device that contacts an edge server. The features of client devices vary widely in screen size and colors, processing power, storage, user interface, software, and network connections. Recently, the WAP Forum and the W3C have also proposed the standards CC/PP and UAProf for describing the client capabilities [1]. The client may also include the resource type it can consume as a meta-information in the HTTP request header. 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 adapted 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].

It is worth to observe that we consider a generic infrastructure that does not involve the content-providing server in the transcoding process as the proposal of server-directed transcoding [11]. Hence, we assume that the content server always returns the original version of the requested resource. Our cooperative architectures for transcoding and caching can be integrated with content server decisions or not, without altering main performance considerations and conclusions of this paper.

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 multi-version 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 cache contains a more detailed and transcodable version of the object that can be transformed to match 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 cache does not contain any valid copy of the requested object. The edge server must activate an external lookup to fulfill the request.

When both exact and useful hits are found in the local cache, the former is preferred because it does not require any adaptation task, and no external lookup is necessary. We recognize that our architectures opens many novel possibilities for push caching and object replacement [7], that we do not consider in this paper.

In the case of local miss and sometimes of useful hit, the edge server may activate some cooperative discovery mechanism to locate a version on some other server. 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. (2) *Remote useful hit*: a remote server contains a more detailed and transcodable version of the requested object that can be transformed to meet the 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 requesting 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 original version of the requested resource must be fetched from the content server.

4 Hierarchical topologies

In this paper, we consider a pure hierarchical architecture where sibling servers do not cooperate, and only the bottom level nodes (called *leaf nodes*) are *edge* servers that can be contacted by the clients [15]. We use the commonly adopted three-level tree from leaf nodes to the root node, because hierarchical architectures follow the idea of hierarchical Internet organization, with local, regional, and international network providers.

Some schemes for distributing the transcoding load among the servers organized in a hierarchy have been described in [4]. In this paper we consider two approaches, called *Hierarchical root* and *Hierarchical leaf* cooperation schemes. In the Hierarchical root scheme, each node is both a *transcoder* and a *cache* server. In the case of local miss, the request is forwarded by the edge server up to the hierarchy, until it is satisfied with either an exact or useful hit. In the case of global miss (that is, no level holds a valid copy of the requested resource), the root node retrieves the original resource from the content provider, if necessary adapts it, 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.

As the root node must perform the transcoding service for every global miss and content adaptation may involve costly 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., [8, 20]). This situation can dramatically worsen in the case of cooperative transcoding. For this reason, we propose the *Hierarchical leaf* scheme, that differentiates the roles of the nodes in the intermediate infrastructure. The leaf nodes maintain the roles of edge, cache and transcoding servers, while the upper-level nodes provide just cache services for original versions of the resources. When necessary, content adaptation is performed locally by the leaf nodes.

5 Flat topologies

An alternative to hierarchical topology is the flat organization, in which all nodes are *peers* and provide the functionalities of *transcoder*, *cache*, and *edge* server. This flat organization allows us to explore various algorithms for cooperative discovery and cooperative transcoding, which are discussed in the following sections.

5.1 Cooperative discovery

Although the discovery phase in a flat topology can be based on different protocols, we limit the research space of alternatives to the most interesting and widely used systems. It is worth to remark that the cooperation protocols for object discovery and delivery considered in this section differ from the traditional ones 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, which 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). 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).

