Collaborative Proxy System for Distributed Web Content Transcoding

Valeria Cardellini
University of Rome
"Tor Vergata"
Roma, I-00133
cardellini@ing.uniroma2.it

Philip S. Yu
IBM T.J. Watson
Research Center
Yorktown Heights, NY 10598
psyu@us.ibm.com

Yun-Wu Huang
IBM T.J. Watson
Research Center
Yorktown Heights, NY 10598
ywh@us.ibm.com

ABSTRACT

Content transformation (or *transcoding*) proxies have been recently proposed to tailor Web content to device characteristics of Web clients. In this paper, we address the problem of distributing the computational load caused by object transcoding throughout a collaborative proxy system organized in a hierarchical network. We evaluate through simulation the impact of load distribution and caching policies on users' response time. We find that the simple global policy that captures the proxy load information along the request path can provide reasonably good load sharing, and that, to effectively share the load, it is necessary to provide the edge proxies a mechanism to push up some transcoding load. On the caching policy, we examine policies that allow different versions of an object to be cached. Our study shows that the *demand based* caching policy which has the transcoding proxy cache the transcoded version performs better than the *coverage based* caching policy that caches the more detailed version and the *anticipatory* caching policy that caches both of these versions.

1. INTRODUCTION

Many emerging network appliances, such as hand-held PC, personal digital assistants, smart cellular phones, and other pervasive computing devices, are increasingly gaining accesses to the Internet, either by wired or wireless connections. These new diverse platforms, that will constitute a predominant fraction of Internet clients in a few years, compose with today's PCs a highly heterogeneous client environment. They differ in network connection bandwidth, processing power, storage, display and format handling capabilities. Additionally, more and more Web content is becoming of a multimedia nature. Therefore, the ever increasing requirements of clients and the diversity in page composition demand techniques to adapt the same content to diverse devices.

The process of converting a multimedia object from one form to another is called *transcoding* and can apply to transformation either within media types (e.g., from JPEG to GIF format), between media types (e.g., speech to text or video item to image set) or to both of them. The development of XML and XSL, that allow the same document to be presented on a variety of Web devices using structured documents with style sheets, will facilitate the presentation of multiple versions of a Web site. Since separate style sheets can be developed for diverse client devices, XML and XSL adoption will simplify the transformation process. However, it will not eliminate the need for content transcoding.

Web content adaptation may be deployed either at the client, or at the server, or at an intermediate proxy, or even at any combination of the three. Client's computing power and connection bandwidth are continuously improving. However, the limited capabilities and low bandwidth of network appliances make content adaptation at the client time consuming, if not impossible at all. In both server-based and proxy-based approaches, specific information can be associated with the client request, such as device and network constraints, or user preferences. The server-based approach [10] is designed to add content adaptation by extending the functionalities of a traditional Web server. The content transformation is usually generated off-line; the different transcoded versions of the same object are stored in the server disk and selected to match the client specification [10].

In the proxy-based approach [5, 7, 8, 9, 14], a Web proxy, located in the network between the client device and the content server, can also analyze and transcode the requested object on-the-fly, before delivering the result to the client. The proxy server can cache the result of the transcoding [8]. The dynamic content adaptation performed by the proxy is transparent to content users and providers. The proxy-based approach can be used to address dynamic variations in network traffic and to reduce end-to-end response time, especially for mobile devices. A transcoding Web proxy can reduce the size of a Web object while maintaining most of its semantic value [5, 7, 9]. Being the proxy generally placed at the border between clients and the rest of the network, object compression significantly improves response time for weakly-connected clients. Proxy transcoding also eliminates the need of content providers to rewrite and maintain multiple versions for different device types. A single transcoding

service located at the network level can also tailor contents coming from different Web servers.

If object transformation is entirely done on the edge proxy that directly connects to the clients (or a cluster of proxies [8]), the proxy may become overloaded, as transcoding is computation intensive [5, 9]. In this paper, we analyze how a network of proxy servers can work collaboratively in content transcoding and caching to improve Web response time. Our main goal is to study techniques that can distribute the computational load caused by transcoding throughout a collaborative proxy system organized in a hierarchical network. In this way, the object transcoding task is not concentrated in any specific proxy but distributed in an adaptive fashion. The selection of which proxy will perform the content adaptation is done in a fully distributed way. Additionally, we explore priority-based request scheduling to guarantee quality-of-service (QoS) to requests that do not require transcoding. A secondary aim of this paper is to analyze proxy caching techniques in such a new collaborative environment. In particular, we explore which object version is more valuable to cache. To analyze the proxy system, we develop a detailed simulation model that include most recent results regarding Web workload characterization. Our results show that distributing the transcoding task can considerably reduce the overall retrieval latency, especially when priority request scheduling is applied. We also find that the more valuable objects to cache are transcoded ones. The software requirements of the proposed collaborative proxy system are fully compatible with existing Web protocols and standards.

