Systematic Multiresolution and its Application to the World Wide Web

Swarup Acharya *Henry F. Korth* *Viswanath Poosala*

Information Sciences Research Center
Bell Laboratories, Lucent Technologies
Murray Hill, NJ 07974
{swarup,hfk,poosala}@research.bell-labs.com

Abstract

Many emerging environments are increasingly facing the problem where the requirements of applications easily outstrip the system resources. This is particularly acute in the World Wide Web (WWW) and many data-intensive applications like OLAP and multimedia databases. In this paper, we address this problem in the Web context via systematic multiresolution*, i.e., a framework for providing responses at different qualities (resolutions) and costs.*

We validate our conceptual contributions by implementing NetBlitz*, a multiresolution-based proxy server on the WWW.* NetBlitz *addresses two key problems facing the Web: high latencies and heterogeneity of client resources and requirements. It solves these problems by dynamically generating the "required version" of a web object based on client preferences and capabilities. We also propose novel multiresolution-aware caching techniques that further improve performance. Finally, we experimentally demonstrate the utility of multiresolution and the caching enhancements proposed.*

1. Introduction

The last few years have witnessed an explosive transformation in the computing and communication landscape. The immense growth of the World Wide Web (WWW) has created a global internetwork allowing users to connect to far away machines at a mouseclick. In parallel, there has been a similar upheaval in the world of applications. Multimedia databases, video and audio-based applications, and online analytical processing (OLAP) applications, which were earlier limited to powerful machines due to resource and computational constraints are becoming more and more commonplace. However, these changes have introduced an important performance problem: the volume of data processed and transferred has grown dramatically, overloading the servers and networks, resulting in large response times. This is clearly intolerable in such interactive applications. In spite of the tremendous growth in the hardware and software capabilities over the last few decades, the appetite for expensive applications seems to indicate that this trend of demand outstripping the system resources is unlikely to change for a while.

In this paper, we consider a novel approach to bridging the gap between the demand and the resources by offering information at different levels of quality (*resolutions*). This approach relies on two key observations:

- A low resolution result often entails lesser storage, processing, and transmission overheads. This can reduce the system load and improve the response time.
- In many applications, a *user* either does not benefit from the actual information in its full form or is willing to forego a loss in quality in exchange for faster responses[1].

We illustrate these observations with an example.

[1]We use the term *user* to refer to the person accessing (or viewing) an object, and the term *content provider (CP)* to refer to the person providing the object. The *client* refers to the user's machine and the *server* to the CP's machine.

Example 1.1 *A high quality (*1024×1024*,* 24*-bit color) raster image occupies* 3 *MB, whereas a* $256 \times 256, 8$*-bit version uses nearly* 48 *times less storage, and in several cases the latter image suffices as a reasonable representative of the original image. Thus, reducing the number of colors or the size creates lossy resolutions of the same image that have similar "semantic" content, but are much cheaper to store and transfer.*

In many cases resolution can be less than obvious. An *abstract* of a publication compared to the *full* document, or, the *approximate* answer to a query compared to the *exact* answer [13, 11]. However, in almost all cases, the lower quality version is potentially much cheaper to supply than the original version.

Often, multiple versions of the *same* quality are also useful if the limited capabilities of the user's system restrict them from accessing some of the versions, e.g., a user without a *postscript* viewer will require the document in a different format. We use the term *multiresolution* to refer to the ability of a system to provide multiple versions of the same object, which may or may not differ in their qualities. Also, we use the term *resolution* to mean "resolution or version" in those cases where the distinction does not matter for our purposes.

The WWW is an ideal environment for applying multiresolution and this is motivated next.

Multiresolution helps in addressing two important problems facing the WWW — high latencies and client-side heterogeneity — as described below.

1. **World Wide Wait:** With the advent of graphical web browsers, the internet has witnessed tremendous growth in usage over the last few years. Coupled with the popularity of large images and video, this has resulted in very high response times, or the so called *World Wide Wait*. This problem is particularly acute when users are connecting over slow-speed modem-based networks. In such a scenario, the user is likely to welcome an option to retrieve a lower quality version of an object very quickly.

