ON THE SCALABILITY OF AN IMAGE TRANSCODING PROXY SERVER*

Anubhav Savant, Nasir Memon, Torsten Suel

CIS Department
Polytechnic University
Brooklyn, NY, USA
anubhav@vip.poly.edu, {memon,suel}@poly.edu

ABSTRACT

Image transcoding proxies are used to improve Web browsing over low bandwidth networks by adapting content-rich web images to bandwidth-constrained clients. Such transcoding proxies dynamically analyze, manipulate and transcode images (e.g. quality reduction, down sampling) on the fly enabling significant reductions in download times over low bandwidth links. However, transcoding proxies have scalability problems if the objective policy that decides whether to transcode an image does not take the client load (e.g. number of concurrent clients) into consideration. We show that seemingly intuitive policies that make decisions solely based on whether transcoding yields savings in transmission time fail to scale. Under high load, the average latency perceived by a client can be improved by a factor of about two by taking overall client load into consideration and properly scheduling transcoding operations on a single CPU. We show that an *Earliest Deadline First Based (EDF)* scheduling policy further improves transcoding performance.

1. INTRODUCTION

Many wide area mobile clients still access the Internet using second-generation wireless technologies such as CDPD, CDMA and GSM. These wireless technologies have constraints in terms of network bandwidth, latency, connection reliability, and access costs. In the context of Web access, the problem of slow and expensive networks is aggravated by the size of the objects that exist on the web. Studies [1] have shown that the average Web page access is about 61 Kbytes in size. Of these, 67% of the data represents images (mostly GIF and JPEG) and 22% is text (HTML files), and the average page has about 15 embedded images. Downloading such as page via a non-persistent connection would take around a minute on a typical 2G cellular connection, with data rates around 10 Kbps and round trip delay of 0.7-1.0 sec. Moreover, slightly faster data connections, e.g., based on GPRS, also have high round trip delays and typically charge a high cost per MB transferred. Unfortunately, most Web servers do not give any consideration to the network connection between them and the client. They return objects assuming the client is capable of efficiently receiving and rendering the object.

Transcoding proxies are used to improve Web browsing over slow wireless networks by adapting content-rich Web images to bandwidth-constrained mobile clients [2, 7, 6, 4, 12]. Such transcoding proxies dynamically analyze, manipulate and transcode images (e.g. quality reduction, down sampling) on the fly enabling significant reductions in download times over low bandwidth links without requiring modifications to Web servers and browsers. A typical architecture of a transcoding proxy is shown in Figure 1. In reality, the transcoding proxy is built into an HTTP proxy as a subsystem. The transcoding subsystem analyzes the input image, decides if it is worthwhile to transcode (based on some policy) and then invokes one of the actual algorithms (e.g. image quality reduction, image down sampling) along with its parameters for image transformation. In most cases, the decision regarding the transcoding policy to be used is based on a number of criteria, including: characteristics of the image (e.g. image spatial dimension), bandwidth estimates on the client-to-proxy and proxy-to-server links, user preferences, and the client's device characteristics such as display capabilities.

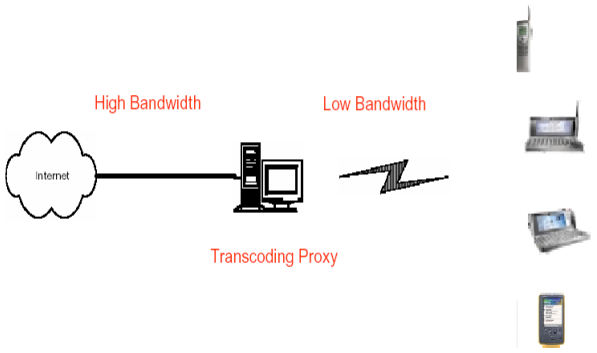

Fig. 1. System Architecture

Image transcoding is a CPU-intensive task, which usually requires the transcoding algorithm to examine each pixel in the input image possibly multiple times. The number of transcoding jobs in the system at any time will be in direct proportion to the number of concurrent clients. Thus, as the number of clients on the proxy increases, each transcoding job would take more time to complete.[1] In such a case, a transcoding policy that does not consider the client load while making transcoding decisions (e.g. as described in [7]) does not scale, i.e., performance for each client degrades. This has a direct consequence on the number of machines required for service deployment as it severely limits the number of clients that can be efficiently served by each proxy. To minimize the cost for a given user population, a service provider would like maximum scalability in terms of client load with only

*This work was supported by a grant from Intel Corporation.