The paper is organized as follows. Section 2 provides an overview of the system environment. Section 3 explores how to share the load generated by transcoding requests in the collaborative proxy system. In Section 4 we describe our model for analyzing the proxy system. Section 5 presents the simulation results. Last, Section 6 concludes the paper.

2. SYSTEM ENVIRONMENT

In a highly heterogeneous client environment, users are connected to the Internet through a wide variety of platforms with different resource constraints. Each client device connects to the Internet using an assigned proxy. The collaborative proxy system is composed of several homogeneous proxy servers organized in a hierarchical network.

2.1 Client devices

Network appliances vary widely in their features such as screen size and color, processing power, storage, user interface and software. Client's access links to Internet also range from wired networks such as LAN, DSL, ISDN, and telephone modems, to wireless networks such as cellular, CDPD, and GSM. The client can include the object data type it can consume as a meta-information in the HTTP request header. The proxy that processes the request cannot send an object that requires more capabilities than the client has, but it has to fit the client needs as much as possible.

Since we consider caching of transcoded objects, multiple versions of the same object can be in the cache at the same time. An object which was already transcoded may be further transcoded to yield a less detailed (or lower-resolution) object. In particular, each version may be transcoded from a subset of the higher-resolution versions. To represent different versions of the same object and the allowed transcoding operations between them, we define a *transcoding relation graph* $G(V, E)$ for each object. The set of nodes $V = \{V_1, \ldots, V_n\}$ represents different versions of the object; the nodes are ordered according to the version quality they represent, i.e., V_i is a more detailed version of V_j if $i < j$. V_1 and V_n are the original and the least detailed version, respectively. A direct edge from V_i to V_j (where $i < j$) exists if V_j can be directly transcoded from V_i. In the latter case, V_i is called a *transcodable* version for V_j.

If each less detailed version can be transformed from each more detailed version, the corresponding graph (called *complete* transcoding relation graph) resembles the one shown in Figure 1a. For example, JPEG images with higher quality factor can be subsequently transcoded to obtained lower quality versions. When too much information has already been lost in previous transcoding operations, the proxy needs the original version of the object to produce the new lower-resolution one. This situation is shown in Figure 1b, where the less detailed versions V_3 and V_4 cannot be transformed from V_2. The following example illustrates the latter instance. A JPEG image V_1 and a color bitmap V_2 obtained from V_1 are cached; a lower resolution color bitmap V_3 has to be produced. If V_3 is not a trivial divisor of the number of pixels in V_2, the proxy may spend a lot of resources interpolating V_2 to yield a decent V_3, without getting any good result. Using V_1, the proxy can directly produce a good V_3.

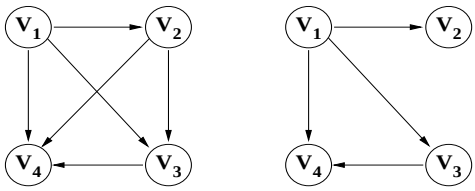

Figure 1: Transcoding relation graph: (a) complete graph, (b) incomplete graph.

2.2 Collaborative proxy system

The *hierarchical caching* architecture [6, 12, 15] assumes a hierarchy of K levels of caches, where the bottom-level proxies, named *edge* proxies, serve client requests directly. In such a cooperative system, a client request is forwarded up the hierarchy until a cache hit occurs; if none occurs at any level, the requested object is retrieved from the content server by the root proxy. When the object is found, either at a cache or at the content server, it then travels down the hierarchy and a copy is stored in all caches along the request path. The top-level cache in the hierarchical architecture is a potential bottleneck. However, this scheme is not inferior to others, especially in wide-area environments [15]. In the hierarchical architecture we consider, the information flow is only between different levels of caches, i.e., sibling caches do not cooperate. Figure 2 illustrates the hierarchical caching scheme and the network information flow.

The hierarchical organization we propose is an extension of the traditional one, in that multiple versions of the same object can be present at the same time in the caches. Furthermore, the proxy nodes not only store static objects, but also

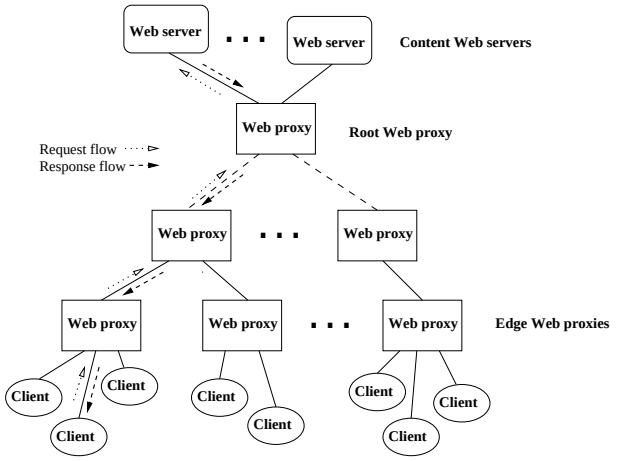

Figure 2: Hierarchical caching architecture.

perform the computation intensive transcoding task, unlike previously studied cooperative caching schemes [6, 15].

