TranSquid: Transcoding and Caching Proxy for Heterogenous E-Commerce Environments

Anuj Maheshwari, Aashish Sharma, Krithi Ramamritham
Center for Intelligent Internet Research
Department of Computer Science and Engineering.
Indian Institute of Technology Bombay
Mumbai, India 400076
anuj@iitbombay.org, ash@ee.iitb.ac.in
krithi@cse.iitb.ac.in

Prashant Shenoy
University of Massachusetts
Amherst, MA 01003
shenoy@cs.umass.edu

Abstract

With the advent of the Wireless Internet, the client space has become heterogeneous in terms of device capabilities. To cater to the needs of these devices in E-Commerce applications, smart intermediaries have been developed to increase the user satisfaction by hiding the inherent weakness of some of the small although handy devices like the PDAs and Web-tops. Transcoding has been a popular technique to render data for small devices that have smaller displays, and lesser colour capabilities. But transcoding comes at the cost of caching at the Intermediary.

In this paper, we describe a transcoding and caching proxy that caches objects for heterogeneous client spaces by maintaining separate caches for different categories of clients (PC, PDAs, Mobiles, etc.) and transcoding the lower fidelity versions from the high fidelity variants at the proxies as opposed to fetching the transcoded variants from the server. To achieve this, the proxy keeps the server-directed transcoding information (if provided by the host server) as part of meta data attached to the cached objects and uses this information to convert its fidelity and modes or uses heuristics. Through such an intermediary architecture that serves a heterogeneous client base, we exploit the availability of cached high-fidelity variants of web resources (brought in to serve the requirements of high-end devices like PC's) to serve low end devices, and thereby decrease latency and bandwidth.

1. Introduction

With the everyday computer user adopting the Internet, the opportunity to communicate and interact for business and personal use has exploded to phenomenal levels. As traditional business takes its steps into the electronic environment, the rate of growth of the e-commerce and m-commerce space has been rapid. The development has gone a step further with the advent of pervasive devices, such as cell phones and personal digital assistants (PDA's). All this has led to a greater potential to do business and to share ideas.

As the figures state, the current mobile subscriptions stand at 170 million. They are forecast to exceed one billion by 2003. According to a study published by Strategy Analytics [1] within its strategic advisory service, Wireless Internet Applications, more than 1.5 billion PDAs, handsets and Internet appliances are to be equipped with the wireless capabilities the end of 2004. In money terms, the global mobile commerce market will be worth 200 billion dollars by then and 130 million customers will be generating almost 14 billion transactions per annum. Taking a look at the mobile devices we can expect 70 percent of the new cellular phones and 80 percent of all the new PDAs to have some form of access to the web and further expect 63 percent of the transactions to be generated by mobile devices [2].

All the above is based on the hypothesis that mobile phones and small handheld devices would be able to provide the quality of service and features desired by a common user to carry out day-to-day business activities. An enabling and essential feature for the same is the ease to access the Internet at a reasonable user satisfaction [1] level. But then one needs to understand that there are several challenges in making mobility a widespread reality. Many problems like lower bandwidth, higher error rates, frequent disconnections, smaller displays are common across mobile phones and PDAs. Another important issue here is that of *heterogeneous client space*.

[1] User satisfaction is the perceived experience of a typical user during Internet surfing.

Heterogeneous client space exists when data is available to users across a growing number of destinations - laptops, desktops, kiosks, automobile browsers, cellular phones, pagers, pocket PCs, Palm OS devices and other handheld devices. Current trends suggest that the evolution of the Internet is towards more and more heterogeneity. In heterogeneous client environments, each destination brings its own unique requirements. Devices present different graphical, bandwidth and display requirements, and may support a variety of data formats (e.g. XML, WML, cHTML, SVG). They have different processing capabilities and constantly evolving standards. Such a heterogeneous client environment is very distinct from the prevailing homogeneous client environments in which the only client type is a personal computer. Every response in a heterogeneous environment is not only a function of the URL but also depends significantly on the kind of the device making the request. This additional constraint in such environments forces a rethink of certain prevalent technologies, which provided performance benefits in homogeneous environments completely useless. One such technology is the caching of web objects.

There is no doubt that caching has increased user satisfaction tremendously by serving web objects locally from a proxy that is near the client. We believe, caching can also provide similar benefits in heterogeneous client environments though this requires additional features at the intermediary and standard protocols for the end user to communicate. One of the fundamental problems that appear in enabling caching for heterogeneous environments is storing of multiple variants of the same object generated because of the transcoding operation at the server side or at an intermediary before the caching server. Presently, all transcoding engines mark transcoded content as un-cacheable. We believe this is unnecessary and leads to wastage of bandwidth and increases latency in a response that needs transcoded data since the content now travels across the Internet when it could very well be served from a location near the client.

We present a novel caching and transcoding system called *TransSquid* that is an extensible and intelligent intermediary for the heterogeneous client environments. *TransSquid* is a modular framework that enables caching in heterogeneous environments by maintaining a multi-level cache and linking it with the transcoder.

