World Wide Web: Internet and Web Information Systems, 9, 63–92, 2006
© 2006 Springer Science + Business Media, Inc. Manufactured in The Netherlands.
DOI: 10.1007/s11280-005-4049-9

Content Adaptation Architectures Based on Squid Proxy Server

CLAUDIA CANALI claudia@weblab.ing.unimo.it
Department of Information Engineering, University of Parma, Parma, Italy 43100

VALERIA CARDELLINI cardellini@ing.uniroma2.it
Department of Computer Science, Systems and Production, University of Roma Tor Vergata, Roma, Italy 00133

RICCARDO LANCELLOTTI lancellotti.riccardo@unimore.it
Department of Computer Engineering, University of Modena, Modena, Italy 41100

Published online: 2 October 2005

Abstract

The overwhelming popularity of Internet and the technology advancements have determined the diffusion of many different Web-enabled devices. In such an heterogeneous client environment, efficient content adaptation and delivery services are becoming a major requirement for the new Internet service infrastructure. In this paper we describe intermediary-based architectures that provide adaptation and delivery of Web content to different user terminals. We present the design of a Squid-based prototype that carries out the adaptation of Web images and combines such a functionality with the caching of multiple versions of the same resource. We also investigate how to provide some form of cooperation among the nodes of the intermediary infrastructure, with the goal to evaluate to what extent the cooperation in discovering, adapting, and delivering Web resources can improve the user-perceived performance.

Keywords: content adaptation, proxy servers, distributed systems, performance evaluation

1. Introduction

In the last decade the Internet has become the infrastructural support to many different services and has penetrated deeply in the daily life of a huge number of users. This overwhelming popularity, combined with the contemporary technology advancements, allows users to be Web-connected through many pervasive computing devices, such as laptops, handheld computers, Web-TVs, personal digital assistants (PDAs), and cellular phones. The major differences among these emerging devices regard network connectivity, processing power, storage, display, and data handling capabilities. Hence, there is an increasing demand for content adaptation and delivery services that enable on-the-fly transformation of (possibly) complex Web content and its delivery to these diverse destination devices.

The process of converting a multimedia resource from one form to another to match the device characteristics is called *content adaptation* or *transcoding*. It can be applied to transformations within media types (e.g., changing size and color depth of an image, converting from high-fidelity JPEG to low-fidelity GIF format), across media types (e.g., from speech to text, from video item to a set of images) or to both of them. Independently of the type of adaptation, the transcoding process should preserve the semantic attributes of the original resource and produce a version which may be consumed by the client device.

The existing approaches for Web content adaptation fall into three broad categories depending on the entity that performs the adaptation process [2,19]: *client-side*, *intermediary-based*, and *provider-side* adaptation.

In the *client-side* solution, the required content adaptation occurs on the client device. Carrying out the adaptation process at the client side avoids communicating information about the device characteristics and allows to use proprietary adaptation technology. Although the processing power, storage, and connection bandwidth of the client devices are continuously improving, their capabilities will always be more limited than those of server machines. This does not allow to perform locally the most complex adaptation tasks that require large amount of computing and storage resources. Another problem of the client-side approach is that it does not save network bandwidth (and transmission time), because the resource is delivered to the client in its original size, which is usually larger than the adapted version [24].

In the *provider-side* approach [14,22,23], the functionalities of the content provider platform are enhanced with adaptation tools. Typically, the content transformation is generated off-line when the content is created or updated, often involving a human designer to hand-tailor the content to some specific requirements. The multiple variants of the same resources are stored on the server and selected to match the client-related information [22]. XSLT is also used to generate the appropriate markup language from a structured XML representation [14,23]. The provider-side adaptation lets to the content author the maximum control over the delivered content. The drawback of this approach is that the transcoding cost completely burdens the server platforms which risk of being overloaded [24]. Our experience with tools such as Mercury LoadRunner [21] leads us to conclude that 8–10 simultaneous requests involving adaptation may overload a server. Hence, the provider-side approach is a valid solution when the provider popularity is medium-low.

In the *intermediary-based* approach [5,12,13,15,20,33], an intermediary node (initially called proxy server, nowadays edge server), that is interposed between the client device and the content provider, analyzes and adapts the requested content on-the-fly, before delivering the result to the client. Intermediary-based adaptation shifts computational load away from the servers of the content providers, simplifies their design, and reduces considerably the user-perceived latency when the adapted content can be served from a node closer to the client than the content server. It is also a viable solution because practically all client devices requiring content adaptation need a gateway to use Web-based services. The motivation of this paper comes from the observation that a possible problem of the intermediary-based solution is that traditional transcoding nodes operate singularly. A single intermediary node is affected by a limited scalability [20] because of the significant computational cost of transcoding operations and the explosion of the working set due to the presence of multiple versions that may be generated from the same original resource. A distributed architecture of cooperative intermediary nodes can address these issues in the research area of delivering adapted content.

In this paper, we describe the architecture and the implementation details of a Squid-based prototype that provides a twofold enhancement of the traditional cooperative caching systems to match the requirements of an environment characterized by heterogeneous client devices. The first enhancement transforms a traditional proxy server into an active intermediary node, which not only caches Web resources but also adapts them, stores the result, and allows multi-version lookup operations [7,28]. The second novel enhancement allows intermediary nodes to cooperate in locating, adapting, and delivering Web content. We describe the modifications necessary to make the intermediary nodes cooperate on the basis of hierarchical and flat distributed schemes when multiple versions of the same resource are available.

Although the cooperative content discovery in the context of adaptation services is based on distributed schemes, it is not a simple extension to traditional cooperative caching because two main issues have to be addressed to achieve a suitable solution. First, the presence of diverse target devices requires to identify, discover, and cache multiple variants of the same original resources. Therefore, we need efficacious solutions to support the multi-version cooperative discovery. Moreover, the distributed architecture should be able to share the load among multiple nodes, because the transcoding operations may be computationally expensive.

It is worthwhile to note that we consider intermediary-based architectures which provide adaptation services for the Web resources of any content provider. The content servers act only as providers of the original Web resources and do not perform any content adaptation operation. As a consequence, in this paper we do not rely on privileged accesses to content provider information for the deployment of the adaptation service, including metadata and already adapted resources. Moreover, we do not exploit mechanisms such as server-directed adaptation [18], that allows the content provider to give directives to guide the transcoding choices. However, our prototype can be easily modified to support the solution of an intermediary infrastructure that operates in collaboration with the content provider, as in the case of Content Delivery Networks.

The rest of the paper is organized as follows. In Section 2, we describe the main functionalities characterizing an intermediary node that provides adaptation, multi-version caching, and lookup of Web content. In Sections 3 and 4, we analyze in detail what prototype functionalities are required at the level of stand-alone and multiple cooperative nodes, respectively. In Section 5, we describe the workload model that we have used to exercise the Squid-based prototype. In Section 6, we present experimental results that demonstrate the different functionalities of our prototype. Finally, we discuss related work in the area of content adaptation carried out by intermediary nodes in Section 7, and conclude in Section 8 with some final remarks.

2. Functionalities of the intermediary nodes

In this section we describe the peculiar features of an intermediary node in a generic architecture for content adaptation. The considered functionalities transform a traditional proxy from a resource repository along the delivery chain between the client and the content server into an enhanced network node which provides cooperative content adaptation services.

Let us first introduce the functionalities that a node belonging to a distributed architecture for content adaptation and delivery may or must provide and then discuss them. We have identified the following basic functionalities for a single non-cooperative node:

- *Gateway*: to receive the client request, identify the device capabilities and characteristics, and provide the response that fulfils adaptation requirements without the intervention of any visible component for the client.
- *Adaptation*: to adapt, when necessary, the original resource to the specific capabilities of the client device that has originated the request.
- *Fetch*: to retrieve the original resource from the content server, when a valid version of the requested resource is not found in the local cache.
- *Cache*: to manage in the cache original copies and adapted versions of previously requested resources.

In addition, a cooperative intermediary node also has to provide the *location* functionality to identify one or more remote nodes that may provide the requested content and/or have a valid version of the resource. The cache functionality collaborates with the location one to activate the external resource lookup on other nodes. Furthermore, in a cooperative node the fetch functionality is activated only when no intermediary node holds a valid version of the requested resource. In this paper, we consider that each functionality is provided by a corresponding node component, which is named in the same manner. Figure 1 illustrates the node components and the relationship among components of different intermediary nodes.

When the intermediary node receives a client request, the first task of the gateway component is to identify the device capabilities and characteristics to decide which version of the resource fits the client needs. The features of client devices vary widely in

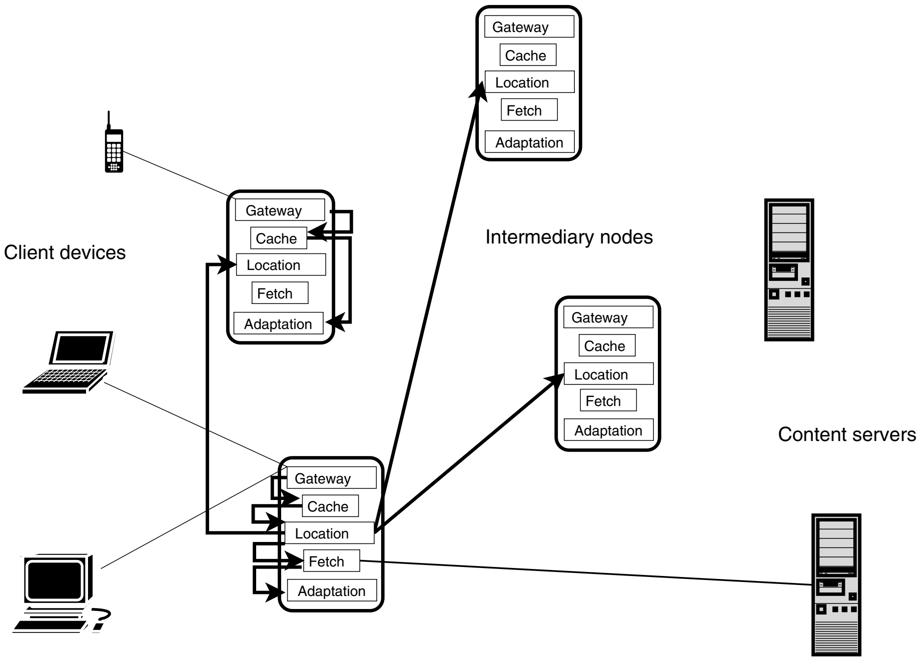

Figure 1. Components of the intermediary-based architecture.

screen size and colors, processing power, storage, user interface, and software. Client's access links to the Internet also range from wired networks, such as LAN, xDSL, ISDN, and telephone modems, to wireless networks, such as GSM, CDPD, and UMTS. The client may include the resource data type it can consume as meta-information into the HTTP request header, taking advantage of the potentialities of the protocols already in use. Recently, the Open Mobile Alliance and the W3C have also proposed the compatible standards CC/PP and UAProf which allow a client device to communicate its characteristics and capabilities [2]. The gateway component can also obtain the information about the client capabilities by accessing a table which stores the characteristics of the various client devices that may be served by that intermediary node. For example, a table entry for a particular client device can be created/stored when the client first registers with the assigned node. This entry can be transmitted to the other cooperative nodes, for example by including it in the client request. Therefore, we can suppose that the intermediary node is aware of the characteristics of the client devices. Hereafter, we will refer to the information describing the capabilities of the requesting client as *requester-specific capability information* (RCI).