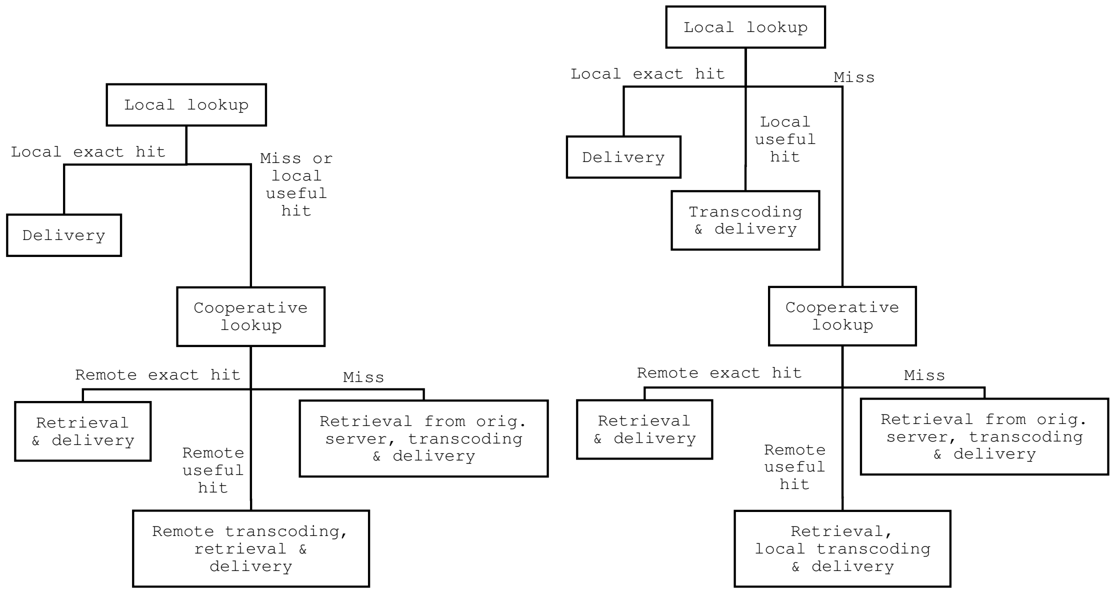
Query-based protocols are conceptually simple. When an edge server experiences a local miss or even a useful hit (depending on the cooperative transcoding algorithm), it sends a query message to all the peers to discover whether any of them caches a copy of the requested resource. In the positive case, the recipient edge server replies with an exact hit or with a useful hit message; otherwise, it may reply with a miss message or not reply at all. In the case of useful hit, the response message should provide some information about the available version of the resource to allow its retrieval. As the protocol for our query-based cooperation, we use the popular ICP adopted in NetCache and Squid [19]. In our ColTrES 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.

Directory-based protocols are conceptually more complex than query-based schemes, especially because they include a large class of alternatives, being the the two most important ones 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. Examples of compression that reduce the message size are the Bloom filters, used by Summary Cache [8] and Cache Digests [16], that compress the cache indexes so that 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 ColTrES 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.

5.2 Cooperative transcoding algorithms

Cooperative transcoding is necessary only when a local or a remote useful hit occurs, while misses and exact hits are handled as described in Section 3 and they are unrelated to the cooperative transcoding algorithms. We can identify



(a) Blind-lazy algorithm.

(b) Blind-active algorithm.

Figure 1: Load-blind algorithms.

some alternatives in the case of local and remote useful hits. 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 to decide which node will perform the transcoding task. We propose two load-blind algorithms and a local load-aware algorithm.

The two load-blind algorithms are 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 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.

The **load-aware** algorithm we propose in this paper is based on load information at the local server. When a local or a remote useful hit occurs, the node decides whether to perform locally the transcoding operation or to continue the discovery phase on the basis of its current load. This is a typical threshold-based algorithm that follows one of the two previous algorithms depending on the server load (i.e., CPU utilization). 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.

6 Workload model

In this section we describe the client and workload models used to evaluate the performance of the cooperative schemes. 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, 7]. 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; the description of the devices capabilities and the values of their popularity 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 [6]. In our experiments we also consider a workload scenario where the transcoding operations have higher costs. This may be found in a near future when the Web will provide a larger percentage of multimedia resources.

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. Some characterizations performed on the images of this workload, such as file size, JPEG quality factor, and colors of GIF images, evidenced that they are very close to the characteristics reported in [6]. 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 second workload model (called **heavy trans-load**) 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. In both workload models, the client request distribution among the edge servers is uniform, with each node receiving the same number of client 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.

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. The total size of the original resources for all workloads is similar. It is 10% higher than the sum of the cache sizes of the nodes used in our experiments. On the other hand, the mean file size of the two workloads differs considerably. 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 requests refers to this hot set.

To study the performance of the cooperative transcoding algorithms, we had to consider a scenario where the servers of the intermediate infrastructure are under heavy stress. To this purpose, we used two workload models (called **uniform** and **bimodal**) which are based on the heavy trans-load, but are characterized by different client request distributions. In the uniform scenario each edge server receives the same number of requests, while the bimodal scenario is characterized by an uneven request distribution among the edge servers, where 50% of the servers receive 90% of the client requests and the remaining half of the nodes handle only 10% of the traffic.