This paper is structured as follows - we first present our technique and then discuss its implementation in detail. Then, we look at related techniques that help in enhancing the user satisfaction through mobile devices like PDA's and mobile phones and compare them with our work. We end with a discussion on the open issues and also present the scope of *TransSquid.*

2. Caching in Heterogeneous Client Environments

To place things in perspective, we first discuss caching technology, as it exists today. Cache technology provides for the three primary characteristics - scalability, availability and responsiveness. The fundamental principle of a cache is to provide frequently requested content locally and reduce the latency for the request to be serviced.

With the advent and the widespread usage of the Internet, caches have been used to store web objects. They could be deployed at the end point or as an intermediary between the client and the server. The essential idea is still the same, to provide faster access to content. On the whole, all caching technologies, whether at the end point or at an intermediate location revolve around reducing latency by avoiding slow links between client and origin server. They try to make up for slow connections and network congestion. For ISPs and Content Providers, caching further reduces the overall traffic on the networks and allows them to offer high-performance distribution at low cost.

For a typical web request, caching functionality is embedded into the HTTP protocol. A HTTP [9] transaction begins when the client sends a request with the URL, using a series of HTTP *header requests*. This request is triggered by an intermediary caching proxy that checks for the presence of the requested object in the local cache. If the object is present in the cache, the proxy would append an *if-modified-since* request header with the request. The server responds to the typical request by first sending HTTP *response headers* that contains information about the requested content eg. *date*, *content_type*, *last_modified*, and *content_length*. In case a *if-modified-since* request header is present in the request, the server simply sends a "`304 Not Modified`" reply if the object at the cache is the most recent version else, the client has to download the web object from the end server. When the client request is furnished from the cache, it is termed as a HIT. If the cache does not have the latest version of the requested object and the object is got from the end server, it is termed as a MISS. All caching algorithms try to maximize the probability of a HIT given the limited storage space available with the cache.

As discussed earlier, the constraints in a heterogeneous environment are very different from that of the homogeneous environment. In the next couple of paragraphs, we shall discuss in more detail, the issues and imperatives in such environments from the angle of caching.

Transcoding and content negotiation lead to multiple variants of a resource or a web object that differ in modality and fidelity [2]. For example, a typical PDA client would be

[2] A media resource can be translated to different modalities, such as text to audio, or video to images. Where as different versions within the same modality, occurring for example due to, image compressions, text summarizations and video abstractions are the fidelity variants of a resource.

satisfied with a lesser fidelity variant than the variant for the same resource for a Workstation client that would demand rich features.

proxy in heterogeneous client environments. To summarize, a caching proxy can be designed to:

Table 1. Client Capabilities

Client Type	Bandwidth	Display Size	Color Device	Storage
Work-Station	10 Mbps	1280x1024	24-bit color	20 Gb
Color PC	56 Kbps	1024x768	24-bit color	10 Gb
PDA (high-end)	14.4 Kbps	320x200	8-bit color	16 Mb
PDA (low-end)	9.6 Kbps	320x200	Greyscale	8 Mb
WAP-enabled Mobile Phone	20 bps	132x176	b/w	2 Mb

Essential requirements for intermediaries, especially caching proxies in such environments are, recognition and persistence of the different variants of the same resource. For the former, standards can help in the identification and resolution of variants through attached headers. It is the persistence of multiple variants of the same resource that would require intelligent caches that have a mechanism to categorize them according to one or a set of parameters (namely fidelity, requesting client, content-type) and store them accordingly in a way that makes retrieval fast and simple. This makes the design of such a cache both interesting and challenging.

For the identification of the client type in an heterogeneous environment, it is imperative for the intermediary proxy and the end server to understand the Client Capabilities and Preference Profiles (CC/PP) [4]. CC/PP is a collection of the capabilities and preferences associated with user and the agents used by the user to access the WWW. For an intermediary caching proxy, understanding CC/PP would help it in the classification of clients so that variants of the same resource could be appropriately made available to them. The service and the content provided by a caching proxy would depend entirely on how well the proxy can understand the requirements of a client and thereby, on it ability to deliver the closest possible variant available. CC/PP, thus gives a standard for such communication.

Intelligent caching proxies in a heterogeneous client environment could have the functionality to exploit the heterogeneity through the conversion of high fidelity variants of a resource to low fidelity variant to serve clients at the lower capacities. A typical example of this would be the conversion of high fidelity JPEG image to a low fidelity GIF image in reduced colour and half the dimensions to serve a low resolution PDA client. This would imply that a higher fidelity variant of a resource could be used to satisfy the needs of clients that have lower capabilities, through local transcoding of content at the caching proxy itself.

For this, a transcoding module that is tightly coupled with the caching engine is needed at the intermediary. The above discussion highlights the requirements for a caching

1. Recognize multiple variants of a resource and categorize them in the cache accordingly.
2. Understand the Clients Capabilities and Preference Profiles (CC/PP) to facilitate client recognition.
3. Manipulate fidelity or modality (or transcode) of variants whenever possible to provide better service.