After having identified the client capabilities, a resource version which fits the client request has to be obtained. First, the cache component looks for the requested resource in the local cache system. Multiple versions of the same resource are originated by the transcoding process and may be cached by the intermediary node. A resource which was already adapted may be further transcoded to yield a lower quality resource. In particular, each version may be transcoded from a subset of the higher quality versions. Different versions of the same resource (and the allowed transcoding operations among them) can be represented through a *transcoding relation graph* [4]. In this paper, we consider a complete transcoding relation graph, assuming that each version can be obtained from any more detailed version.

Due to the presence of multiple versions of a given resource, the lookup phase differs from the corresponding one of a traditional caching system. Therefore, it is necessary to perform a multi-version lookup, which may cause one of the following three events:

- *Exact local hit*: the cache contains the exact version required by the client. The gateway component can immediately deliver the resource to the client without any content adaptation.
- *Useful local hit*: the cache contains a more detailed and transcodable version of the requested resource that can be transformed by the adaptation component to obtain the requested version before sending the response to the client.
- *Local miss*: the cache does not contain any exact or adaptable version of the requested resource. In case of cooperation, an external multi-version lookup has to be managed by the location component. If no remote (contacted) node contains any valid version of the requested resource (or the cooperation has not been activated), the fetch component retrieves the original version from the content server.

The main modifications to the traditional behavior of an intermediary node caused by the introduction of content adaptation and multi-version caching are summarized below and will be discussed in more detail later:

- the identification of the client device capabilities;

- the identification of multiple versions of the same resource;
- the provisioning of transcoding functionalities to adapt resources on the basis of information regarding the client device;
- the handling of multiple versions of the same resource in the cache of each intermediary node;
- the multi-version lookup at local as well as global level.

Our prototype is implemented as a modification of the freely available Squid 2.4 software [29]. Squid is an open source, high-performance proxy caching server. We chose Squid as a basic platform for its robustness and wide spread usage. Moreover, as Squid can support many cooperation mechanisms, such as Internet Cache Protocol [32] and Cache Digests [25], the multi-version cooperative lookup can be implemented by modifying the existing code instead of writing it from scratch. Since our intermediary node can operate not only singularly, but also in a cooperative way on the basis of a hierarchical or flat distributed cooperation scheme, its behavior has been modified both at the single node level and at the cooperation level. The modifications to Squid have been brought in such a way to leave unchanged its standard behavior when content adaptation is not required.

3. Interventions at the single node level

In this section we describe the main modifications we have brought to the original Squid to supply the multi-version caching and adaptation functionalities in a single intermediary node. Our prototype presents some substantial differences with respect to the original Squid architecture [31]. As regards the Squid code, the most significant changes regard the Squid modules called *Client Side* and *Storage Manager*, which handle client request processing and caching operations, respectively.

3.1. *Identifying the client device*

Our prototype aims at tailoring Web content to match the device capabilities and characteristics as much as possible and using all available information contained in the client request. We have chosen not to implement a specific protocol for communicating the client capabilities, but rather to use the HTTP protocol to let the client communicate the resource data types it can consume.

The *Client Side* Squid module, which is in charge of parsing the client request [31], has been modified in order to handle the RCI. To this purpose, we use the weak consistency of the entity tag (`ETag`), which is an identifier used to compare two or more versions of the same resource [11]. A numerical code, contained in this field, indicates the required version of a resource. By including this information into the HTTP request header, we avoid altering the current communication mechanism between clients and intermediary nodes. Each client request contains the request-header field `If-Match` with a weak `ETag` which indicates the required version of a resource. When the request header is processed by the intermediary node, the gateway component checks whether there is the field `If-Match` and extracts from it the weak `Etag`. A code value different from zero means that the required version is not the original one: in this case, the code value is stored in the data structure `clientHttpRequest`, which keeps information regarding each client request.

We have chosen this approach to simplify the testing of our prototype. However, it is possible to use any information included in the client request, such as another HTTP header or the client IP address, since the gateway component identifies the requested version by analyzing the client HTTP request.

A note is in order about the choice of the resource version that satisfies the client request. In this paper, we adopt the *exact matching* scheme that is, the client receives the resource version that has been identified as the most suitable for the client capabilities. Other schemes are possible, although not considered in this paper. For example, in the *no-more-than matching* scheme, the client may receive a resource that can have a lower quality than that requested. The *at-least matching* approach assumes that the client device has some transcoding ability to handle even higher quality resources and to adapt them to its precise capabilities. Both such schemes can save CPU resources by avoiding transcoding operations when a slightly less or more detailed version of the resource is found in some cache. The at-least matching approach can also improve cache hit rates because requests for different versions of the same resource can be satisfied using only a limited number of cached versions. However, we are interested in evaluating the performance of the content adaptation architectures also in case of high computational requirements. Hence, in this paper we focus our attention on the exact-matching scheme, which puts more stress on the architecture.

3.2. Identifying multiple versions

Squid identifies every resource using its relative URL, through which it elaborates the hash key for finding the relative `StoreEntry`. This is the data structure that keeps track of the resources stored in the cache of the intermediary node.

Since the transcoding process implies the presence of multiple versions of the same resource, the URL alone is no more sufficient to identify a resource, because it would be identical for any version. To manage multiple versions of the same resource, we have decided to modify the relative URL by adding the string "/X" to it, where X represents the numerical identifier of a particular version. This modification has some impact on the *Storage Manager* Squid module, which is responsible for handling the cache content. The URLs encoded with the version identifier are completely transparent to the clients as well as to the content servers because they are only used within the intermediary architecture (specifically, for the internal representation of the resource in the Squid data structures and the flat cooperative protocols described in Section 4.2).

As shown in Figure 2, the gateway component identifies the requested version by parsing the HTTP request header. If the `If-Match` field is present, the numerical code contained in the `Etag` is extracted and added to the original URL.

3.3. Adapting a resource

To provide the adaptation functionality, our prototype uses an external process, called *Transcoder*, which is created when necessary and exchanges information with Squid

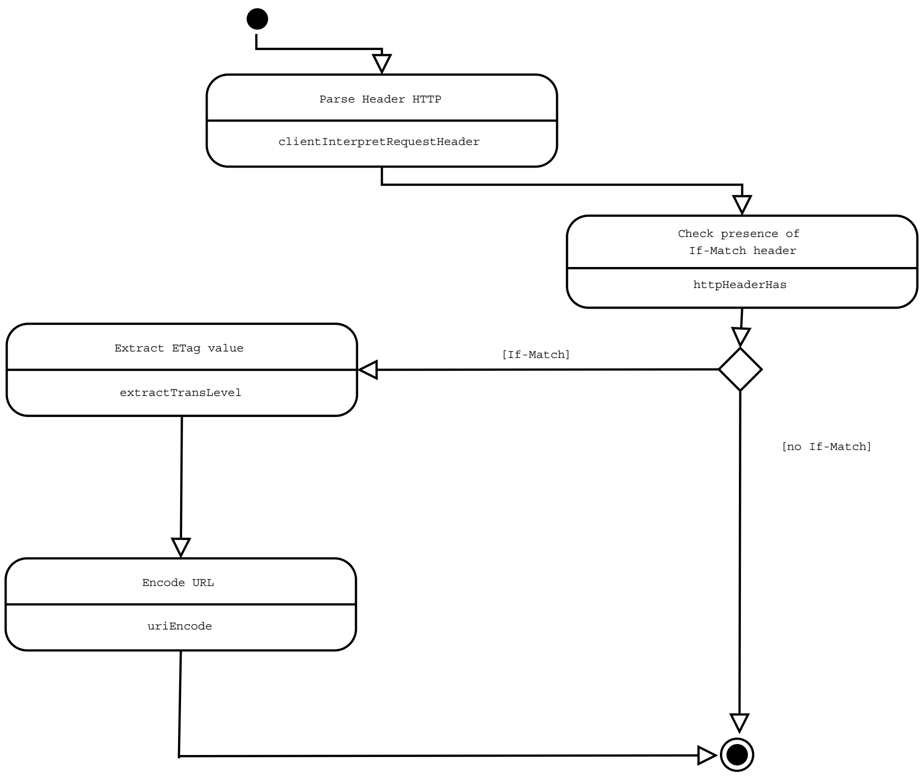

Figure 2. Parsing of the HTTP request.

through a pipe. The motivation for this choice is that, unlike traditional caching software, Squid handles all the requests in a single, non-blocking, I/O-driven process. Since transcoding operations are generally longer than any other typical Squid task, it would not be efficient to include them in the `main()` program. Our solution allows to perform in parallel multiple transcoding operations. Moreover, the use of multiple instances of the Transcoder process can increase the performance in SMP systems (our experience is on bi-processor machines). For example, we found that on a bi-processor SMP system this approach reduces the 90-percentile of the transcoding time up to 110% with respect to a uniprocessor system.