The proxy that receives a request, upon serving an object, first has to determine the capabilities of the client device. The accomplishment of this task depends on the role that the proxy plays in the hierarchy network. An edge proxy can obtain the information regarding client capabilities by accessing a table wherein the characteristics of the various client devices are stored. The table entry for a particular client device can be stored when the device first registers with the edge proxy. Then, this entry can be transmitted by the edge proxy to the upper-level proxy by including it in the client request. Therefore, if the proxy is located at an intermediate level, the information regarding client capabilities is provided by a lower-level proxy with the object request. Hereafter, we will refer to the information describing the capabilities of the requesting client as *requester-specific capability information* (RCI).

Once the proxy has determined the client capabilities, it has to look for a copy of the requested object in its cache. As requests originated from diverse client devices can reach the same proxy, one of the following events can occur: (1) the cache contains a less detailed or untranscodable version of the requested object; (2) the cache contains the exact version of the requested object; (3) the cache contains a more detailed and transcodable version of the requested object; (4) the cache does not contain any version of the requested object. As a more detailed and transcodable version can be transformed to obtain a less detailed version that meets the client request, it is referred to as a *useful version*. A less detailed or untranscodable version does not satisfy the request and leads to a traditional *cache miss*. The same happens if no version is found in the cache. In both cases, the object request is forwarded up through the caching hierarchy, until a useful version of the object is found. If the object is not retrieved along this path, the request reaches the proper content server. However, even if object transcoding is needed, the content server does not perform it, but returns the original object version to the root proxy. If the proxy owns the exact version of the requested object, its copy can be directly passed down the hierarchy, leaving a copy at each intermediate cache, until it reaches the client. We define this kind of hit as *exact cache hit*.

Since a more detailed and transcodable version available in the cache does not completely fulfill the client request but imposes some processing overhead on the system, we define this event as *useful cache hit*. Applying the selected load distribution policy, the proxy can decide either to perform the transcoding task locally based on the RCI associated with the request (thus returning to the lower-level proxy or the destination client the completely transcoded object), or not to perform the required transcoding and return its cached version to a lower-level proxy.

3. POLICIES

In this section we present some strategies that allow each proxy to decide whether to perform locally or not the task of transcoding a retrieved object. This decision is made when the proxy either finds a useful object version in its own cache or receives it from an upper-level proxy or from the content server. We also examine policies that allow different versions of an object to be cached.

3.1 Dynamic load distribution policies

The load distribution policies considered can be grouped on the basis of the amount of system information being used, assuming that all proxies hold software capable of performing the transcoding required by any type of client device:

NOINFO. The proxy uses no system information to decide about transcoding. The object is processed locally, if the proxy finds a useful version in its cache or obtains it from a content server. Otherwise, the proxy sends the client request to an upper-level proxy. So, the proxy located at the root of the proxy hierarchy is often heavily utilized, as frequent cache misses in lower levels cause the root proxy to get the original object from the content server and then transcode it.

THR (Threshold). This policy considers only local load information. The proxy decides to delegate to a lower-level proxy the transcoding task of a retrieved object if its utilization exceeds a given load threshold; otherwise, the proxy transcodes the object locally.

LL (Least Loaded). This policy represents the case of using global load information. Specifically, we consider a simple global policy that uses the load information along the request path. The proxy performs the object transcoding locally only if its load index is the lowest among all lower-level proxies on the request path. For example, the root proxy, after retrieving a useful version, compares its load index to those of the edge and intermediate proxies located on the request path and transcodes the object only if its index is the lowest. Otherwise, the root proxy returns the useful version to the closest intermediate proxy on the request path. The load index can be either the utilization evaluated over a short interval, or the instantaneous CPU queue length periodically observed. To provide the load information to upper-level proxies, each proxy inserts its load index in the RCI of any request that travels up

the hierarchy. Thus, there is no additional overhead needed for periodical exchange of load information in the network. Furthermore, the staleness of load information is highly reduced, as the load information is updated with every new request that reaches a proxy.