7 Experimental results

In this section we first outline the performance metrics and the server-side experimental setup and then discuss the experimental results. As main performance metrics we consider the *cache hit rates* (local, global, exact, useful), the *CPU utilization* of the servers, and the *system response time* that corresponds to the interval elapsed 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 [10]. We also consider only complete transcoding relation graphs, where each version can be obtained 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.

7.1 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 a possible system bottleneck. Hence, the global cache hit rate may impact the response time.

We consider the **Hierarchical leaf** and **Hierarchical root** schemes for the hierarchical architecture, the query-based (**Flat query-based**) and summary-based (**Flat summary-based**) for the flat architecture.

The hierarchical architectures 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 leave nodes receive their requests. The configuration for Flat query-based and Flat summary-based are based on ICP and Cache Digests protocols, respectively, and a flat cooperation scheme, where all edge servers have sibling relationships among them. For a fair comparison, in this set of experiments the flat schemes use the blind-active algorithm as the hierarchical schemes.

In these experiments we use both light trans-load and heavy trans-load workloads. First, we evaluate the cache hit rates, and then we focus on the system 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 describe Table 1 and use the results in Table 2 to confirm our observations or to evidence differences. From the last column of Table 1 we can observe that there are some significant differences in the global hit rates, depending on the used cooperation mechanism. In particular, Flat query-based provides the best results, while Flat summary-based turns out to be less effective in finding hits. Flat summary-based performance deteriorates because the Cache Digests protocol tends to become imprecise (i.e., the accuracy of the exchanged cache digests decreases) and its remote hit rates diminish. This is particularly evident for the heavy workload, but it can be also observed for the light workload: the presence of larger objects causes faster changes in the caches, having as a consequence a reduction of the accuracy of the 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.

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

	Local exact	**Local useful**	**Remote exact**	**Remote useful**	**Global**
Flat query-based	19.4%	16.9%	13.8%	19.3%	69.4%
Flat summary-based	21.2%	11.9%	11.5%	11.5%	56.1%
Hierarchical root	17.9%	6.8%	7.1%	7.7%	39.5%
Hierarchical leaf	10.2%	8.2%	19.6%	n/a	38.0%

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

	Local exact	**Local useful**	**Remote exact**	**Remote useful**	**Global**
Flat query-based	5.1%	4.7%	20.3%	22.1%	52.2%
Flat summary-based	5.3%	4.6%	10.3%	8.9%	29.1%
Hierarchical root	6.3%	4.7%	5.2 %	4.4 %	20.6%
Hierarchical leaf	6.1%	4.3%	11.6%	n/a	22.0%

The two hierarchical schemes achieve similar global hit rates (last column of Table 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 model, because the object sizes in the heavy trans-load model are much larger. The lower hit rate of the heavy trans-load model increases the replacement activity, thus reducing the hit rate of Flat summary-based. For this reason, the reduction in remote hit rates of this scheme, which has been already observed for the light trans-load model, is even more evident from columns 4 and 5 of Table 2.

We now pass to consider the response time. Figures 2 and 3 show the cumulative distribution of system response

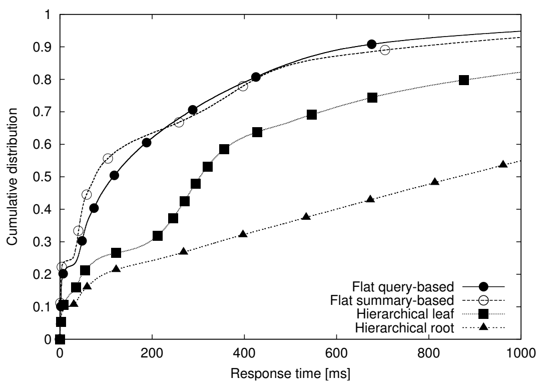


Figure 2: Cumulative distributions of system response times (*light trans-load*).