An alternative approach to multi-processing is to use multiple threads for serving the client request. The use of threads has a clear benefit when there is a high number of short client sessions. This is the typical case of HTTP servers serving static Web resources, for whom the performance improvement introduced by multi-threading is significant (e.g., Apache 2.0 vs. Apache 1.3 servers [1]). However, we should consider that a transcoding intermediary node is characterized by a more limited number of long-lasting sessions, because the transcoding operations add a computational impact that traditional HTTP servers do not have to support. Hence, for a transcoding node the benefit achieved from multi-treading over multi-processing may be negligible. For this reason, we preferred the simpler approach of using a separate transcoding process instead of deploying a multi-threaded transcoder. To motivate our choice, we also carried out some preliminary tests where the transcoding operation is a file compression. We processed a set of images using both the fork system call and threads to carry out transcoding. We found that the average processing time is 151 ms for the thread-based solution and 152 ms for the fork-based approach. Therefore, the performance gain of multi-treading over multi-processing is less than 1%. For more complex transcoding operations this performance gain can be even lower. Indeed, Figure 7 shows that the adaptation task can take a time up to seconds.

Squid and the Transcoder process use temporary files to exchange data. First, the transcodable resource is written on a temporary file, then an external process is created and a communication pipe is opened through the Squid Inter Process Communication functions. Squid uses other external processes, like pinger or unlinkd, so that our solution is completely consistent with the Squid code. When the adaptation is over, the Transcoder process notifies Squid through the pipe that the adapted resource is available on the temporary file. Figure 3 illustrates the interaction among the processes: the continuous lines represent data exchanges, the dashed ones the execution flow.

The Transcoder process uses the freely available ImageMagick library [16] to adapt the image resources to the client capabilities. This library allows to manipulate many image formats, supplying numerous transcoding algorithms. To extend the adaptation functionalities to different media types, other transcoding tools can be used with simple software integration. Indeed, the adaptation component is implemented as an external

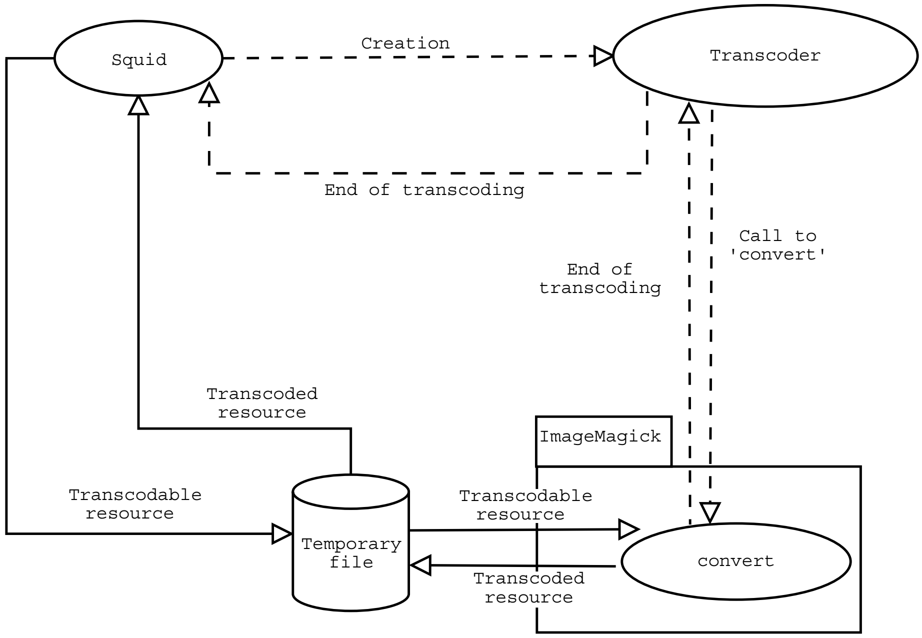

Figure 3. Interaction among processes.

process with the specific purpose of maximizing flexibility and making the prototype independent of the transcoding functionality. The manipulation methods for adapting media resources are a function of their current type (HTML, image, video, and audio) and the client device characteristics.

3.4. *Caching multiple versions*

After having performed the transcoding operations, the intermediary node has to decide which resource version is more useful to cache. To avoid repeating the likely expensive transcoding operations at the same node, the output version resulting from adaptation may be cached. On the other hand, the input version, on which the adaptation has been applied, may be useful to serve a larger number of devices being more detailed.

Three simple caching policies have been described in [4] to manage the caching of multi-version content. In the *demand-based* policy, the intermediary node caches the output version resulting from adaptation; in the *coverage-based* policy, the node caches only the input transcodable version retrieved from its local cache or from another remote node; finally, in the *anticipatory* policy, the node caches both the versions.

Our prototype adopts the anticipatory policy. Hence, the Squid caching mechanism has been modified to cache both versions of the same resource, before and after transcoding. Figure 4 shows the sequence of operations performed when the intermediary node receives an HTTP reply from either a remote cooperative node or the content server.

The Squid standard behavior, according to which the resource caching is carried out contemporaneously to the resource delivery, has been modified in the following way. The transcodable version is both cached and written on a temporary file at the same time. After the transcoding process, a new `StoreEntry` structure is created to save the adapted version in the local cache. As shown at the bottom of Figure 4, the caching of the transcoded version happens contextually to its delivery. As replacement algorithm our prototype adopts the standard LRU. It is worthwhile to note that the Squid caching mechanism has been preserved; hence, every cache replacement policy already implemented in Squid can be used in our prototype without any software integration. Moreover, our prototype represents an extensible framework that also allows to integrate more sophisticated cache replacement policies as those described in [7,28].

4. Interventions for cooperation

The use of a distributed system of intermediary nodes as an infrastructure for cooperative content adaptation and delivery opens up many possible alternatives, because the nodes can be organized according to different topologies and for each specific topology various cooperation mechanisms can be defined [24]. In this paper, we consider two feasible architectures that is, the *hierarchical* and the *flat* ones. In a *hierarchical* architecture, the nodes are organized in a hierarchy of different levels [8], where only the bottom level (*leaf*) nodes handle client requests directly. In a *flat* architecture, all the nodes are placed on the same level. In the following of this section, we describe the main modifications brought to Squid with the aim of providing content adaptation and delivery services in a cooperative environment.

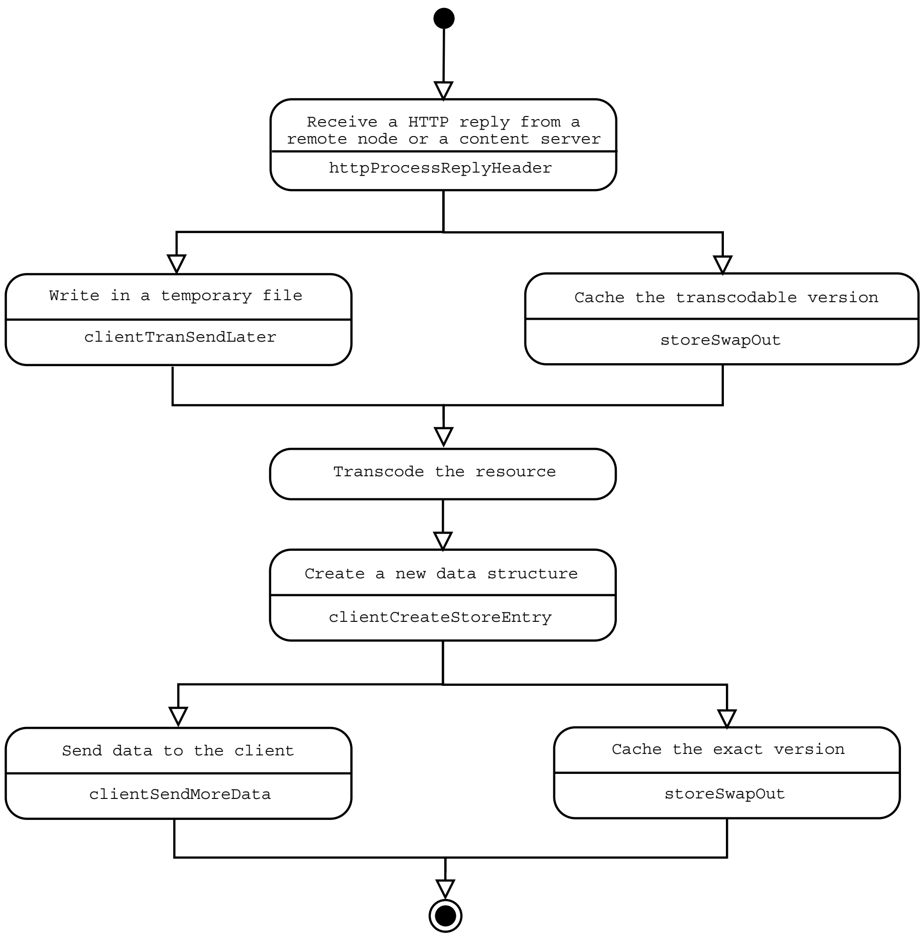

Figure 4. Anticipatory caching.

4.1. Hierarchical topologies for cooperative content adaptation

In a *hierarchical* caching architecture, the device-contacted leaf node forwards the client request up to the hierarchy until an hit occurs. If no hit occurs at any level, the root node retrieves the original version of the requested resource from the content server. We assume that only the leaf nodes host the gateway functionality to manage client requests, while the fetch functionality is played only by the root node. The cache and location functionalities are provided by all the intermediary nodes, although no cooperation occurs among sibling nodes at the same level. Message and resource exchanges occur only along the vertical direction between a parent and a child node. As a consequence of the presence of a cache in each node of the infrastructure and the vertical way in which the location task is managed by the nodes, the global working set may be replicated on each path from a leaf node to the root node, thus causing a scarce usage of the global storage resource and leading to low cache hit rates.

The introduction of the content adaptation task in a hierarchical architecture opens up the issue of mapping such a task on the nodes of the hierarchy. It is possible to realize an homogeneous scheme where each node of the hierarchy provides both *adaptation* and *cache* functionalities, as well as to consider a scheme where nodes at different levels provide differentiated functionalities.