[1] Either due to concurrent execution of different transcoding tasks or, in our architecture, due to the waiting time in the transcoder's queue.

modest degradation in service, particularly during peak times.

SPAWN Proxy: We have implemented a scalable dual-proxy [8] architecture called SPAWN (Scalable Proxy Architecture for web access over slow Wireless Networks); see [9, 11] for details. SPAWN is designed to overcome delays associated with Web access over slow wireless networks by combining a variety of known and new techniques, based on compression and differential compression of HTML pages, transcoding of images, caching of old and similar content, and optimization of the transmission protocol (HTTP) to avoid extra roundtrips. SPAWN's proxy server is based on the *single-process event driven* architecture for maximum concurrency. The event-driven approach implements the processing of each operation as a finite state machine, where various events, either system or user generated (e.g. readiness or completion of a network I/O), trigger transitions between states, allowing an application specific centralized control over all the events. SPAWN makes use of this architecture to implement intelligent scheduling decisions for efficient resource allocation.

In this paper, we are only interested in analyzing the effects of SPAWN's image transcoding component on the overall system performance when subjected to different client load levels. We show that even at for moderate number of clients performance drops significantly if the image transcoding policy does not considers client load while making transcoding decisions. We show that the performance can be improved significantly by considering client load as one of the criteria while making transcoding decision and by properly scheduling transcoding operations on a single CPU.

2. IMAGE TRANSCODING

Transcoding is an important technique that allows a HTTP proxy server to customize the delivered image (embedded in a Web page) for the network bandwidth available on the "last hop" to the mobile client. Common transcoding operations are down-scaling, color reduction, and reducing the JPEG quality factor. For a modern workstation transcoding time is small compared to the time saved in transmission of a transcoded image. However, as the number of clients on the proxy system increases, the rate at which transcoding needs to be done becomes more than the transcoder's speed. As a result image transcoding becomes a bottleneck. For example, if we assume average transcoding speed to be 1 Mbps and average transcoding ratio (reduction in size) to be 2, then the transcoder can only serve 50 clients at a rate of 10 Kbps. Thus, the overall transcoding time for the i^{th} client also includes the transcoding times for the other $i-1$ clients as part of the queue latency for the i^{th} client. The average queue latency for all the clients increases with the number of clients on the system. Once the average queue latency is more than the average latency of sending a transcoded image to the client, the overall latency of a Web page as perceived by a client increases linearly with the number of clients.

In all our experiments, we used a model (simulator) for transcoding images instead of the actual algorithm in SPAWN. Using a model allows us to generalize results without having to depend on any particular transcoding algorithm. For any transcoding algorithm, the two critical parameters, called *transcoding vector*, are: *transcoding ratio* and *transcoding speed*. Our model accepts these two parameters and simulates a transcoding operation simply by deleting the appropriate number of bits as calculated from the transcoding ratio and sleeping for an amount calculated from transcoding speed. We understand that the model has practical limitations as it is generally hard to know the transcoding vector in advance on an image by image basis. However, in a previous study [7], it was shown that the transcoding vector for an image can be reasonably well predicted based on basic image properties.

3. SCHEDULING TRANSCODING OPERATIONS

In order to minimize average latency for the case of multiple connected clients, an image transcoding proxy server needs to make various decisions regarding scheduling transcoding jobs, such as:

- *How to schedule transcoding operations to satisfy some objective (e.g. minimizing average latency):* There are many known scheduling algorithms such as "First-Come First-Serve (FCFS)", "Round-Robin (RR)", "Shortest Job First (SPT)", "Earliest Deadline First (EDF)" etc. They were proposed for different purposes: some of them intend to optimize overall service response time, some of them focus on fairness in accessing a common resource. In this work, we compare the costs and benefits of RR and EDF scheduling policies.
- *How to decide whether to perform transcoding:* The transcoding subsystem in an HTTP proxy analyzes and transcodes images based on a number of criteria such as: the characteristics of the image (i.e. content analysis), the current estimate of the bandwidth on the client-to-proxy link, the characteristics of the client device, and user preferences. Based on these criteria, the transcoding subsystem decides whether it is beneficial to transcode (i.e. whether the reduction in response time justifies the delay introduced as a result of this transcoding) and, if so, what transcoding policy to use. Almost all prior work on transcoding proxies has concentrated on developing efficient algorithms based on the above criteria. However, as the number of clients on the system increases, the length of the transcoding queue also increases. As a result, the transcoding delay for a particular image increases due to additional waiting time in the queue. In such a case, the transcoding proxy should also consider client load as one of the criteria to decide about the transcoding policy in order to provide each client a similar quality of service as under the single client case.

  We introduce the notion of "deadline" for a client, which efficiently captures our idea of keeping a client's connection busy all the time. The deadline is based on how much data has already been sent to a client for a particular Web page, and how much is still waiting to be transmitted after having been transcoded. Deadlines are initialized and updated for each client. Define:

  BW_i - bandwidth to client i
  b_i^j - bytes sent in the j^{th} *write()* to client i
  D_i^j - deadline after j^{th} *write()* to client i
  T_c - current time

  Then

$$D_i^0 = \frac{b_i^0}{BW_i} + T_c$$

  and

$$D_i^j = D_i^{j-1} + \frac{b_i^j}{BW_i}$$

  Thus, if many deadlines are missed, this implies that proxy-to-client connections are often idle due to the transcoding bottleneck. To avoid this, the transcoding subsystem decides to transcode for a client only if it would not miss the deadline.

- *How to select an image for transcoding:* It is common to have multiple images available for a particular client that need to be transcoded. Recent work in [3] suggested that not all images are suitable for transcoding, as it would not result in efficient savings in terms of byte size. For example, small GIF images representing bullets, logos etc. are not suitable for transcoding. For each embedded image received from the Web server, the transcoding subsystem first decides whether it is beneficial at all to transcode, based on image characteristics and independent of client load.[2] Next, to decide whether to transcode an image for a client given the current client load, the transcoding system checks whether the deadline is missed or not. The transcoder always tries to pick an image with an expected transcoding time closest to the difference of current deadline and current time. If we cannot transcode any image for a given client without missing the deadline, then we transmit in non-transcoded form all the images that are marked as not suitable for transcoding, or an image that would give minimum benefit.

4. PERFORMANCE EVALUATION

In this section, we present experimental results that compare the performance of an image transcoding server under the different scheduling policies and optimizations mentioned earlier. The main goal of these experiments is to support our claim that for a transcoding proxy server better service can be provided by considering client load and by using non-traditional service order. We will also demonstrate that it is not efficient to transcode always even in the case when the savings in transmission time are more than the time to produce a useful transcoded image.

4.1. Experimental Setup

4.1.1. Experimental Environment

To generate HTTP requests from a set of concurrent clients, we implemented our own custom workload generator (*SimUser*). SimUser makes requests using synthetic clients, each of which operates in a loop, alternating between requesting a complete web page (including all embedded images) and lying idle.[3] The load that SimUser generates is varied by varying the number of simulated clients. All our tests were conducted using two client machines (evenly splitting the SimUser clients) running Windows 2000 equipped with a Pentium II 589 MHz processor and 384 MB RAM, and one machine for the proxy server running Linux 2.4.7-10 equipped with a Pentium II 451 MHz processor and 750 MB RAM. The client machines and the proxy server were on the same 100 Mbps LAN. The proxy server was connected to the Internet through our campus T3 connection. All test were run for 15 minutes, to allow each simulated client to request Web pages a reasonable number of times.

In order to simulate a typical 2G cellular connection (around 14.4 Kbps peak), we used traffic control mechanisms available for recent Linux kernels [5]. In all our tests, we specified 10 Kbps as cellular bandwidth for all clients. Here, we assume that a connection can be monitored in order to predict current bandwidth to a client. There are tools available for such tasks, such as [10]. We also assume round-trip delays to be zero.

[2] In our experiments, we assume all images are suitable for transcoding.

[3] Idle time is set to zero in all our experiments.