Both THR and LL strategies have a drawback. The decision about whether to transcode a useful object version locally or to pass the processing down to a lower-level proxy does not apply to edge proxies, as the edge proxy performs the transcoding if it finds a useful version in its cache. Furthermore, since the object that needs transcoding flows down the hierarchy, the edge proxy is the last server that can adapt the object before it is delivered to the client. As a result, an edge proxy cannot effectively shift away the transcoding overhead. To alleviate the computational load caused by performing transcoding at the edge proxies, an alternative is to allow the edge proxy to delegate the transcoding of a useful version found in its cache to an upper-level proxy. Both the request and the version retrieved by the edge proxy are forced to travel up the hierarchy, if the proxy load index exceeds a given threshold. In this case, the edge proxy determines which upper-level proxy on the path has to perform the transcoding, selecting the proxy with the least load index. Load information regarding upper-level proxies can be piggybacked to edge proxies within the header of each response that flows down the hierarchy. When such a pushing feature is added to the load distribution policy, the suffix **push** is appended to the policy name.

When client requests have distinct computational requirements, it is worth investigating priority-based scheduling of incoming requests. In fact, should all requests be handled in a first-come first-served manner, those requests that need transcoding would occupy the server's resources for a long time and lengthen the response time of lighter requests. We identify two classes of priority based on the request type. A low priority is assigned to requests that have been selected for transcoding on the current proxy (and therefore impose a much higher computational load), while a high priority is assigned to all other requests. When the proxy resource is assigned using the priority discipline, the suffix **prior** is added to the policy name.

3.2 Caching policies

Due to storage capacity limitations, cache misses can occur. As the main goal of this paper is not to design cache replacement policies, we adopt the standard LRU. This is the most widely-used Web cache replacement policy, even if it is not the best performing one [1]. Other policies will be investigated in future work.

Our interest is focused on evaluating which version of the object is more useful to cache. To avoid repeating transcoding operations at the same proxy, every transcoded object can be cached. We analyze three alternative selections about which object version is more valuable to cache, after the proxy has performed the transcoding task. Under the first policy, referred to as the *demand based* caching policy, the proxy caches the object version resulting from transcoding. It aims at caching the version that is being *demanded* by the clients, or at least the version close to it, if the demanded version is not available. Under the second policy, referred to as the *coverage based* caching policy, the proxy caches the object version on which the transcoding process has just been applied. This is the *retrieved version* which might have been retrieved from the local cache of the transcoding proxy or received from an upper-level cache. The goal is to provide maximum coverage on subsequent requests so as to cover various versions that are transcodable from the cached version. Both choices have advantages and drawbacks. Caching the transcoded version may avoid future processing overhead. However, this version, being less detailed than the original one, can be useful for requests originated from a smaller number of devices. On the other hand, the retrieved version, being a more detailed version of the object, may be useful to satisfy a greater number of devices; its caching, however, may make the proxy repeat some recent object transcoding operations. The third alternative, referred to as the *anticipatory* caching policy, is to cache both the transcoded and the retrieved versions. It anticipates the more detailed version that may also be requested in the future. In this paper, all objects were considered as cacheable and no expiration modeling was made; future work can address these aspects.

4. SIMULATION MODEL

A detailed simulation model has been developed to evaluate the system performance. The model is designed to reflect the focus of this paper, which is a collaborative proxy system accessed by a highly heterogeneous client environment. In this section, we explore the major Internet components that affect the performance of the proxy system.

4.1 Client model

In this paper, the client devices are classified according to their capabilities of displaying different objects and connecting to the assigned proxy. We have identified the following classes of devices:

- **HIGHPC**: a high-end workstation/PC whose network link ranges from Ethernet to ISDN; it can consume every object in its original form.
- **MIDPC**: a midrange PC or a laptop connected through a fast/medium wire-connected modem.
- **HPC**: a hand-held PC connected through a modem; it can display color images with different resolution.
- **PDA**: a personal digital assistant using a wireless CDPD connection; it is not capable of displaying colorful and large images.
- **SMARTPHONE**: a cellular smart phone using a wireless GSM connection; the HTML text is summarized, while images are converted to a brief textual description that can be scrolled on the small screen.

Table 1 shows the different capabilities for handling HTML and image objects and the bandwidth of the network link. As to images, we show the color space and size reduction needed. In this paper, we consider transcoding operations only on HTML and image objects, as more than 85% of

Table 1: Client device types.

Device	*HTML*	*Image color*	*Image scaling*	*Bandwidth*
HIGHPC	original	color (24 bit)	0%	128 Kbps - 10 Mbps
MIDPC	original	color (16 bit)	25%	28.8 Kbps - 56 Kbps
HPC	original	color (16, 8 bit)	50%	28.8 Kbps - 56 Kbps
PDA	summarization	gray, b&w (4, 2 bit)	75%	19.2 Kbps
SMARTPHONE	summarization	to text	to text	9.6 Kbps

the transferred Web objects belongs to these types [2]. The probability of accessing to the system for a given type of device is defined a *device vector*, $device[i]$ where $i \in \{HIGHPC, MIDPC, HPC, PDA, SMARTPHONE\}$.

4.2 Workload model

The model incorporates the most recent results on Web workload characterization. The high variability and self-similar nature of Web access load is modeled through heavy-tailed distributions such as Pareto, lognormal and Weibull distributions [2, 3, 4, 11, 13].