As transcoding may involve computationally expensive operations, in a hierarchical architecture where all nodes perform adaptation there is a great risk of overloading upper level nodes and particularly the root node. Indeed, preliminary results obtained by the authors [3] show that an homogeneous scheme achieves poor performance due to overload in the upper level nodes, which have to handle all misses from the lower levels. We found that the root node can achieve a mean CPU load higher than 0.90, while the leaf nodes are often idle (the corresponding mean CPU load is less than 0.02), waiting for upper level nodes to process their requests. Therefore, in this paper we consider a hierarchical scheme where the adaptation functionality is performed only by the leaf nodes, while the intermediary nodes of the upper levels and the root node of the hierarchy act only as traditional caching servers. In the latter case, the CPU load on the leaf nodes is close to 0.3, while the root node is less utilized (its CPU load is below 0.1). If compared to the homogeneous hierarchical scheme, the latter solution provides a better performance while remaining simple and easy to implement.

After the gateway component on the leaf node has received a client request and identified the desired version of the resource, the cache component on the same leaf node performs a local multi-version lookup in its cache. In case of exact hit, the node delivers immediately the resource to the client, while in case of useful hit, the node performs the transcoding operations locally before sending the resource to the client. A local miss forces the location component on the leaf node to forward the request to the upper levels of the hierarchy. That request refers to the original version of the resource, hence no content adaptation is required to the upper nodes. If the client-requested version of the resource was not the original one, the adaptation component on the leaf node performs the transformation needed to obtain the proper version, before the gateway component delivers it to the client. When a resource passes down the hierarchy, it is left in the cache of each intermediary cache, until the resource reaches the leaf node.

The hierarchical cooperation does not require further modifications to the Squid code in addition to the ones described in Section 3.

4.2. Flat topologies for cooperative content adaptation

We now consider flat architectures for intermediary-based adaptation where the nodes are peers and provide all the functionalities (*gateway*, *cache*, *location*, *fetch*, and *adaptation*). External lookup in a distributed system has been studied for a while and many mechanisms have been proposed to address the related issues [24]. Most of those mechanisms are viable solutions also for the multi-version lookup of resources. Apart from the specific lookup protocol, the existence of multiple versions of the same resource implies modifications to the cooperative discovery. Indeed, the external lookup may provide one of the following results:

- *Remote exact hit*: the exact version of the required resource is found in the cache of a remote node.

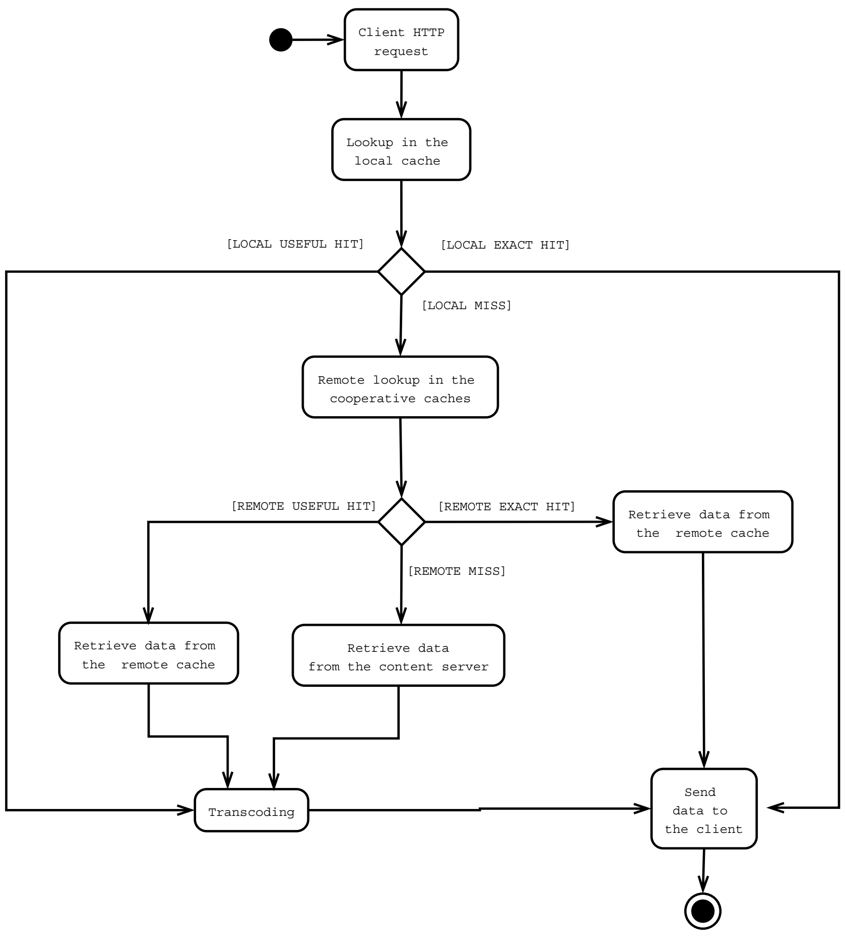
Client HTTP request
Lookup in the local cache
[LOCAL USEFUL HIT]
[LOCAL EXACT HIT]
[LOCAL MISS]
Remote lookup in the cooperative caches
[REMOTE USEFUL HIT]
[REMOTE EXACT HIT]
[REMOTE MISS]
Retrieve data from the remote cache
Retrieve data from the remote cache
Retrieve data from the content server
Transcoding
Send data to the client

Figure 5. Multi-version lookup in a flat architecture.

- *Remote useful hit*: a transcodable and more detailed version of the required resource is found in the cache of a remote node.
- *Remote miss*: no version of the requested resource is found in any node caches; in this case, a *global miss* occurs and the original version is retrieved from the content server and, if needed, transcoded.

Figure 5 shows the behavior of an intermediary node in a flat architecture when it receives a client request: it searches simultaneously for exact and useful hits, but the exact hit is preferred to the useful one as it allows to avoid transcoding operations.

In the following two sections, we consider two opposite schemes that are at the basis of different distributed architectures for intermediary content adaptation (namely *query-based* and *summary-based* schemes), and their extension to provide the multi-version location functionality.

4.2.1. Flat summary-based scheme. In summary-based schemes, as Cache Digests [25], the location component of each node stores locally information about the cache contents of the other intermediary nodes and exchanges periodically (independently of the client request arrivals) this content information with other location components to refresh its local estimation on remote cache server contents and to inform the others about changes in its own set of cached resources. When a local miss occurs, the location component on the node checks the summary of the resources cached at the other nodes to discover potential remote exact or useful hits.

To allow all the location operations that are at the basis of the flat summary-based scheme, the cache component must offer a richer set of operations than that provided in the other considered architectures. In addition to the local multi-version lookup, insertion, and delete operations, the cache component must make available to the location component the whole cache index, in order to allow content information exchange with the remote nodes.

For the experiments, we choose Cache Digests as a representative of the summary-based architectures, because of its popularity and its implementation in the Squid software [25,31]. By adding to the URL the resource version identifier, our prototype supports in a transparent way the summary-based lookup process with multiple versions of the same resource. Therefore, the basic mechanism of Cache Digests cooperation is preserved. Since the hashing mechanism used by Cache Digests is very fast, the need of one lookup for every possible useful version has a negligible impact on the overall lookup time.

4.2.2. Flat query-based scheme. Unlike a summary-based location scheme where the messages are periodically exchanged among the nodes, in the query-based cooperation the location component activates message exchanges only at lookup time. When a node experiences a local miss, the remote lookup starts with the location component of the local node issuing query messages for the requested resource to the location component on the other nodes in order to locate either the exact or a useful version of the requested resource. The most important query-based protocol is ICP, defined in [32] and implemented in Squid [31]. For this reason, we used ICP as the basic protocol for query-based cooperation.

The pros and cons of the two flat schemes should be clear. The cache component complexity of the summary-based scheme is no longer needed in the flat query-based scheme, which can rely on a simpler cache component similar to that of the hierarchical architecture. Moreover, due to the synchronous nature of query-based interactions, this scheme tends to offer more accurate information. This means that we expect a remote cache hit rate, both exact and useful, potentially higher than that of the summary-based scheme. On the other hand, the remote lookup operations of the query-based scheme are inherently slower and more expensive. The location component of the summary-based scheme has only to access locally available information, while the discovery process of the query-based scheme requires an expensive query/reply message exchange that is also subject to network conditions. Moreover, while a remote exact hit is detected as soon as the location component receives an exact hit reply message, the cases of misses and useful hits may be particularly expensive. Indeed, the location component declares a global miss and a remote useful hit only when it has received the last reply message. This means that the location component must wait for the slowest remote node to respond or the expiration of a timeout (set for default to 2 seconds in ICP).

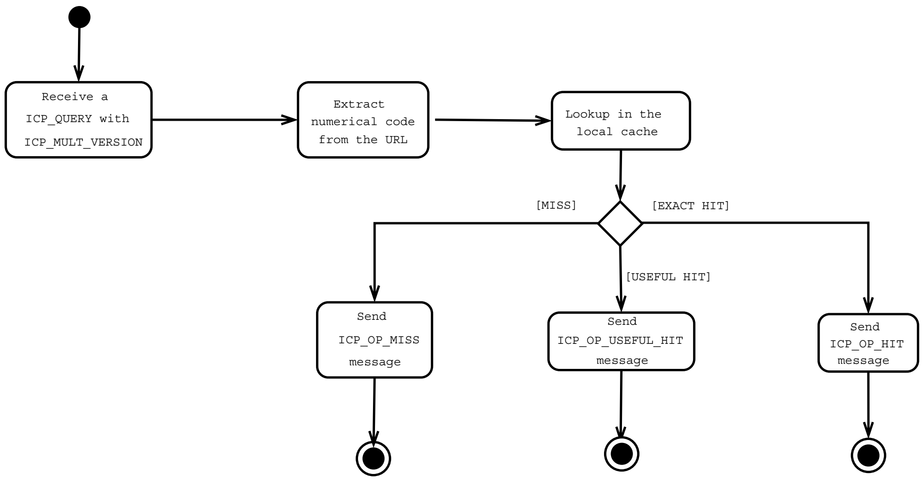
Receive a ICP_QUERY with ICP_MULT_VERSION
Extract numerical code from the URL
Lookup in the local cache
[MISS]
[EXACT HIT]
[USEFUL HIT]
Send ICP_OP_MISS message
Send ICP_OP_USEFUL_HIT message
Send ICP_OP_HIT message

Figure 6. Receiving an ICP query with ICP_MULT_VERSION flag set.