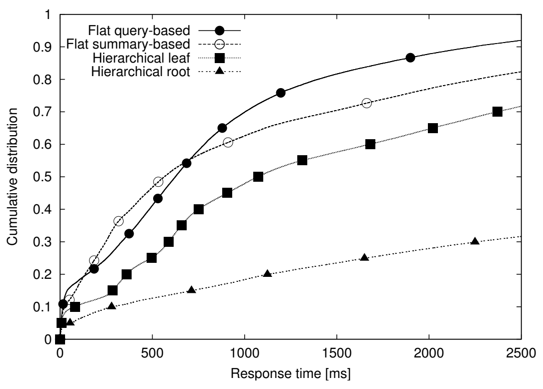


Figure 3: Cumulative distributions of system response times (*heavy trans-load*).

time for the considered schemes under the light trans-load and heavy trans-load workloads, respectively. Most of the curves shows several steps as a consequence of the different kinds of cache hit (that is, remote vs local, and useful vs exact) during the discovery phase. All the 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 have even longer response times, but do not show evident steps on the response time curve because of the larger variance in the response time. Misses generate the highest response times, hence they are typically located in the right side of each curve.

The two figures confirm that the hierarchical leaf scheme clearly outperforms the hierarchical root architecture. However, none of the hierarchical schemes can compete with flat architectures. The expected bad performance of the hierarchical root scheme is due to the bottleneck of the root node. We observed that the higher levels of the hierarchy are often overloaded because they have to handle transcoding operations of 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 model, 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 workload models), thus waiting for the upper-level nodes to process their requests.

The hierarchical leaf scheme achieves better performance: the response times in Figures 2 and 3 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 Flat query-based and Flat summary-based is in [3]. With the experiments carried out in this paper we confirm the previous observations: the higher global hit rates of Flat query-based tend to reduce the response time of the resources that are found in the nodes of the intermediate architecture. On the other hand, due to the faster lookup mechanism of Flat summary-based, remote hits are typically served faster than those of Flat query-based. For this reason, it seems interesting to analyze the cumulative distribution of the response time. Table 3 provides a summary of data in Figures 2 and 3. It shows the median (50-percentile) and 90-percentile of the response time for each cooperation scheme and both workload models.

Figure 2 shows that the difference between the two curves of Flat query-based and Flat summary-based is slight, with the former only slightly superior to the latter on the right side of the graph. This result occurs even if the global hit rate of the two flat schemes differs significantly (69.4% vs. 56.1%). Moreover, if we analyze the median response time, we can see that Flat summary-

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

	Light trans-load		Heavy trans-load	
	median	90-perc.	median	90-perc.
Flat query-based	0.11	0.64	0.62	2.24
Flat summary-based	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

based is faster than Flat query-based (also shown in the column 2 of Table 3). This can be explained by the high lookup time required by the Flat query-based scheme. On the other hand, under the heavy trans-load model (Figure 3) the curves of response times are more differentiated, with Flat query-based outperforming Flat summary-based. This result is due to the higher difference in their cache hit rates (52.2% vs. 29.1%) that cannot be compensated by the faster lookup of Flat summary-based. However, even in this case the median response time for Flat summary-based is the lowest.

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

7.2 Cooperative transcoding algorithms

In this section we compare the performance of the cooperative transcoding algorithms for flat architectures described in Section 5.2. For this set of experiments we choose the Flat flat-based scheme, because it typically offers the highest cache hit rates and lowest response times. Indeed, the flat-based scheme performs well for a wide range of workload models, 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 the server CPUs are nearly always busy due to transcoding operations. To this purpose, we used the bimodal and uniform workloads described in Section 6.

Figure 4 shows the cumulative distribution of response time for the load-blind and load-aware algorithms with the bimodal workload, while Figure 5 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 the activation parameter. We performed experiments for 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 the 90-percentile of response time. Therefore, Figure 4 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 5 the curve of the load-aware algorithm corresponds to the same load threshold value (0.66) used in Figure 4. Under the uniform workload, the curve corresponding to the best threshold value (0.9) is in between the curve of the blind-active algorithm and the one 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 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.

Table 4: System response times for load-blind and load-aware algorithms [sec].

	Bimodal workload		Uniform workload	
	median	90-percentile	median	90-percentile
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

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

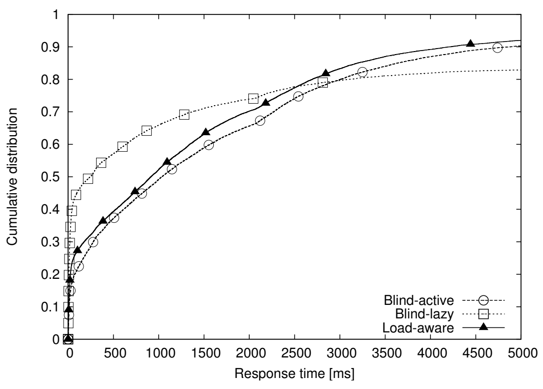


Figure 4: Cumulative distributions of system response times (*bimodal workload*).