The number of consecutive Web pages a user requests from the Web system during a session follows the inverse Gaussian distribution [11]. The client's silent time between the retrieval of two successive Web pages (*user think time*) is modeled through a Pareto distribution [3, 11]. The self-similarity of Web traffic requests is explained with the superimpositions of heavy-tailed ON-OFF periods. The number of objects that make up a whole Web page, including the base HTML object and its in-line referred files, also follows a Pareto distribution [3]. The function that models the distribution of the *object size* requested to the proxy system varies according to the object type. For HTML object, it is obtained from a hybrid distribution, where the body follows a lognormal distribution, while the tail is given by a heavy-tailed Pareto distribution [2, 3]. The size distribution of in-line objects in a page is obtained from the lognormal distribution [3]. The relative frequency with which Web objects are requested follows a Zipf-like behavior [4]. Zipf-like distribution is largely believed to be a good estimate of real Web usage patterns. Table 2 provides a summary of the distributions and parameters value for the workload model.

Table 2: Workload model.

Category	*Distribution*	*Parameters*
Pages per session	Inverse Gaussian	$\mu = 3.86$, $\lambda = 9.46$
User think time	Pareto	$\alpha = 1.4$, $k = 1$
Objects per page	Pareto	$\alpha = 1.245$, $k = 2$
HTML object size	Lognormal Pareto	$\mu = 7.630$, $\sigma = 1.001$ $\alpha = 1$, $k = 10240$
In-line object size	Lognormal	$\mu = 8.215$, $\sigma = 1.46$
Object popularity	Zipf	$\alpha = 0.15$

4.3 Network and proxy system model

We model the underlying proxy network topology as a full O-ary tree, where O is the nodal out-degree. The proxies communicate using a network, whose link capacity is set to 45 Mbps and 34 Mbps at regional and national level, respectively. We also consider the delay experienced by the root proxy in retrieving the original object from the content server. The client-proxy bandwidth depends on the type of device, as described in Section 4.1.

The Web proxy system consists of N_P servers with homogeneous processing and storage capacity. The cache size is set to contain 85% in bytes of original objects. We also model the CPU overhead for processing HTTP messages. The system utilization varies in the range [0.45, 0.75] of the capacity of the entire proxy system. New client sessions are generated with exponential interarrival time [13]. When the load information is used in the decision making process, each proxy evaluates its own information every 8 seconds.

An important issue is how clients are assigned to edge proxies. We assume that clients are partitioned based on a Zipf-like distribution. In most experiments, the clients are partitioned among 9 edge proxies with parameter $\alpha = 0.35$, that corresponds to a skewed function. The sensitivity analysis in Section 5.3 shows that the main conclusions of the paper are not affected by the choice of parameters such as client distribution and proxy cache size. Table 3 summarizes some parameters used in the simulation experiments.

Table 3: Parameters of the system model.

Category	*Parameter*	*Value (default)*
Web proxy	Number of proxies (N_P)	13
	Levels of caches (K)	3
	Nodal out-degree (O)	3
	HTTP processing time	0.001 sec.
	Disk transfer rate	10 MB/sec.
	Transcoding rate	20 KB/sec.
	Bandwidth	34 Mbps - 45 Mbps
	Content server delay	Exp. ($\beta = 1$ sec.)
Client	Interarrival time	Exp. ($\beta = 0.5$ sec.)
	Edge proxy	Zipf ($\alpha = 0.35$)
	Device vector	[0.15, 0.1, 0.35, 0.25, 0.15]

5. EXPERIMENTAL RESULTS

The main objective of this study is to understand the impact on the user response time of distributing the transcoding task among the collaborative proxy system with the aim to minimize it. We use the *Mean Response Time* of requested objects as the main performance measure. It corresponds to the object retrieval latency elapsed between the submission of the client request and its completion at the client.

Transcoding requests may require a fair amount of processing. Compared to the processing time to access the exact object stored in the cache, the processing time required to adapt the original object obtained from the content server can be orders of magnitude larger [5, 9]. Therefore, we also use as an alternative metric the *Stretch Factor*, defined as the ratio of the response time of a sequence of requests over the service time of these requests at the different proxies they flow through. This metric relates the client waiting time to the service demand. User devices that require transcoding

may be willing to wait longer to complete their requests, while more powerful devices expect that their requests are completed quickly. Both Mean Response Time and Stretch Factor can be defined either for every client issuing requests to the proxy system, regardless of the device it represents, or for every class of device.

The simulator, based on the independent replication method, was implemented using the CSIM18 package. Each value is the result of ten simulation runs with different seeds. Confidence intervals were estimated for all simulation results, and the 90% confidence interval was estimated to be within 10% of the mean. In this paper, we assume that the proxy transcodes the Web object to best fit client characteristics, not exploring object compression to reduce transmission time [7, 9]. We also consider almost complete transcoding relation graphs, except for images destined to phones that can only be transcoded from the original version.