Some essential modifications are necessary to add the support for multi-version lookup into the Squid version of ICP (defined in the module *ICPv2*). ICP does not supply support for HTTP semantics: an ICP query contains only the URL of the resource, but none of HTTP headers. Hence, we insert the encoded URL with the resource version identifier in the payload of the ICP message in order to indicate the required resource version. To distinguish the multi-version lookup from the standard lookup of the ICP protocol, we define a new flag, namely ICP_MULT_VERSION. We also introduce a new response code to distinguish a useful hit from an exact hit. In the ICP_OP_USEFUL_HIT response, the URL field of the ICP message contains the identifier of the transcodable version encoded into the URL.

The multi-version enhanced ICP lookup is activated by setting the flag ICP_MULT_VERSION in the request message. Then, we have three possible response codes: ICP_OP_HIT in case of exact hit in the cache of the remote node, ICP_OP_USEFUL_HIT for useful hits, and ICP_OP_MISS when neither exact nor useful hits are detected. Figure 6 shows the ICP reply mechanism of an intermediary node upon receiving an ICP_QUERY message with the ICP_MULT_VERSION flag.

If different useful versions are detected in the remote caches, the location component on the client-contacted node tries to retrieve the least detailed version to reduce both the transmission time and the adaptation cost.

5. Client and workload models

In this section, we describe the client and workload models used to evaluate the performance of the Squid-based content adaptation architectures.

We consider a classification of the client devices on the basis of their capabilities of consuming different resources and connecting to the assigned intermediary node [4,7]. We have identified the following six classes of devices:

- *HighPC*: a high-end workstation/PC with a network link ranging from Ethernet to DSL; this device can consume every resource in its original form.
- *MedPC*: a midrange PC or a laptop connected through a fast/medium wire-connected modem; to reduce network-resource usage, the image size and the quality factor (for JPEG images) are reduced.
- *TVBrowser*: a set-top box that can turn a TV into a Web browser; as the maximum resolution is limited by the TV screen, the image size should be trimmed down not to waste network and CPU appliance resources.
- *HPC*: a handheld PC connected through a modem; it can display color images with different resolutions, but the screen is typically small.
- *PDA*: a personal digital assistant using a wireless connection; it is not capable of displaying colorful and large images.
- *Phone*: a cellular phone using a wireless GSM connection; the screen is small and can only display black and white (1 bit) graphics.

Table 1 shows the bandwidth of the network link connecting the device to the intermediary node and the different device capabilities for handling images.

To characterize the workload, it is worth making prediction about the future diffusion of different Web-connected devices. In this paper, we assume that traditional devices, such as PCs and laptops, are still more popular than other client devices. The percentages of diffusion of client devices are reported in the last column of Table 1. It is difficult to predict the trend of diffusion of future Web-connected devices. However, we found that experiments with percentages within 10% of those reported here did not affect the main conclusions of the paper.

In this paper, we consider that transcoding operations are applied only to image resources (both GIF and JPEG formats), as more than 70% of the files requested on the Web still belongs to this class [6]. However, in our experiments we also consider a likely future scenario where the transcoding operations have higher costs because of the larger percentage of non-textual resources.

To choose efficient and appropriate transcoding algorithms it is useful to determine which characteristics of Web images may be critical for displaying on devices with limited capabilities. The nature of typical images in Web workload and their transcoding characteristics have been analyzed in [6], where the authors found that 74.81% were GIF and 24.41% were JPEG images. GIF and JPEG images represent almost all the image

Table 1. Client device characteristics.

Device	Bandwidth	Color	Maximum resolution (pixels)	Percentage of diffusion %
HighPC	10 Mbps	24-bit color	original	25
MedPC	64 Kbps	24-bit color	800×600	25
TVBrowser	56 Kbps	8-bit color	640×480	12.5
HPC	56 Kbps	8-bit color	120×120	12.5
PDA	28.8 Kbps	grayscale	120×120	12.5
Phone	9.6 Kbps	b&w	120×120	12.5

bytes transferred, but the percentages are evenly distributed between the two formats. With regard to the image size, most of the GIF images accessed on the Web (about 80%) are smaller than 6 KB, while about 40% of the JPEG images are larger than 6 KB. Transcoding can be performed along a number of different axes. To obtain a file size reduction, the transcoding algorithms seem to perform better if operating on spatial geometry, color depth, quality, animation, and MIME subtype.

In our experiments we used two different workload models, namely *light* and *heavy*, being the working set the main difference between the two workloads.

The light workload aims at capturing a realistic Web scenario with a reduced content adaptation load. The set of resources used in this workload is based on proxy traces collected in August 2002 and belonging to the nodes of the IRCache infrastructure [17]. We downloaded the resources from their content servers and placed them on our content server. 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 [6]. Hence, we can assume that our workload captures a realistic scenario of present Web requests. We measured the costs of transcoding operations in the light workload by submitting a sequential stream of requests on the machines used for our experiments. The results of such test are 0.04 and 0.22 seconds for the median and the 90-percentile service time, respectively (the cumulative distribution is shown in Figure 7).

The heavy workload has a higher majority of larger files (pictures). This workload does not aim at being realistic, but we used it to put much more stress on the transcoding process. 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 them. 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 workload, the cumulative distribution of the transcoding time is shown in Figure 7).

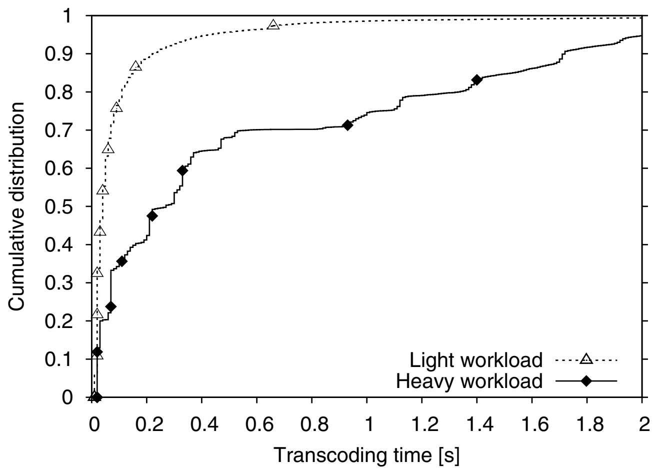

Figure 7. Cumulative distributions of transcoding time for the light and heavy workloads.

For both workloads, the size of the *original working set* (i.e., the working set composed only by the original versions of each resource) is similar. If we focus on the *actual working set*, that includes the original resources and all their adapted versions, we observe that its size exceeds significantly that of the original working set. In particular, the working set growth due to the presence of multiple versions is 50% and 100% for the light and heavy workloads, respectively.

The size of the caches depends on the workload model that is, we use caches of different size in such a way that for both light and heavy workloads the global cache size (i.e., the sum of the sizes of the caches used in our experiments) is nearly 80% of the actual working set for the basic setup of our servers (in Section 6.3 we vary the size of the caches). Specifically, the global cache size is 3 GB for the light workload and 4 GB for the heavy workload.

The mean file size of the two workload models differs considerably; the main consequence is that the heavy workload causes cache hit rates lower than those of the light workload.

We have also introduced a popularity resource distribution for both workloads by defining a set of hot resources (corresponding to 1% of the working set): 10% of the total number of requests refers to this set.

From the file list of each workload, we obtained 80 different traces that were used in parallel during the experiments. The traces have been obtained as follows. For each workload, we have identified a set of hot resources (selected randomly from the workload file list) and a set of normal resources. Each trace contains 1000 requests. For each request, we have determined randomly whether it belongs to the hot set or not. A further random number is then used to select the specific resource from the proper set. Finally, each request is associated to a client device according to the percentage of diffusion reported in Table 1.

In each trace a random delay elapses between two consecutive requests. The client request rate can be controlled by adjusting the mean value of the delay. Figure 8 shows the results of a preliminary test conducted on a single machine to analyze how the 90-percentile of response time is affected by the client request rate for both workloads. As the request rate grows, the response time increases more than linearly due to the higher CPU load caused by transcoding operations. In our experiments, the default client request rate has been set to 5 requests per second, being this value consistent with our analysis of real-world IRCache proxy logs. The results shown in the figure also evidence the transcoding load imposed on the CPU and motivate the need of caching already adapted resources and distributing the content adaptation on multiple nodes.

6. Experimental results

The experimental results described in this section have a twofold goal. First, we aim to evaluate the effectiveness of the transcoding process in reducing the resource sizes and, consequently, the transmission times on the link between the client and its assigned intermediary node. We then focus on cooperative resource discovery, analyzing the performance of different cooperative architectures of intermediary nodes organized in hierarchical and flat distributed topologies, and comparing them to the case of a

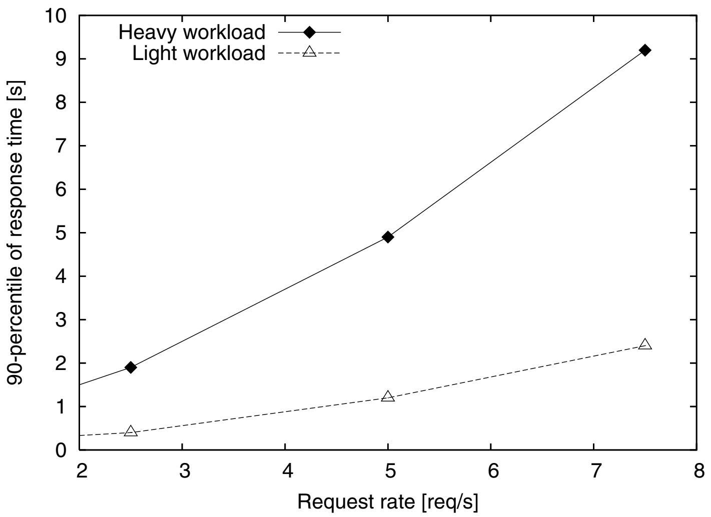

Figure 8. 90-percentile of response time as a function of client request rate for the light and heavy workloads.

non-cooperative system. In this second set of experiments we use both light and heavy workloads to provide a more detailed analysis.