3. TransSquid: A Caching and Transcoding Intermediary

Our novel technique is a multi-level caching solution for the emerging heterogeneous client environments. The *TransSquid* architecture is designed as a smart intermediary that can cache and transcode web objects in an environment where the client requests have to be serviced intelligently according to the client capabilities. *TransSquid* tries to solve the problem of caching in a heterogeneous client environment by taking a *client centric approach* for categorizing different variants and then storing them in a multi-level cache on the basis of fidelity and the modality.

As can be seen in Table 1, there are an ever-increasing myriad of devices for accessing the web. Given this, it would not be sensible to provide for caches for each type of device as the current base of these devices is high and is increasing because of innovation and newer players. If one cache were provided for each device and if a cached object that serves a Windows CE based iPAQ, it would not be able to support a Palm device, which could share the same data given their capabilities. Separate caches for each device would lead to low and perhaps ineffective HIT rates in the cache, which defeats the purpose of the cache itself. *TransSquid*, therefore provides a limited level caching architecture, by dividing the client space into a limited number of (say three) categories based on the capabilities. A client device is a member of one and only one of the 3 categories depending on its capabilities like display size, colors, storage and bandwidth of connection. All web objects accessed from a client device are stored in the cache that the client has membership to.

The 3 major categories that we divide the client space are:

1. **High Capability Clients:**

   The clients that fall in this category are Personal Computers, Work Stations and Laptop Computers. These devices have large storage capacities (typically more than 64 MB RAM and 10 GB or more disk space), large screen size (640 x 480 pixels or above), multimedia support, and good processing power. These devices have functionality to view video, audio and high resolution images. Content available on the Internet is default for such kind of computing devices.

2. **Medium Capability Clients:**

   Portable Computers like PDA's and WebTops fall in the category of Limited Capability clients. Typical examples of such devices are the Palm Pilots and iPAQ's. The demand for such portable and handy devices has grown exponentially. These devices have smaller screen size (typically 320 x 200), limited colors, lesser processing power (100 Mhz) and are connected to the Internet through slow wireless links. Some devices in this category also have audio functionality.

3. **Limited Capability Clients:**

   These are very low-end devices that have been used to surf the internet only for score updates, news headlines and other text-only or have very low graphics support. Some models of mobile phones have come up with larger screens, but inherently the data transfer to these devices is very slow. SMS - a popular technique for data transfer to mobile phones has a bandwidth of 20 bytes per second. Though with new Wireless Transfer Protocol (WAP) and other efforts from the research community this has improved, surfing using a the mobile phone remains slow.

As can be seen, we take a middle path. We justify our categorization into a limited sets by saying that broadly the requirements of all clients in a particular category are the same. Though, certain differences in capacities might exist, we take a broader view for user satisfaction. The overall picture is represented in Figure 1. Any necessary modification could be done on the fly. Our experiments suggest that PDA devices with Windows CE and the Palm OS (the two major players in the small device OS market) give almost the same visibility and feel for content. Though different transcoding engines have proprietary techniques for transcoding, we feel that the difference in the rendering for these clients that are close to each other in CC/PP is not very high, and hence the same cached object could provide the necessary level of user satisfaction.

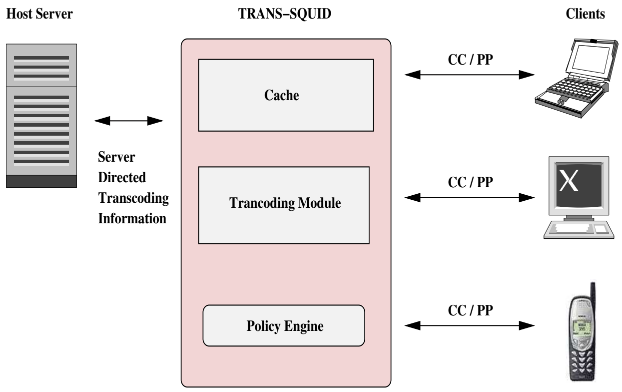

Figure 1. Basic scenario for TransSquid

If certain transcoding needs to be done on an object in a cache to suit the requirements of a request made from the client in the same category - it could be done on the fly or at the client side, but the performance benefits provided by giving a Cache HIT would outweigh the less optimal solution in transcoding.

Intra-cache communication between the different levels in *TransSquid*, allow the low end users to benefit from the availability of high fidelity content. Our assumption is that normally a high fidelity variant can be converted into a lower fidelity variant and hence objects of a higher fidelity cache can respond to a request for a lower fidelity cache by local transcoding. This is broadly true for images and HTML data whose fidelity can be changed by decreasing resolution or by removing unnecessary information. We term this as Partial HIT and discuss this concept in our proceeding discussion.

3.1 Notion of Partial HIT

HIT and MISS are the two primary events that take place when the client sends a request for a object to the network. Depending on attributes like *if-modified-since* and client preferences, the object is either returned from the cache or fetched from the server, as seen in our earlier discussion on caches.

In the *TransSquid* architecture, when a client with low fidelity requirements, makes a request, for which the cache already contains a higher fidelity variant but no object in the cache that the client maps to exactly, the cache returns a Partial HIT. In such a case, the cache sends the object to the transcoding module of the *TransSquid* architecture, which uses the information available in the meta-data of the Object, the *content_type* and Characteristics of the Object as its input to return a suitable variant of te resource. If the meta-data contains the directive given by the host server (based on information semantics) then this information is used for transcoding. This is called *Server Directed Transcoding* [21].