5.1 Performance of load distribution policies

In the set of experiments comparing load distribution policies, the proxy that performs the transcoding task adopts the demand based caching policy. As shown in Section 5.2, this is the best performing alternative. Figure 3 shows the system mean response time against the client interarrival time of the load distribution policies where all requests are scheduled in a first-come first-served (FCFS) manner. The LL policy outperforms the THR scheme, even when the best performing load threshold value is selected for the latter. Since the THR policy considers only a local load information, a proxy can delegate its transcoding task to a more critically loaded lower-level proxy. Therefore, edge proxies can be overwhelmed by transcoding tasks not accomplished by upper-level proxies. The results of the NOINFO policy are not shown, since this scheme causes the system to be unstable with a mean response time always above 250 seconds. Under this policy, all transcoding tasks end up at the root proxy, that does not have the capacity to handle the offered load. Figure 3 also shows the impact of the proxy load index on the performance. Two variations on the load index are considered: LLutil, based on the proxy CPU utilization, and LLqlen, based on the number of requests in the proxy queue. The result indicates that the queue length is the most useful load index.

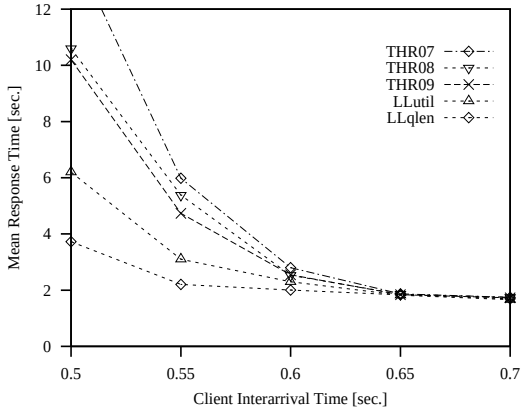

Figure 3: Mean response time of load distribution policies without priority.

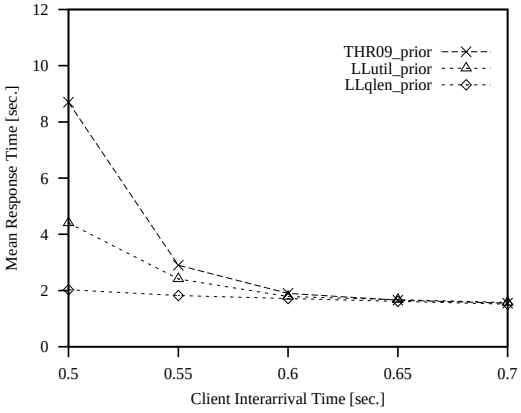

Figure 4: Mean response time of load distribution policies with priority.

In Figure 4, we evaluate the performance of different load distribution schemes when a priority-based scheduling discipline is adopted. Two classes of priority are considered: requests requiring transcoding on the current proxy are assigned a low priority, while all the other requests are given a high priority. As in the case of no priority shown in Figure 3, the LL policy outperforms the THR scheme and the load index based on the queue length allows to achieve better results. The mean response time of the NOINFO policy is always above 70 seconds, that is even with priority scheduling the system is still highly unstable. Nonetheless, we can note that the priority discipline improves the performance.

In the following experiments, the client interarrival time is set to 0.5 seconds, and the LL policy uses as load index the queue length. Figure 5 shows the stretch factor of the LL policy with FCFS and priority discipline for the diverse types of client devices. We see that the priority discipline can provide a better QoS as expected by more powerful devices that do not require transcoding and should not be blocked by requests requiring transcoding.

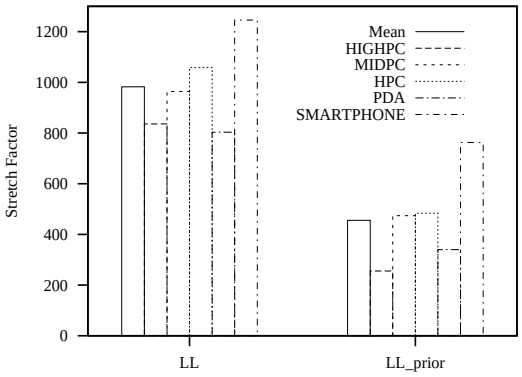

Figure 5: Stretch factor of different client devices for LL policy without or with priority.