	Average queue latency per image		
clients	Minimum (s)	RR (s)	EDF (s)
40	0.96	1.0	1.0
80	1.92	2.0	2.0
120	2.9	3.2	3.2
160	3.84	4.3	4.3
200	4.8	5.4	5.2

Table 1. Non Scalable Selective Transcoding: A comparison of average queue latency per image under different scheduling policies.

	RR			EDF		
clients	Original (bytes)	Reduced (bytes)	Queue latency (s)	Original (bytes)	Reduced (bytes)	Queue latency (s)
40	58446	23167	1.0	58822	23370	1.0
80	59288	27477	1.5	58994	27155	1.4
120	59374	31877	1.9	59357	33014	1.6
160	59663	34754	2.2	59666	37028	1.8
200	60070	36626	2.5	59795	40268	1.9

Table 2. Scalable Selective Transcoding - Average Web Page Statistics and average queue latency per image under different scheduling policies.

In the experiments reported in this paper, we assumed the transcoding ratio to be 2 and the transcoding speed to be 1 Mbps. Note that the choice of these parameters affects the number of clients that can be efficiently served by a single machine, but has no effect on the relative performance of different scheduling optimizations as described in the next section.

4.2. Experimental Results

Before presenting the results of our experiments, we list the various cases that we have compared against each other.

- *No Transcoding (NT):* In this case, the proxy server acts as traditional HTTP request forwarder.[4] This will serve as the base case.
- *Round Robin Schedule, Non Scalable Selective Transcoding (RR-NSST):* In this case, we attempt to capture the behavior of a typical transcoding proxy, where transcoding is performed on a round-robin basis and the decision whether to transcode or not does not take client load into consideration (e.g., [7]).
- *Earliest Deadline First Scheduling, Non Scalable Selective Transcoding (EDF-NSST):* This is similar to RR-NSST except transcoding is done in order of increasing deadlines.
- *Round Robin Schedule, Scalable Selective Transcoding (RR-SST):* Here, we only transcode if we would not miss the deadline, as described earlier.
- *Earliest Deadline First Scheduling, Scalable Selective Transcoding (EDF-SST).*

Figure 2 shows the average latency for a web page under different cases, as we increase the number of clients (or load on the system). As expected, the average latency under the NT case remains almost constant, since in this mode the proxy server simply acts like a request forwarder. As the average size of a web page is

[4] However, compression of HTML pages and optimized HTTP/1.1 protocol were switched on in this and all other cases.

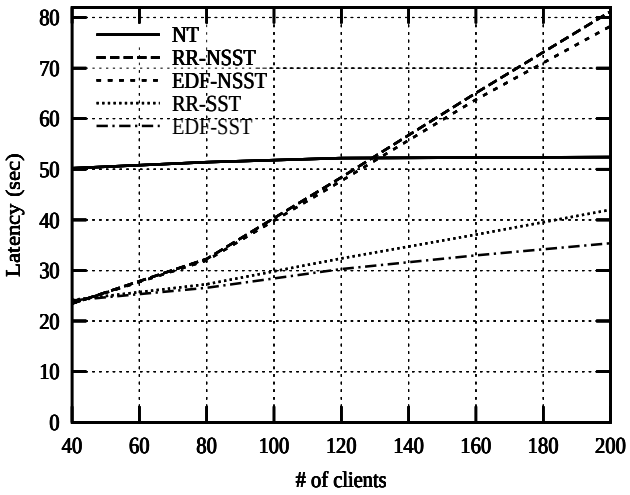

Fig. 2. Average Latency Per Web Page under different cases.

around 60K, the latency is constant at around 50s. RR-NSST and EDF-NSST perform almost identically, which can be explained as follows. If the average image size is B_I, then for N number of clients the average queue latency per image would *at least* be $\frac{NB_I}{T_S}$. However, this is only true if we always decide to transcode[5] and there is at least one image per client in the transcoder's queue. Table 1 shows the average queue latency per image for different scheduling policies. The values for RR-NSST and EDF-NSST are quite close to the minimum values and therefore both of them perform similar. This also explains the steep increase in the average latency per Web page. Since we are transcoding each image by a factor of 2 and the average image size is around 3K, the acceptable average queue latency would be around 1.3s, which is the time to transfer 1.5K of data over 10Kbps. However, the observed latency soon becomes significantly larger than this, and at around 130 clients RR-NSST and EDF-NSST become worse than a proxy with no image transcoding at all.