A Partial HIT is more time consuming than the HIT in which the object is uploaded from the disk and served to the client. The additional time is primarily consumed in - firstly, trying to determine the fidelity and the modality of the variant from the variant available in the cache, and secondly, time taken to perform the actual transcoding operation. This is still an order above the MISS case, in which the object is fetched from the server, and possibly transcoded at an intermediary as well. Figure 2 shows the relative times taken by MISS, HIT and a Partial HIT.

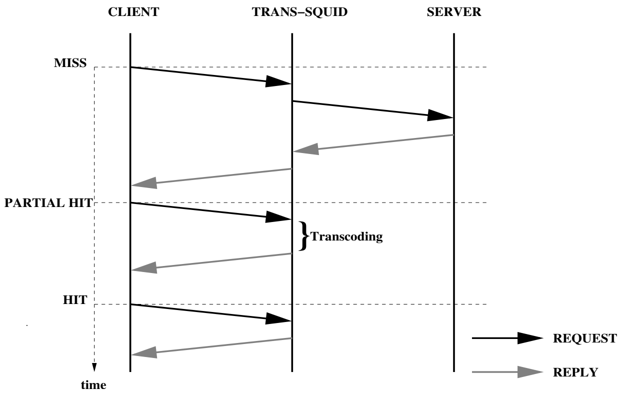

Figure 2. Concept of a Partial HIT

In this section, we saw the *TransSquid* architecture and the problem that it tries to solve. Our solution is based on certain assumptions that are based on our understanding of the WWW and also the heuristics and rules of thumb available for the Internet in general. In the next section, we discuss the implementation of *TransSquid*.

4 Implementation of TransSquid

4.1 Architecture

The *TransSquid* is designed as a modular framework where each module has a specific function. The major parts of the *TransSquid* are:

1. Multi-level Caching Module
2. Transcoding Module
3. Client Side Module
4. Policy Engine

A schematic representation of the architecture is shown in the Figure 3. The Multi-level Caching Module is structured in the form of three caches that store the web objects

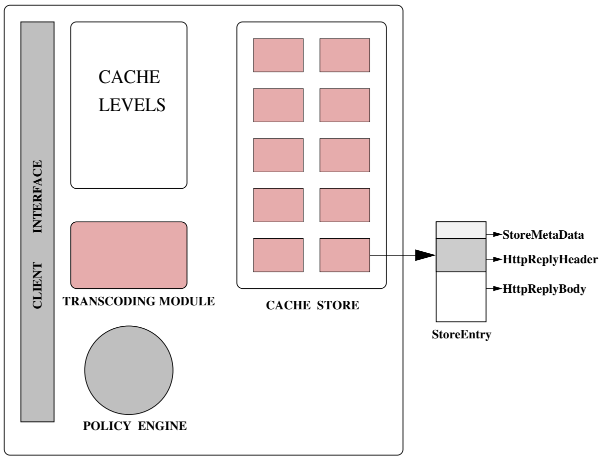

Figure 3. Architectural overview of *TransSquid.*

according to the requesting client type. Each level functions as a separate cache though they have a central persistence. The cache replacement policy for the cache is FIFO. The cache replacement policy by itself in such architecture is an interesting and unexplored issue for research.

These caches are hierarchical as the caches serving the high capability clients can also serve requests from devices like mobile phones and PDA's after passing the objects through the transcoding module. As discussed later, the transcoding module renders the objects for lower capability devices.

The Transcoding Module in our implementation is a heuristic based web object processor, which recognizes two types of objects - images and text. For images the transcoding modules applies functions like reducing the dimensions, decreasing the number of colors and decreasing the quality factor of JPEG images. The function to be performed on an image is chosen through the policy engine or by using any server directed information appended with the web object. For text, we apply simple techniques like removing unwanted parts like advertisements, buttons and unnecessary information replacing them by simple text links.

The Policy Engine provides the Transcoding Module with information about current modality and fidelity of a web object and also suggests the function that should be carried out to render it for other devices types. Again, this is based on heuristics like *content_type*, size, and dimension, in the case of images.

The Client Side Module is the interface between the client and the proxy. It is used for intercepting the client requests and mapping it to the correct cache. The client side maintains a client specific state so that a client need not advertise its capabilities and preferences (CC/PP) every time it makes a request.

We use the Squid web proxy cache as the basic platform for our implementation. Squid [5] is an open-source, high-speed, Internet proxy-caching program. We use Squid version 2.3.STABLE 4 available freely on the Internet. Our code is written in C language. We chose Squid as a platform for its robustness and wide spread usage, and also because hierarchies of squid proxies, arranged in complex relationships are possible. Our code uses the caching functionality of Squid for implementing the multi-level cache, though the Transcoding Module and Policy Engine are written using some of the commonly available libraries like lib-jpeg [6].