We now evaluate the effect of pushing the transcoding to an upper-level proxy when the load is too high, even if a

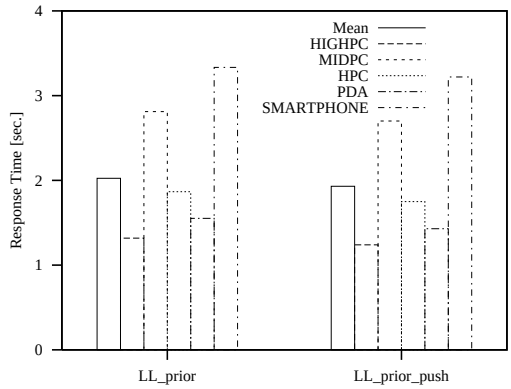

Figure 6: Comparison of LL priority policy with or without pushing strategy.

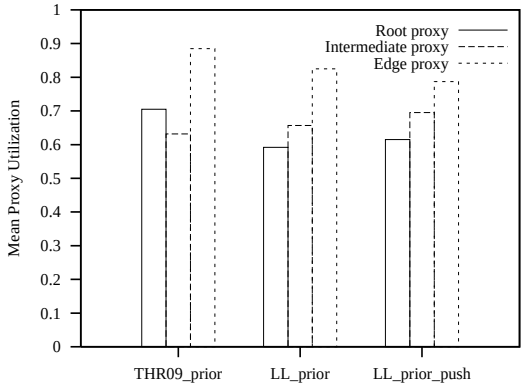

Figure 7: Proxy mean utilization on the most congested path.

more detailed transcodable version of the required object is available locally. In Figure 6, the proxy decides to push up the object if its queue is longer than a pre-specified threshold (in this case 35 pending requests). We see that letting the proxy forward a transcoding task improves slightly the response time perceived by the users. When the edge proxy delegates the transcoding task to upper-level proxies, the time the request spends at the edge proxy is greatly reduced; however, the request incurs in a higher network latency, as it has to travel up and down the proxy hierarchy. Figure 7 shows the advantage achieved by the push strategy from the load sharing point of view. The performance index we use to measure the system load level is the mean utilization of each of the three proxy servers on the most congested path that is, the path from the most skewed edge proxy to the root proxy. The push strategy outperforms the other policies in sharing the load on the most congested path, as it alleviates the edge proxy from the task of processing locally every transcoding request generated from a useful cache hit.

5.2 Caching strategies

In Figures 8 and 9 we use the percentage of cache hit and miss for the different device types to show the tradeoffs between the demand based policy caching the transcoded version and the coverage based policy caching the retrieved version. We observe (in not shown results) that for the coverage based policy the mean response time is above 40 seconds under LLqlen_prior_push, while the same load distribution policy achieves a response time equal to 1.93 seconds under the demand based policy, as shown in Figure 6. This result can be explained by considering the diverse processing load imposed on the system by the two caching strategies. If we compare Figures 8 and 9, we note that under the coverage based policy, the percentage of useful cache hit improves considerably as a more reusable version is cached, while the number of exact cache hits decreases for classes of devices that require transcoding. As a consequence, heavier transcoding requests are generated and their processing causes an overutilization of the system. We conclude that even if the choice of caching the transcoded version increases the percentage of cache miss, it is still better than caching the more detailed original version. We also considered the anticipatory caching policy where both the transcoded and retrieved versions are cached and found that the cache miss increases due to cache size limitation and the response time is worse than just caching the transcoded version.

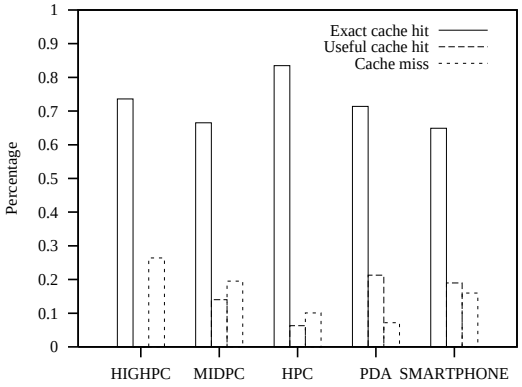

Figure 8: Cache hit and miss percentage under demand based caching policy.

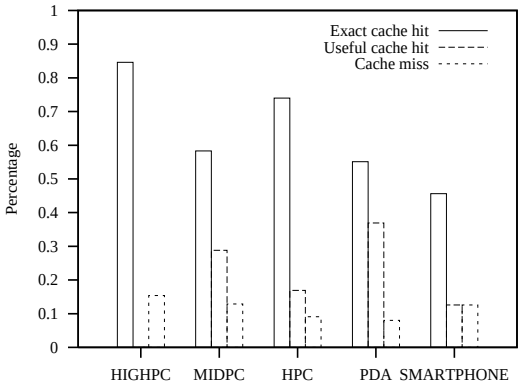

Figure 9: Cache hit and miss percentage under coverage based caching policy.