In our experiments, we set up a system of 16 intermediary nodes. Each machine has a P-III 933 MHz CPU with 256 MB RAM and runs Linux. The servers are equipped with the prototype described in this paper and configured to cooperate through different architectures and resource discovery protocols. An additional node acts as the server which hosts the original Web content.

6.1. *Performance benefits of transcoding*

Figures 9 and 10 show the size reduction obtained by transcoding JPEG and GIF images, respectively, in the light workload. As performance index we use the cumulative distribution of the *stretching factor*, which is defined as the ratio of the size of a transcoded image over the size of its original version.

We can see that in some cases the adaptation is not productive in term of size reduction. The reason is that the effect of the applied transcoding algorithm depends on the peculiar characteristics of the images. For example, GIF format uses a compression algorithm to reduce the number of bits required to store frequent color map values; an adaptation that reduces the spatial geometry could consequently increase the number of unique colors with low occurrence frequency, with the side effect of increasing also the output image size. Hence, we observe that for the GIF format almost 20% of images adapted for HPC is larger than the original ones, but for other transformations we obtain a considerable size reduction. As regards JPEG images adapted for TVBrowser devices, we have obtained a stretching factor value lower than 1 for 85% of images. For JPEG images adapted for other devices this percentage reaches 95%, as shown in Figure 9. The adaptation for cellular phones reduces all the JPEG images at least of 50%: for this

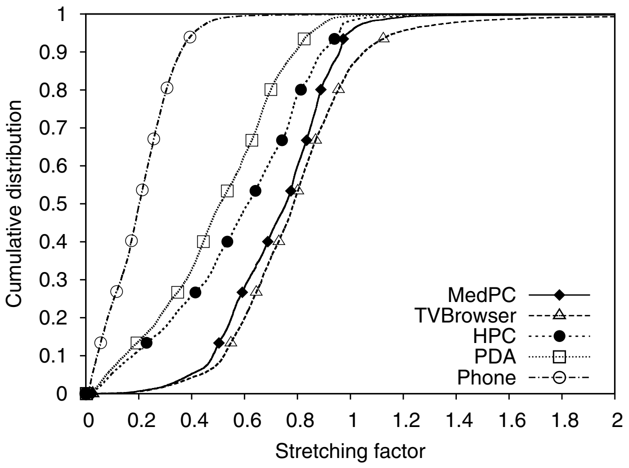

Figure 9. Size reduction for JPEG images.

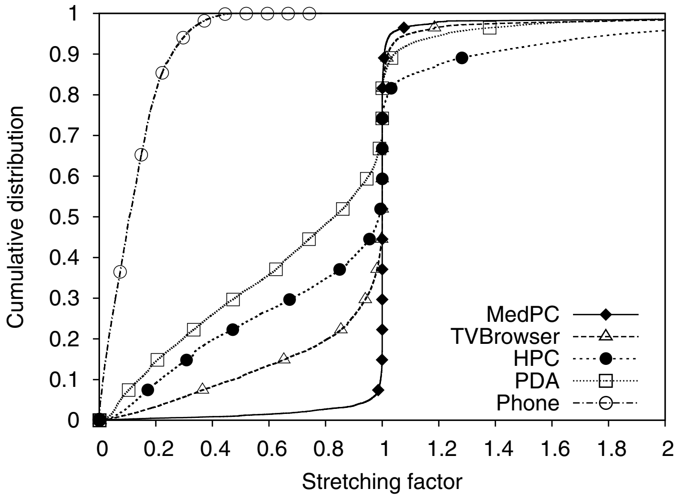

Figure 10. Size reduction for GIF images.

class of devices, we convert JPEG images to the GIF format because the latter is more suitable for black and white images. In this case, it is necessary to change dynamically the MIME subtype within the HTTP reply header. For MedPC the size of GIF images after adaptation is reduced only by a little percentage (5%), while better results are achieved for the other classes of devices (40% for TVBrowser, 50% for HPC, 70% for PDA and over 90% for Phone), as shown in Figure 10.

The size reduction obtained by performing adaptation reduces the transmission time of the adapted resources on the link between the intermediary node and the client device. Figures 11 and 12 report for the two image formats the median value of transfer and

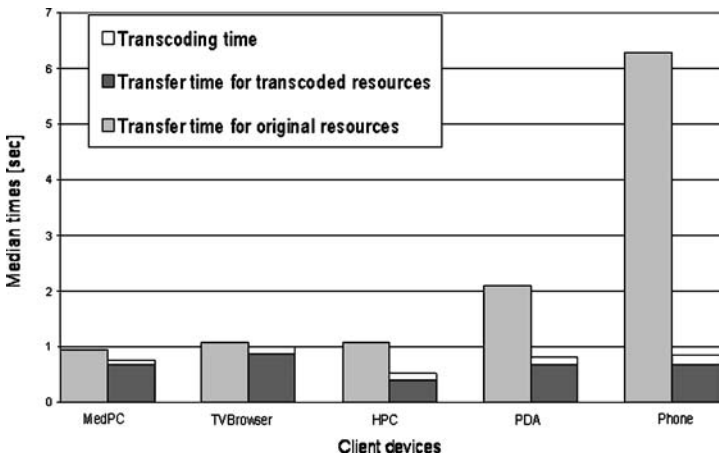

Figure 11. Median of transfer and transcoding times for JPEG images.

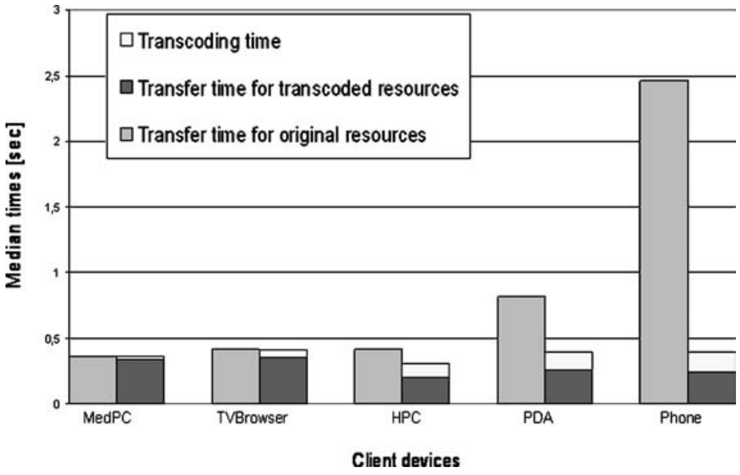

Figure 12. Median of transfer and transcoding times for GIF images.

transcoding times for the different classes of devices requiring adaptation (the transfer time values for HighPC devices, which consume only original resources, are equal to 0.002 and 0.005 sec. for GIF and JPEG formats, respectively.)

It is interesting to note that the median value of transfer times for the transcoded resources is comparable for all the client classes, while the corresponding times for original resources show an increase of one order of magnitude for clients with slow connections. As shown in Figure 12, in case of GIF images the transmission time of transcoded resources slowly connected devices as Phone is even lower than that of better connected devices as MedPC.

On the other hand, the content adaptation costs vary according to the resource type and the transcoding algorithm, but they are generally high as confirmed by measurements on machines used for our experiments (see Figure 7). In Figures 11 and 12 we also report the median value of transcoding times for each class of client devices and the two image formats.

We observe that for each device and image format the sum of transcoding and transfer times for transcoded images (the column on the right) never exceeds the corresponding transfer times of original resources (the column on the left). Hence, the size reduction of transcoded resources can counterbalance the time spent performing adaptation. For MedPC and TVBrowser devices, the two measurements are comparable because of the lower size reduction obtained by adaptation, especially for the GIF format. On the other hand, for the devices with slow network connections the gain achieved by performing adaptation is considerable: Figures 11 and 12 show that summing up the transcoding time to the transfer time, the results for these devices are of one order of magnitude lower than the corresponding transfer times of original resources.

It is worthwhile to note that the transcoding time values shown in these figures represent a worst case, because transcoding costs do not necessarily affect each client request; indeed, the resource adaptation is not performed when the desired version is found in one of the caches belonging to the distributed architecture (i.e., in case of exact hit). Hence, cache hit rate is a fundamental parameter to improve the user-perceived response time as discussed in the following set of experiments.

6.2. *Performance benefits of cooperation*

In this section, we compare the performance of the hierarchical and flat architectures for collaborative content adaptation of Web resources and quantify to what extent cooperation among nodes improves the system performance with respect to the case of no cooperation.

We set up a scenario where the intermediary nodes are well connected among them and with the clients. The content server is 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 2 Mbps. We verified that in this scenario the network path to the content server (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. In the experiments described in this section, we set the global cache size so that it is nearly 80% of the actual working set. Even if storage costs decrease day by day, we can consider the finite cache size assumption as realistic, because the global size of the actual working set largely exceeds the original one.

We consider a **Hierarchical** architecture, a flat architecture using a query-based discovery protocol (**Flat-query**), and a flat architecture adopting a summary-based protocol (**Flat-summary**). The configuration for Flat-query and Flat-summary schemes is based on a flat cooperative architecture, where all intermediary nodes have sibling relationships among them. The 80 traces used for the experiments are fed to the system so that each of the 16 nodes running our prototype receives 5 traces. The hierarchical architecture is configured on the basis of a three-level hierarchy with 12 leaf nodes, 3 intermediate nodes (with a nodal out-degree of 4), and one root node. The client requests reach only the leaf nodes. In this case, the original 80 traces are redistributed over the 12 leaf nodes, so that 8 nodes receive 7 traces each while the remaining 4 leaf nodes receive only 6 traces. For performance comparison, we consider also the non-cooperative scheme (**No_Coop**), in which the intermediary nodes do not cooperate. In this scheme, exact and useful hits can only be local, while local misses cause a request of the original resource to the content server. In the following experiments we use both light and heavy 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 index for the users.