Our contribution thorugh the *TransSquid* is in the form of:

- Enhanced work-flow of a typical request.
- Intelligent storage caches that can communicate with each other.
- Addition of a Transcoding Module and Policy Engine, to show a proof of concept *Caching and Transcoding Intermediary.*
- Interaction between the caching related processes and transcoding related functions for higher optimization.

4.2 Data Structures

This section describes some of the key data structures that have been changed from existing Squid implementation or that have been added.

The Data Structures for the caching functionality of the *TransSquid*, not only need to keep typical caching information but at each point of a typical request-response process, they need to keep track of the requesting client type. This information is required to base decisions like which cache to retrieve or store.

In Squid, the *request_data* and the *reply_data* are linked together using the *httpStateData* which contain the *request_t* data structure. In *TransSquid*, this data structure is modified to contain another attribute called *device* which contains information on the type of client making the request. Additionally, the *StoreEntry* that keeps information on the storage of a web object in the cache, also has an attribute called device. This attribute is used to keep track of which cache (*store_table*) should it be linked to as the Squid now maintains three different caches. Internally the caches store all the data in the same central cache.

Transcoding information is stored in a data structure called *transcode_info* and the Policy Engine usually generates it. The client registers its CC/PP to the proxy and then uses a unique key through which the *TransSquid* knows its CC/PP for the later sessions. The Client Site maintains persistence storage and has appropriate data structures for keeping such state information.

4.3 Flow of Typical Request

The flow of a typical request in the Trans-Squid architecture is as follows.

1. A client connection is accepted by the client-side. The HTTP request is parsed. The client device is identified and the *clientHttpRequest* device flag is set.
2. The access controls are checked. The client-side builds an ACL (Access Control Layer) state data structure and registers a callback function for notification when access control checking is completed.
3. After the access controls have been verified, the client-side looks for the requested object in the cache. If it is a cache HIT, then the client-side registers its interest in the *StoreEntry*. Otherwise, Squid needs to forward the request, perhaps with an *if-Modified-Since* header. If a Partial HIT is encountered, Squid calls the Transcode Module and transcodes the content of the *HttpReply* according to server directed transcoding information information that is stored in the *Store_Meta_Data* or according to its Policy Engine. Once this is done the request is forwarded to the client, and the new variant of the *StoreEntry* is added to the cache (*hash_table*) of that client type, and the client-side registers its interest in that *StoreEntry*.
4. The HTTP module first opens a connection to the origin server or cache peer. If there is no idle persistent socket available, a new connection request is given to the Network Communication module with a callback function.
5. When a TCP connection has been established, HTTP builds a request buffer and submits it for writing on the socket. It then registers a read handler to receive and process the HTTP reply.
6. As the reply is initially received, the HTTP reply headers are parsed and placed into a reply data structure. As reply data is read, it is appended to the *StoreEntry*. Every time data is appended to the *StoreEntry*, the client-side is notified of the new data via a callback function. Meta information from the server side like the transcoding details for various devices is added to the *Store_Meta_Data*. This information is used if a client encounters a Partial HIT as discussed in step 3.
7. As the client-side is notified of new data, it copies the data from the *StoreEntry* and submits it for writing on the client socket.

Figure 4. A GIF image of size 3473 bytes transcoded into a much smaller sized version of 1473 bytes as represented below it.

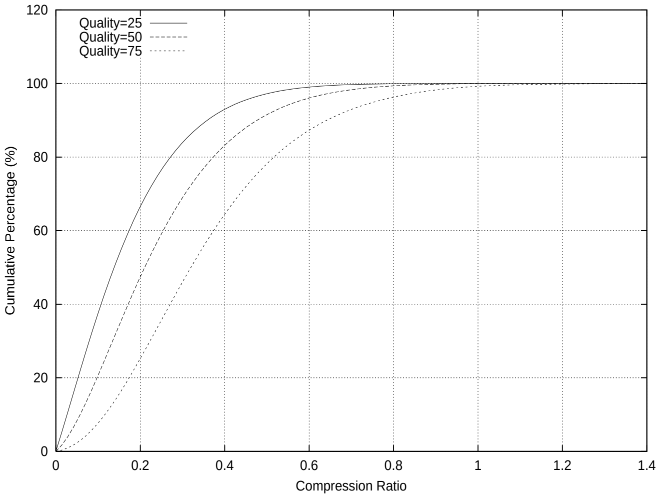

Figure 5. Transcoding GIF (size $>$ 5 kB) images to JPEG images.

8. As data is appended to the *StoreEntry*, and the client(s) read it, the data may be submitted for writing to disk.

9. When the HTTP module finishes reading the reply from the upstream server, it marks the *StoreEntry* as *complete*. The server socket is either closed or given to the persistent connection pool for future use.

10. When the client-side has written all of the object data, it unregisters itself from the *StoreEntry*. At the same time it either waits for another request from the client, or closes the client connection.

4.4 Transcoding Results

In *TransSquid*, various transcoding techniques are used for changing the modality and fidelity of web objects. For images, some of the transcoding techniques used are based on:

- **Size** : minify, subsample
- **Fidelity** : JPEG Compress, GIF compress, reduce resolution
- **Colour Content** : reduce colors, convert to grey, convert to b/w

The transcoding for a web object is a function of its current mode (HTML, jpeg/image, gif/image) and the current client category it serves and that it would aspire to serve. For images, it is further a heuristic function of its resolution, colour, size (in bytes), geometry and priority.