5.3 Sensitivity analysis

Figure 10 shows the sensitivity to the client distribution among the edge proxies. Specifically, it plots the mean

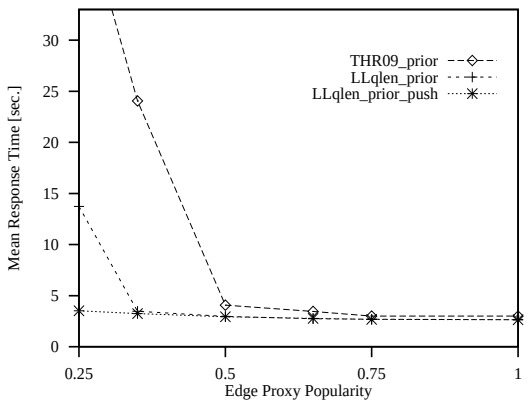

Figure 10: Sensitivity of response time to client distribution among edge proxies.

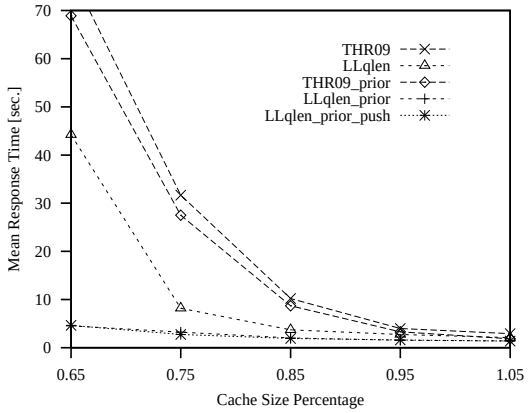

Figure 11: Sensitivity of response time to proxies cache size.

response time against the Zipf parameter α. In previous figures the Zipf parameter α was set to 0.35. (The most skewed function corresponds to $\alpha = 1$) As one could expect, the performance of all the policies improves as the edge proxy popularity is less skewed, that is when client requests distribution resembles more like the uniform function. We observe that the LLqlen_prior_push policy is the least sensitive to this parameter. Even if not shown in this figure, the result achieved by the NOINFO policy does not improve as the client distribution becomes less skewed.

Another interesting aspect is the sensitivity to the cache size. Figure 11 shows that the relative order of the policies does not change for different size of proxy caches. The default cache size used in the previous figures is set to contain 85% (in bytes) of the original objects.

6. CONCLUSIONS

In the coming pervasive computing environment, the same content may have to be transcoded in different forms according to device capabilities. In this paper, we described how a hierarchical collaborative proxy network can adapt objects to client specifications and cache diverse object versions. We found that any load distribution policy that only takes into account the local load condition or no load information at all is not able to spread the computational load caused by transcoding across the proxy nodes. For global type policies, even the simple policy that captures the load information along the request path can provide reasonably good load sharing. Furthermore, to effectively share the load, it is not sufficient just to tune the amount of processing load passing down the proxy hierarchy to the lower-level proxies. We found that it is necessary to provide the edge proxy a mechanism to push up some transcoding load. On the caching policy, we examine policies that allow different versions of an object to be cached. Our study showed that the demand based caching policy, where the transcoding proxy only caches the transcoded object, performs better than the coverage based caching policy that caches only the more detailed source object of transcoding and the anticipatory caching policy that caches both the transcoded object and the more detailed object.

7. REFERENCES

[1] C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the World Wide Web. *IEEE Trans. on Software Engineering*, 11(1):95–107, Jan. 1999.

[2] M. Arlitt, R. Friedrich, and T. Jin. Workload characterization of a Web proxy in a cable modem environment. *ACM Performance Evaluation Rev.*, 27(2):25–36, Aug. 1999.

[3] P. Barford and M. E. Crovella. A performance evaluation of Hyper Text Transfer Protocols. In *Proc. ACM Sigmetrics 1999*, pages 188–197, Atlanta, May 1999.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proc. IEEE Infocom 1999*, Mar. 1999.

[5] S. Chandra and C. Schlatter Ellis. JPEG compression metric as a quality-aware image transcoding. In *Proc. USENIX 2nd Symp. on Internet Technology and Systems*, pages 81–92, Boulder, CO, Oct. 1999.

[6] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel. A hierarchical Internet object cache. In *Proc. USENIX 1996*, San Diego, Jan. 1996.

[7] R. Floyd and B. Housel. Mobile web access using eNetwork Web Express. *IEEE Personal Communications*, 5(5):47–52, Oct. 1998.

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

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

[10] 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.

[11] J. E. Pitkow. Summary of WWW characterizations. *World Wide Web*, 2(1-2):3–13, 1999.

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

[13] W. Willinger and V. Paxson. Where Mathematics meets the Internet. *Notices of the American Mathematical Society*, 45(8):961–970, Aug. 1998.

[14] Wireless Application Protocol Forum. Wireless Application Protocol. http://www.wapforum.org.

[15] 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. 17th ACM Symp. On Operating Systems Principles*, Dec. 1999.