Tables 2 and 3 show the cache hit rates for light and heavy workloads, respectively. The first four columns report the local and remote hit rates (exact and useful), while the last column summarizes the global hit rate, given by the sum of the various hit rates.

From the last column of Tables 2 and 3, we can observe that there are some significant differences in the global hit rates, depending on the used cooperation mechanism. For both workloads, flat architectures offer the best results. In particular the Flat-query scheme achieves higher values for remote hit rates, while Flat-summary turns out to be less effective in finding hits. Indeed, when the cooperation occurs among many nodes, Flat-summary 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 workload (but it can be also observed for the light one): the presence of larger resources causes faster changes in the cache contents, thus reducing the accuracy of the exchanged digests. Columns 4 and 5 in Table 2 show that the reduction in the global hit rate is caused by a reduction of the remote hit rate. On the other hand, the Flat-query architecture clearly presents a higher overhead than Flat-summary in terms of messages exchanged over the network, as pointed out in [10]. In our experiments the time for ICP queries lies in the range from 10 to 50 ms. By comparing the time for the query-based lookup to that required by adaptation, we see that queries are much cheaper than transcoding. Taking into account the computational cost of transcoding, we can conclude that, for the distributed content adaptation architecture, the most important objective is to reduce

Table 2. Cache hit rates (*light workload*).

	Local exact HR (%)	Local useful HR (%)	Remote exact HR (%)	Remote useful HR (%)	Global HR (%)
No_Coop	18.4	7.6	n/a	n/a	26.0
Hierarchical	10.2	8.2	10.4	9.2	38.0
Flat-summary	21.2	11.9	11.5	11.5	56.1
Flat-query	19.4	16.9	13.8	19.3	69.4

Table 3 Cache hit rates (*heavy workload*).

	Local exact HR (%)	Local useful HR (%)	Remote exact HR (%)	Remote useful HR (%)	Global HR (%)
No_Coop	5.1	4.3	n/a	n/a	9.4
Hierarchical	3.6	2.8	4.1	3.1	13.6
Flat-summary	5.2	4.5	10.7	9.3	29.7
Flat-query	5.1	4.7	20.1	22.1	52.0

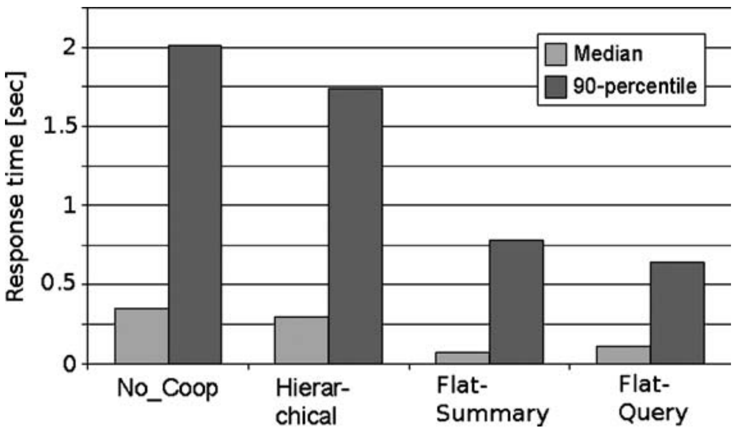

Figure 13. Median and 90-percentile of response time (*light workload*).

the load at the nodes. Therefore, the Flat-query scheme turns out to be preferable as it considerably increases the cache hit rates. The Hierarchical architecture shows hit rates lower than those of other cooperative architectures. The motivation is that, for every client request resulting in a local miss, the original resource is stored at each level of the hierarchy. This leads to an inefficient use of cache space that, due to the limitation of the cache size, notably reduces hit rates.

The strong relationship between hit rate and response time is confirmed by the following set of experiments. Figures 13 and 14 shows an histogram of median and 90-percentile values of response time for the various cooperation schemes for light and heavy workloads, respectively. If we consider the 90-percentile, we can see clearly the relationship between the hit rate and the response time: higher hit rates lead to lower response times. However, it is also interesting to note the slightly different values of the median. In particular, even if the global hit rate of the Flat-summary scheme is lower than that of Flat-query, nevertheless the median response time of Flat-summary is lower than that of Flat-query. This is a consequence of the faster lookup mechanism of Flat-summary that can carry out a remote lookup using only already available information on the local node. Therefore, remote hits are typically serviced faster than those of the Flat-query scheme.

The Hierarchical scheme is penalized with respect to the flat architectures. There are two reasons for this result. In the Hierarchical scheme, the upper level nodes can only act as pure cache servers (in our testbed prototype, 4 nodes over 16 do not perform transcoding operations); hence, the overall computational power available for adaptation is reduced. Moreover, as shown in Tables 2 and 3, the flat cooperation schemes achieve the highest cache hit rate that contributes to improve the response time.

6.3. *Sensitivity to cache size*

As a further analysis of the proposed cooperative content adaptation schemes, we now present a sensitivity analysis to the cache size aiming to demonstrate the best performance of the Flat-query scheme over a wide range of values for this parameter. In

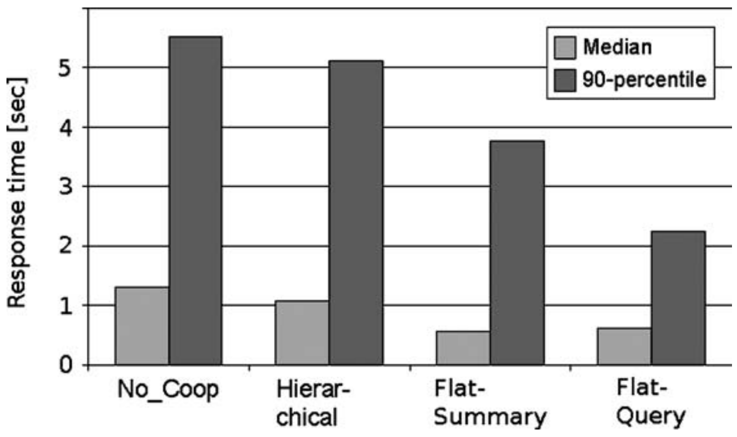

Figure 14. Median and 90-percentile of response time (*heavy workload*).

this set of experiments we focus only on the flat architectures, because those schemes have outperformed the hierarchical one in the previous architectural comparison.

Tables 4 and 5 show the global hit rate of the two flat architectures. We report only the global cache hit rate because the disaggregation of hits in local, remote, useful and exact is similar to the one shown in Tables 2 and 3. The cache size refers to the globally available cache storage (i.e., the sum of the cache sizes of each intermediary node) and is normalized against the actual working set size. As expected, increasing the amount of storage available for caching has a positive effect on the hit rate, which grows monotonically as the cache size increases. The experiments have been carried out also for values of global cache size greater than 100%. The achieved hit rate for a cache size of 106% is far lower than 100%: this indicates that both cooperation schemes present a fair amount of cache content duplication.

The most significant result arises from the comparison of the two flat cooperation schemes: from the values shown in Tables 4 and 5 it is evident that, for both workloads and for each tested cache size, the Flat-query scheme achieves a higher hit rate than Flat-summary.

Figures 15 and 16 show the 90-percentile of the response time as a function of the cache size. The relationship between the hit rates in Tables 4 and 5 and the results shown by the figures is clear: the better cache hit rate of the Flat-query scheme determines a lower 90-percentile of response time with respect to Flat-summary.

Table 4 Global cache hit rates as a function of cache size (*light workload*).

Cache size (%)	Flat-summary (%)	Flat-query (%)
40	36.8	42.6
80	56.1	69.4
106	80.4	82.1
132	98.2	98.3

Table 5 Global cache hit rates as a function of cache size (*heavy workload*).

Cache size (%)	Flat-summary (%)	Flat-query (%)
40	22.1	40.2
80	29.7	52.0
106	34.3	66.9
132	53.6	78.4

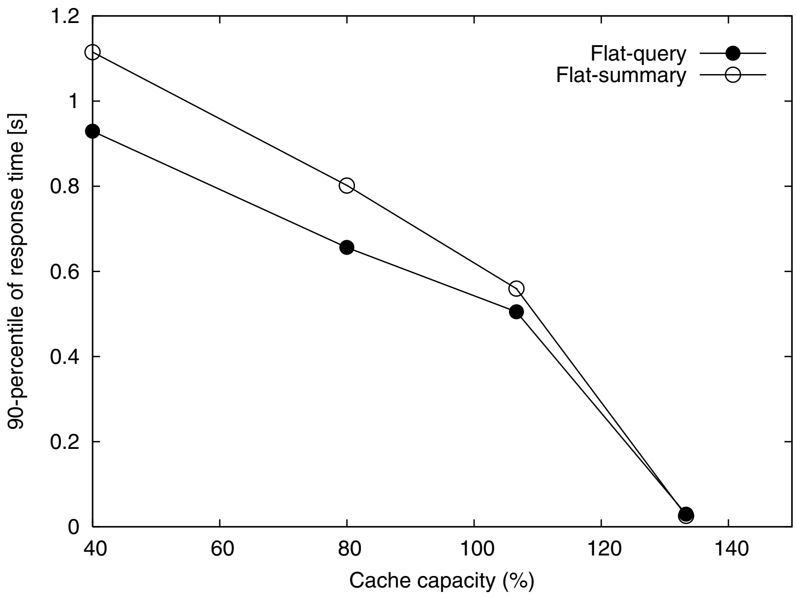

Figure 15. 90-percentile of response time for different cache sizes (*light workload*).

To summarize the results that can be got from all the experiments, cooperation improves the response time for both workload models with respect to the non-cooperative case. Among the cooperation schemes, the Hierarchical one is the slowest; Flat-query turns out to be the best performing scheme to serve the large majority of the requests, even if Flat-Summary can be faster than Flat-query to serve half of the requests for both workloads. Furthermore, a sensitivity analysis of response time and cache hit rate to the cache size confirms that the Flat-query scheme achieves a better performance than Flat-summary for cache sizes ranging from 40% to more than 130% of the actual working set.