For our analysis, we work on a a total of 9336 images, which comprise of 51.2 % GIF's and rest JPEG's. Further we classify, images based on size, colour and purpose. Each such combination has a function preassigned that will be used for conversion between category of clients.

A typical transcoding operation is shown in Figure 4. The transcoding engine converts a GIF image with 256 colours for PC user into a reduced colour (b/w) GIF for a Palm V user. The simple function executed by the transcoding module is the reduce color operation. The size of the image reduces from 3473 bytes to 1473 bytes without any significant compromise in quality.

Some of the other operations like conversion of high colour GIF's above 50 Kbytes into JPEG's are based on analysis done on our sample image collections. Figures 5-7 show some of the statistical results obtained from transcoding operation performed on images. Here, the resulting image size as a percentage of original image size is plotted as a cumulative distribution. We see in Figure 5, transcoding from GIF to JPEG provides atleast 50% decrease in size, for 90% of large images (above 50 Kbytes). This comes out at hardly any difference in perceivable quality if the resultant image is viewed through a limited capacity client. On the other hand, the same function doesn't work for GIF's of smaller size (below 5 Kbytes) as shown in Figure 6. Figure 7 illustrates that spatial geometry reduction leads to a image size reduction both for GIF's and JPEG's.

5. Related Work

The objective of this section is to compare our line of thought and implementation with other research efforts. Ex-

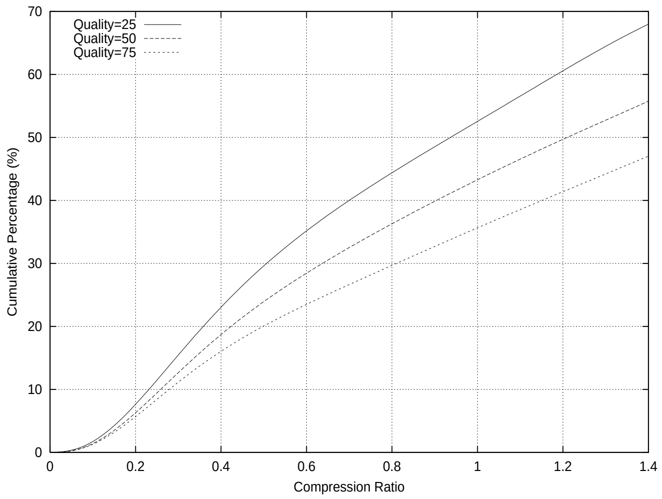

Figure 6. Transcoding GIF (size $<$ 5 kB) images to JPEG images.

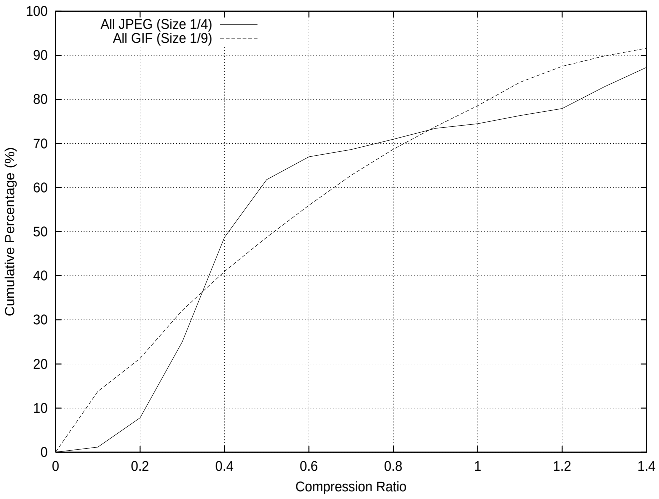

Figure 7. Transcoding that changes the Spatial Geometry of Images.

amples of commercial products and research prototypes in this area include Spyglass [12], ProxiNet [23], Intel QuickWeb [11], and IBM Transcoding Proxy [2]. These products have a focus towards providing quality of service in heterogeneous client environments through transcoding. Our approach on the other hand has been coupling caching and transcoding to provide a enhanced service through which the clients can reap the benefits of the heterogeneity and also have faster access through caching of static content.

A line of thought that might be of some relevance here is the Soft Caching Technique [8, 17, 18]. In Soft Caching, proxies should be able to perform recoding of the images in the cache so that lower resolution versions of the images can be stored and made available to the clients. Here, the idea is to provide a client with a lower resolution of image until the client explicitly asks for a higher resolution. Hence, if a client wants a high resolution image, it would have to explicitly ask for it. If we interpolate this to heterogeneous client environments, it could mean that every client gets the lowest resolution variant of a image (assuming progressive compression) until the client specifically asks for its high resolution variant. The problem with this approach is that it would be an additional burden on the user to keep requesting for higher resolution images until it becomes satisfied. And besides, it would mean storing multiple variants of a resource some of which might not be required by any device.