The results under RR-SST and EDF-SST demonstrate the impressive performance improvements under "scalable selective transcoding" as compared to "non scalable selective transcoding", i.e., the benefits of considering client load while making transcoding decisions. Also, it shows that an EDF scheduling policy provides further performance improvements over the RR scheduling policy.

In order to understand the difference in the performance under "Non Scalable Selective" and "Scalable Selective" transcoding, we need to look into how each of these strive to reduce latency as perceived by a client. While the former tries to reduce latency by sending a minimum amount of data over the network and in turn incurs huge queuing delays, the later does so by keeping an appropriate balance between the data sent over the network and the queuing delays in the transcoder. If we try to perform too much transcoding, the reduction in latency by sending less data over the network is largely offset by additional queuing delays.

Table 2 shows the average web page statistics under RR-SST and EDF-SST. As we increase the number of clients on the system, both RR-SST and EDF-SST try to minimize queuing delays per image, by selectively deciding about transcoding operations. In the process, both actually send more data over the network than RR-NSST and EDF-NSST. While both cases yield similar average latencies initially, as the load increases further the performance of RR-SST degrades compared to EDF-SST. Since under EDF-SST we deal with the clients in the order of increasing deadlines, the average queue latency per image is less.

[5]This is true in the case of "Non Scalable Selective Transcoding" because of the relatively large difference in bandwidth between the proxy-to-server and client-to-proxy links.

5. CONCLUSIONS

We presented results from a systematic performance study of an image transcoding proxy server. We have shown that seemingly intuitive transcoding policies that decide solely based on whether transcoding allows savings in transmission time fail to scale. The image transcoding performance can be improved by a factor of two in terms of average Web page latency as perceived by a client, by taking client load into consideration and by properly scheduling transcoding operations on a single CPU. We have also shown that an *Earliest Deadline First (EDF)* based scheduling policy further improves transcoding performance.

6. REFERENCES

[1] *ALL THINGS WEB.* Second state of the web survey (SOWS II). May 1998. http://www.pantos.org/atw/35654.html

[2] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakul. *An Active Transcoding Proxy to Support Mobile Web Access.* In Proc. of the IEEE Symp. on Reliable Distributed Systems, 1998.

[3] S. Chandra, A. Gehani, C. S. Ellis, and A. Vahdat. *Transcoding characteristics of web images.* In Proc. of the SPIE Multimedia Computing and Networking Conference, 2001.

[4] S. Chandra, and C. S. Ellis. *JPEG Compression Metric as a Quality Aware Image Transcoding.* In Proc. of the 2nd USENIX Symp. on Internet Technologies & Systems, 1999.

[5] S. Floyd and V. Jacobson. *Link-sharing and Resource Management Models for Packet Networks.* IEEE/ACM Transactions on Networking, Vol. 3 No. 4, 1995.

[6] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. *Adapting to Network and Client Variability Via on Demand Dynamic Distillation.* In Proc. of ASPLOS-VII, 1996.

[7] R. Han, P. Bhagwat, R. LaMaire, T.Mummert, V. Perret, and J. Rubas. *Dynamic Adaptation in an Image Transcoding Proxy For Mobile Web Browsing.* In IEEE Personal Communications Magazine, December 1998.

[8] B. C. Housel, G. Samaras, and D. B. Lindquist. *WebExpress: A Client/Intercept Based System For Optimizing Web Browsing in a Wireless Environment.* In Proc. 2nd Annual Int. Conf. On Mobile Computing and Networking, 1996.

[9] A. Savant. *SPAWN: a Scalable Proxy Architecture for web access over slow Wireless Networks.* MS Thesis, CIS Department, Polytechnic University, 2003.

[10] S. Seshan, M. Stemm, R. Katz. *SPAND: Shared Passive Network Performance Discovery.* In Proc. 1st Usenix Symp. on Internet Technology and Systems, 1997.

[11] *SPAWN Project Homepage.* Polytechnic University. `http://cis.poly.edu/spawn/`

[12] J. Smith, R. Mohan, C. Li. *Content-based Transcoding of Images in the Internet.* Proc. of the Int. Conf. on Image Processing, 1998.