7. Related work

While the investigation of the intermediary-based approach, in which the adaptation and caching functionalities are provided by stand-alone nodes that do not cooperate

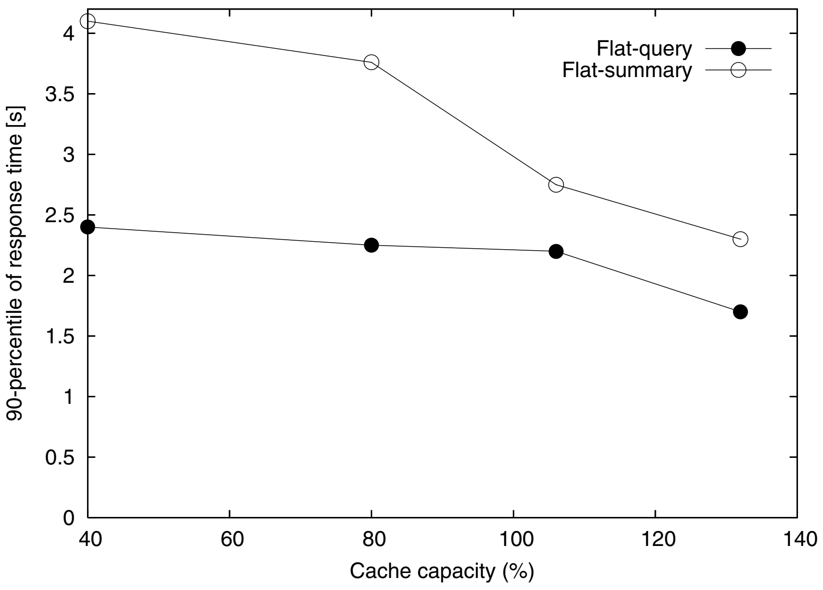

Figure 16. 90-percentile of response time for different cache sizes (*heavy workload*).

among them (or at most by cluster of nodes [12]), has attracted the attention of many researchers, only recently cooperative intermediary architectures for content adaptation have been addressed [3,4,27] (although the study in [27] regards peer-to-peer networks and is more focused on content personalization). Therefore, in this section we mainly discuss proposals related to non-cooperative intermediary nodes.

Most research efforts have focused on handling the variations in client bandwidth and display capabilities [5,12,13,15,33]. In these proposals, the node that directly connects to the clients typically reduces the resource size, thus reducing bandwidth consumption, apart from providing a resource version that the client device can consume. Although some of the above mentioned works investigate resource compression techniques [13,33], none considers caching.

A limited number of recent proposals have exploited techniques to combine both content adaptation and caching to reduce the computational resource usage at the intermediary node [7,26,28,30]. In [28] the authors have examined how to improve the effectiveness of transcoding by integrating it with caching mechanisms and taking into account the transcoding utility of any cached resource to decide about transcoding. The transcoding operations regard images and are carried out using the ImageMagick library, as in our proposal. While the architecture discussed in [28] is based on the Java-based proxy Rabbit, we propose to enhance with transcoding and caching functionalities the Squid proxy server, which is characterized by robustness and wide spread usage. Moreover, we introduce a new challenge by enhancing some cooperation mechanisms provided by Squid to provide multi-version resource discovery. A Squid-based transcoding proxy has been first presented in [20], which also discusses how to match client requests to cached transcoded resources. Again, this work considers only a stand-alone intermediary node.

More sophisticated cache replacement policies designed for transcoding intermediary nodes have been proposed in [7,26,30]. The transcoding cost is taken into account in the caching policies investigated in [7]. While the works in [7,20,28] focus primarily on the transcoding of static Web resources (as also this paper), [30] and [26] combine caching with transcoding of streaming media resources. Comparison of sophisticated cache replacement policies for multi-version content is out of the goals of this paper. Nevertheless, our Squid-based prototype provides an extensible framework that allows to integrate new caching policies and evaluate their performance in a real Web environment. Indeed, most of the above mentioned works provide a performance evaluation of the proposed policies through simulation.

The main motivation that leaded us to study cooperative intermediary-based architectures for content adaptation is the limited scalability to which the existing intermediary-based approach may be subject. The main issues arise from the computational cost of transcoding operations, especially for large multimedia resources [20,26], and the network connection between the client and the intermediary node. The scalability issue has been addressed in [12] through a cluster of locally distributed proxies. A cluster-based approach may solve the CPU-resource constraint of the computational nodes, but it tends to move the system bottleneck from the node CPU to the cluster connection to Internet. On the other hand, in our proposal the cooperative intermediary nodes can be distributed over a wide area network. With respect to a cluster-based architecture, a WAN solution allows to use network resources more fairly and to reduce response time variance due to network-related delays, as demonstrated in [9] for traditional cooperative Web caching.

In [4] the authors have obtained some preliminary results through simulation that demonstrated the importance of distributing the computational load of transcoding in a hierarchical architecture of cooperative intermediary nodes. In [3] the authors have focused on the performance comparison of hierarchical cooperative architectures and transcoding algorithms for flat cooperation schemes.

8. Conclusions

In this paper we proposed a Squid-based prototype that provides on-the-fly image transcoding services on the basis of the client device capabilities and operating on various image parameters, such as spatial geometry, color depth, quality factor, and MIME subtype. We modified most of the traditional phases of the standard Squid proxy server in order to support adaptation and multi-version caching functionalities on the intermediary node. We used an external transcoder process to maximize flexibility and make the prototype independent of the particular tool used to adapt the resources. We also investigated how to enable cooperation for intermediary nodes organized in hierarchical and flat distributed topologies. Our results demonstrated that the flat cooperation among the nodes clearly improves the response time with respect to the non-cooperative case for different workload scenarios. In particular, we found that the Flat-query cooperation scheme achieves the best performance even for different cache sizes.

Integrating content adaptation services within cooperative intermediary infrastructures raises many research issues. Some interesting topics that deserve further investigation include the use in a cooperative environment of different transformations regarding content types such as video and audio data, the integration of content server and intermediary node operations, as well as the analysis of cache replacement policies designed for multi-version content in a cooperative distributed architecture.

Acknowledgements

The authors would like to thank the referees for their valuable comments and suggestions which were helpful in preparing the final version of the paper. The authors acknowledge the support of the Italian MIUR in the framework of the FIRB project "Wide-scale, Broadband, Middleware for Network Distributed Services" (WEB-MINDS).

References

[1] Apache, Apache HTTP Server, 2005. http://httpd.apache.org/

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

[3] C. Canali, V. Cardellini, M. Colajanni, R. Lancellotti, and P. S. Yu, "Cooperative architectures and algorithms for discovery and transcoding of multi-version content," in *Proc. of 8th Int'l Workshop on Web Content Caching and Distribution*, 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*, Nov. 2000, pp. 520–527.

[5] C. S. Chandra, S. Ellis, and A. Vahdat, "Application-level differentiated multimedia Web services using quality aware transcoding," *IEEE J. on Selected Areas in Communication*, 18(12), 2000, 2544–2465.

[6] 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.

[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), 2003, 611–624.

[8] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel, "A hierarchical Internet object cache," in *Proc. of USENIX Ann. Tech. Conf, Jan. 1996*, pp. 153–163.

[9] S. G. Dykes and K. A. Robbins, "A viability analysis of cooperative proxy caching," in *Proc. of IEEE Infocom 2001*, April 2001.

[10] 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), 2000, 281–293.

[11] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol — HTTP/1.1*. RFC 2616, June 1999.

[12] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-based scalable network services," in *Proc. of 16th ACM SOSP*, Oct. 1997, pp. 78–91.

[13] 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 Comm.*, 5(6), 1998, 8–17.

[14] IBM. IBM WebSphere Transcoding Publisher, 2005. http://www.ibm.com/software/pervasive/transcoding_publisher/

[15] S. Ihde, P. P. Maglio, J. Meyer, and R. Barrett, "Intermediary-based transcoding framework," *IBM System Journal*, 40(1), 2001, 179–192.

[16] ImageMagick, 2005. http://www.imagemagick.org/

[17] IRCache project, 2005. http://www.ircache.net

[18] B. Knutsson, H. Lu, and J. Mogul, "Architecture and performance of server-directed transcoding," *ACM Trans. on Internet Technology*, 3(4), 2003, 392–424.

[19] 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*, September 2002, pp. 239–250.

[20] 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 Res. Issues in Data Eng.*, Feb. 2002, pp. 50–59.

[21] Mercury. Mercury LoadRunner, 2005. http://www.mercury.com/us/products/performance-center/loadrunner/

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

[23] A. Pashtan, S. Kollipara, and M. Pearce, "Adapting content for wireless Web services," *IEEE Internet Computing*, 7(5), 2003, 79–85.

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

[25] A. Rousskov and D. Wessels, "Cache digests," *Computer Networks*, 30(22/23), 1998, 2155–2168.

[26] B. Shen, S.-J. Lee, and S. Basu, "Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks," *IEEE Trans. on Multimedia*, 6(2), 2004, 375–386.

[27] W. Shi, K. Shah, Y. Mao, and V. Chaudhary, "Tuxedo: A peer-to-peer caching system," in *Proc. of 2003 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications*, June 2003.

[28] A. Singh, A. Trivedi, K. Ramamritham, and P. Shenoy, "PTC: Proxies that transcode and cache in heterogeneous Web client environments," *World Wide Web*, 7(1), 2004, 7–28.

[29] Squid Internet Object Cache, 2005. http://www.squid-cache.org

[30] X. Tang, F. Zhang, and S. T. Chanson, "Streaming media caching algorithms for transcoding proxies," in *Proc. of Int'l Conf. on Parallel Processing*, Aug. 2002, pp. 287–295.

[31] D. Wessels, *Squid Programmers Guide*, 2004. http://www.squid-cache.org/Doc/Prog-Guide/.

[32] D. Wessels and K. Claffy, *Internet Cache Protocol (ICP), version 2*. RFC 2186, Sept. 1997.

[33] B. Zenel, "A general purpose proxy filtering mechanism applied to the mobile environment," *Wireless Networks*, 5(5), 1999, 391–409.