IBM Transcoding Proxy which is a part of the IBM WebSphere EveryPlace Suite Version 1.1 [3] does caching for transcoded contents by implementing a flat cache that stores objects. Such a cache would have a low HIT rate since the number of devices might be large in which case a HIT is returned if the device requesting is an exact match to the device that had earlier requested the object in the cache.

Significant recent work for development of e-commerce over the heterogeneous client space has been done in the personalization systems like the AvantGo Channels [24]. Another line of work has been in the field of providing offline browsing support for mobile devices over wireless channels. The Mowgli System [10] uses its own variant of the HTTP protocol to communicate over wireless channels to reduce data flow. All these and many more techniques are being developed to increase the migration of a user to using mobile clients like PDA's, WebTops and Internet over mobile phones.

5.1 End-To-End versus. Intermediary Approach

An interesting point of discussion in the context of heterogeneous client environments is the end-to-end approach vs the intermediary or proxy based approach. The end-to-end argument suggests that all the manipulations of a web object should be done at the end points of the network and not at the intermediary level. On that other hand, the intermediary based approach suggests the opposite.

The intermediary based approach has its own disadvantages. Firstly, when an intermediary transcodes data it completely loses the semantic information that the object related to. This could be corrected through a *Server Directed Transcoding* Information appended with the web object in the form of HTTP headers. Secondly, we have reasons to believe that the intermediary based approach would not be very scalable if we increase the myraid of internet devices since transcoding is a resource intensive process. This could be offloaded by having intelligent caches at the intermediaries, that reduce the load on the server.

The end-to-end arguments would require all the end servers to provide for transcoding– this is something that cannot be assumed. The inherent advantage though, of the end-to-end approach is that the end server could carry out more optimized and complex transformations based on client needs. It can do these operations offline and hence just keep multiple versions of a resource in its cache to serve different clients. A possible point of concern is the increasing variety of access devices.

Though the functionality provided through the end-to-end approach is more optimized, it is still not possible to assume that all end points would have such functionality in the near future. *TranSquid* tackles the problem by addressing the increasing variety of client devices, something which is faced in both end-to-end and intermediary approaches, by categorising devices into a few sets and then performing caching and transcoding intelligently. And hence, it makes sense to keep the intermediary based transcoding as an option but let the end-to-end approach take over wherever possible.

The *TransSquid* multi-level caching and transcoding end provides for support of both the end-to-end approach and the intermediary based approach. If the end servers provide transcoding support then the *TransSquid* just behaves as a caching engine, providing a mechanism to decrease latency of popular requests by serving them through the cache. In case the end-to-end approach is not followed, then *TransSquid* has the additional task of not only transcoding but also storing transcoded variants in the Cache. Thus, we feel that TranSquid is a scalable solution to the emerging heterogeneous client space.

6. Discussion

In this paper, we have presented a system that supports caching and transcoding in the heterogeneous client space. Our system uses a multi-level cache architecture and caches transcoded versions of web objects which typically are marked non cacheable since the present flat cache architecture cannot handle different variants of the same resource. Not much work has gone into the field of caching transcoded web content. Our main focus, has been to increase the perceived user satisfaction through the delivery and manipulation on content locally rather than going through interoceanic distances. Our caching architecture uses the heterogeneity of the client space to provide better service to clients with low fidelity through the Partial HIT. Our system provides support for both the end-to-end approach of doing trancoding at the host server and for the intermediary approach where the proxy intercepts the request and renders data.

In our on going work we try to improve upon the functionality both in terms of the increasing transcoding capability as well as improving cache design. Our proxy should be an intelligent entity that can make decisions in various scenarios such as, if the server has the capability of providing data in various fidelity versions that the client might want, and/or the server data might be dynamic and changing in time. Some issues might need to be worked out in cases like these. If a server has multiple versions of a media resource a requirement for a lower fidelity object can either be serviced by a server fetch or by transcoding from an appropriate higher fidelity version that might be cached.Now this would be an important decision that the proxy would have to make based on various factors including reducing the user latencies, current load on the transcoding engine and network overheads involved in actually fetching the object from the server. We are right now trying to simulate such conditions and working upon algorithms to base such decisions on.

We have already discussed the issue of a Partial HIT. Equally parallel to that could be the case when the cache has a lower fidelity version than requested. Such a request can again be serviced by a server fetch followed by transcoding to the appropriate fidelity or the other method could be to cater the client request with a lower fidelity version than was requested by it. The latter could serve for reducing the latencies in instances where the network delays are very high and server fetch could take time. A transcoding engine would base such decisions on factors like network overheads and delays, and on current load on the transcoding engine. Issues like these also need to be addressed.

Besides this, other similar issues can be studied in the realm of cache replacement policies that would be needed to refresh and update the a cache with our kind of architecture. Some more immediate issues arise such as, whether to make the separate cache levels watertight and have different replacement policies for each, or, have a single cache replacement scheme for all the levels. Scenario one would need to have efficient intercache communication protocols for optimization, while the second would have a simpler structure overall. A plethora of research issues can be studied on this front.

Acknowledgments

We thank Prof. Abhay Karandikar, Department of Electrical Engineering, Indian Institute of Technology Bombay for his support.