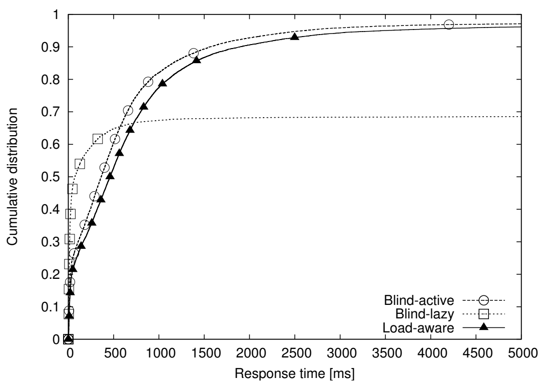


Figure 5: Cumulative distributions of system response times (*uniform workload*).

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 (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 requests being served and not to the requests directed to other peers. The response time has a more definite upper bound, thus reducing the 90-percentile of the response time with respect to the blind-lazy algorithm. On the other hand, the median response is higher than that of the lazy policy.

The load-aware algorithm offers some performance gains 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 skew of the workload is low, we need a more sophisticate algorithm, possibly based on information on the load of a larger (maybe entire) set of servers of the intermediate architecture.

8 Conclusions

In this paper, we have proposed an intermediate distributed architecture for cooperative caching and transcoding that can be implemented in the existing Web infrastructure. We have investigated various schemes that use different server organizations (hierarchical, flat), and different cooperation mechanisms for resource discovery (query-based, summary-based) and transcoding (load-blind, load aware). We have compared their performance through ColTrES , a flexible prototype testbed based on Squid that implements all proposed mechanisms. From the performance evaluation, we have found that flat peer-to-peer topologies are always better than hierarchical schemes, because of bottleneck risks in the higher levels of the hierarchy combined with limited cache hit rates. Among the flat cooperation schemes, we evaluated multi-version lookup extensions of Cache Digests and ICP and found that, ICP tends to have better performance due to the lower cache hit rates of Cache Digests. As a further contribution of this paper, we verified that the proposed load-aware algorithm can achieve some performance gains only when the client load is unevenly distributed among the edge servers of the intermediate infrastructure. On the other hand, in the case of rather uniform load distribution, the load-aware algorithm does not seem to achieve any significant improvement.

An intermediate infrastructure of distributed servers that cooperate in multi-version content caching, discovery, and transcoding opens many research topics. 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 other research issues that this paper opens up, such as cooperative cache replacement policies for multi-version content, transcoding policies based on global information on server load and available network bandwidths, and integration with server-direct transcoding to preserve the end-to-end content semantics.

Acknowledgements

The first four authors acknowledge the support of MIUR-Cofin 2001 "High-quality Web systems" and MIUR-FIRB "Wide-scale, Broadband, Middleware for Network Distributed Services".

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 Comp. Eng., 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*, pages 66–70, 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, C. S. Ellis, and A. Vahdat. Application-level differentiated multimedia Web services using quality aware transcoding. *IEEE J. on Selected Areas in Communication*, 18(12):2544–2465, Dec. 2000.

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

[7] C.-Y. Chang and M.-S. Chen. On exploring aggregate effect for efficient cache replacement in transcoding proxies. *IEEE Trans. on Parallel and Distributed Systems*, 14(6):611–624, June 2003.

[8] 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.

[9] 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 Princ.*, pages 78–91, Oct. 1997.

[10] 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.

[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] W. Shi, K. Shah, Y. Mao, and V. Chaudhary. Tuxedo: a peer-to-peer caching system. In *Proc. of the 2003 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'03)*, Las Vegas, NV, June 2003.

[18] A. Singh, A. Trivedi, K. Ramamritham, and P. Shenoy. PTC: Proxies that transcode and cache in heterogeneous Web client environments. *World Wide Web*, 2003.

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

[20] 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 Princ.*, Dec. 1999.