This research was supported in part by Tata Consultancy Services, Intel, IBM, and Microsoft, as well as NSF grants IRI-9619588, CDA-9502639, CCR-0098060, CCR-9984030, CDA-9502639, and EIA-0080119.

References

[1] Wireless Internet and Mobile Applications Division, Strategy Analytics. “Mobile Commerce Market Opportunity Report,” http://www.strategyanalytics.com/cgi-bin/greports.cgi?rid=061999120631

[2] IBM, IBM Web Intermediaries(WBI), “Transcoding Publisher.” http://www.almaden.ibm.com/cs/wbi/

[3] IBM, WebSphere Transcoding Proxy. http://www-4.ibm.com/software/webservers/transcoding/

[4] Composite Capability/Preference Profiles (CC/PP): “A user side framework for content negotiation.” W3C Note, http://www.w3.org/TR/NOTE-CCPP/ (07/1999).

[5] The National Laboratory for Applied Network Research, A distributed testbed for national information provisioning. http://ircache.nlanr.net/

[6] T. Lane, P. Gladstone, L. Ortiz, J. Boucher, L. Crocker, J. Minguillon, G. Phillips, D. Rossi, and G. Weijers, “The independent jpeg group’s jpeg software release 6b.” ftp://ftp.uu.net/graphics/jpeg/jpegsrc.v6b.tar.gz.

[7] Masahiro Hori, Go Kondoh, Kouichi Ono, Shin-ichi Hirose, and Sandeep Singhal, “Annotation-based Web content transcoding.,” *Proceedings of the 9th World Wide Web Conference (WWW-9), Amsterdam.*

[8] Antonio Ortega, Fabio Carignano, Serge Ayer, and Martin Vetterli, “Soft caching: Web cache management techniques for images”, *IEEE Signal Processing Society 1997 Workshop on Multimedia Signal Processing, Princeton NJ*, Jun 1997.*Wisconsin*, June 1998.

[9] Wessels, D., “Intelligent Caching for World-Wide Web Objects”, *Proceedings of INET-95, 1995.*

[10] M. Liljeberg, M. Kojo and K. Rattikainen, “Enhanced services for world wide web in mobile wan environment”, http://www.cs.Helsinki.FI/research/mowgli/mowgli-papers.html,1995.

[11] Intel, “Intel Quick Web.” http://www.intel.com/quickweb.

[12] SpyGlass Prism. http://www.spyglass.com/products/prism.

[13] Armando Fox, Steven D. Gribble, Eric A. Brewer and Elan Amir, “Adapting to network and client variability via on-demand dynamic distillation”, *ACM SIGPLAN Notices*, vol. 31, no. 9, pp. 160-170, Sept. 1996, Co-published as *SIGOPS Operating Systems Review 30(5),* December 1996, and as *SIGARCH Computer Architecture News*, 24(special issue), October 1996. in *Proceedings of the Sixth International World Wide Web Con-ference, Santa Clara, CA,* April 1997.

[14] Surendar Chandra, Ashish Gehani, Carla Schlatter Ellis, and Amin Vahdat, “Transcoding characteristics of web images”, Tech. Rep. CS 1999 17, Department of Computer Science, Duke University, November 1999, (submitted to WWW9).

[15] Tim Berners-Lee, R. T. Fielding, H. Frystyk Nielsen, J. Gettys, and J. Mogul, Hypertext Transfer Protocol HTTP/1.1, January 1997.

[16] John R. Smith, R. Mathur, and Chung-Sheng Li, “Transcoding Internet Content for Heterogenous Client Devices”, *Proc. IEEE Int. Conf. on Circuits and Syst. (ISCAS)*, May 1998.

[17] Jussi Kangasharju, Younggap Kwon and Antonio Ortega, “Design and Implementation of a Soft Caching Proxy”, *3rd Intl. WWW Caching Workshop, Manchester, England,* June 1998.

[18] Jussi Kangasharju, Younggap Kwon, Antonio Ortega, Xuguang Yang and Kannan Ramchandran, “Implementation of Optimized Cache Replenishment Algorithms in a Soft Caching System ”, In *IEEE Signal Processing Society Workshop on Multimedia Signal Processing, Redondo Beach, CA,* Dec 1998.

[19] Brian D. Davison, “A survey of proxy cache evaluation techniques,” In *Proceedings of the 4th International Web Caching Workshop*,April 1999. http://www.ircache.net/Cache/Workshop99/Papers/davison2-final.ps.gz.

[20] Anja Feldmann, Ramon Caceres, Fred Douglis, Gideon Glass, and Michael Rabinovich. “Performance of Web proxy caching in heterogeneous bandwidth environments.” In *Proceedings of the INFOCOM ’99 conference*, March 99.

[21] Adaptive web caching research homepage, University of California, Los Angeles http://irl.cs.ucla.edu/AWC/

[22] Jeffrey C. Mogul, “Server-Directed Transcoding,” In *Proc. 5th International Web Caching and Content Delivery Workshop, Lisbon, Portugal,* May, 2000.

[23] ProxyNet. http://www.proxynet.com

[24] AvantGo. http://www.AvantGo.com