2. **Heterogeneity:** WWW can be considered to be an extreme wide-area distributed system with vastly heterogeneous clients. This heterogeneity arises in several forms [9]: *network*, *hardware*, *software*, and *user preferences* (Table 1). It is clear that delivering the same content to all clients is not very effective. For example, sending a large, high quality image to a mobile hand-held PalmPilot is pointless; of course, delivering the same image to a graphics workstation on an ethernet is perfectly reasonable. Similarly, users without support for *pdf* documents on their systems may welcome an option to view the documents in plain text.

Resource	Typical Options
Network	
Bandwidth	28.8 kb .. 1 Mb
Latency	1s .. 10ms
Processor	
CPU Speed	100MHz .. 266 MHz
Memory	16Mb .. 512 Mb
Software	
Operating Systems	UNIX, Windows 95,98/NT, Macintosh
Document Formats	text, ps, pdf, html, ppt, MS Word
Video Formats	MPEG, Quicktime

Table 1. Differences in client-side resources

In summary, it is clear that the WWW can benefit tremendously from a multiresolution based approach, where users can receive different versions of the object based on their resources and preferences.

1.1. Systematic Multiresolution

The need for multiresolution in various forms has been realized by researchers working in several areas including graphics and visualization [8] and even on the WWW [9]. In the database arena, it has been proposed for approximating set data [19] and approximate query answering [11, 13, 15].

Unfortunately, the support for multiresolution in existing systems is either non-existent or ad-hoc at best. Even in research, most of the earlier work in the area has applied multiresolution in a somewhat narrow sense. For example, Fox *et al.* [9] developed a system to generate low resolution images and video dynamically on the Web, but did not study how to get better performance by extending the system to encompass other components, such as cache management. Also, most of the work so far has focussed on developing specific ways of generating resolutions for each data type, but does not identify the essential properties of multiresolution that are common to all data types. Hence, the extensibility and scope of these approaches is limited. In contrast, we have developed a general framework for applying multiresolution systematically to the entire application and have implemented such a system for the WWW. First, we describe the requirements and benefits of this approach, which we call *systematic multiresolution*.

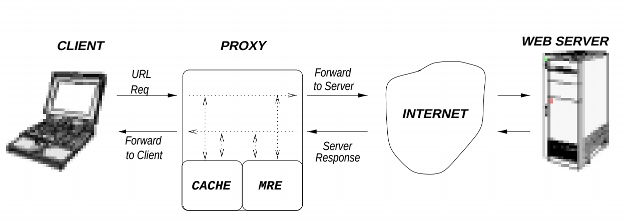


Figure 1. Proxy Architecture

We believe that a fully effective multiresolution system needs to address these issues:

- *Version Generation:* Obviously, it is necessary to design a multiresolution engine (MRE) that generates the versions automatically and efficiently when it is possible to do so, and supplies the desired version to the user.

- *System Redesign*: To utilize multiresolution effectively, many system and software components have to be re-designed to take multiresolution into account. For example, in the WWW context, by making the cache manager recognize that some versions can be derived from others, one can keep fewer versions of an object and thus, save space.

- *Interface:* Since an object may have multiple versions, a uniform user interface is required which creates a *single* point of access to all versions and describes the relationship among them.

In this paper, we present a multiresolution-based system for the WWW, called *NetBlitz*, which addresses each of the above issues. Our initial experience with the system has been extremely promising and consistently provides significant improvement in performance. The key features of our scheme include:

- A *proxy-based design* for dynamically generating versions. This design is backward compatible requiring no modifications to the browser, web server, or the HTTP protocol [17] (Section 2.1).

- A *multiresolution-aware* caching algorithm which offers higher hit rates and significantly lower latencies than regular caching schemes. This is probably the first web-based scheme that deals with interrelated objects (Section 2.2).

- A graphical user interface which allows communication of resolution information between the CP, user, and the system over a standard browser (Section 2.3).

We have conducted an extensive set of experiments which demonstrate that *NetBlitz* and our multiresolution-aware cache management offer significant performance benefits over standard approaches (Section 3).

Due to space constraints, in this paper we summarize the implementation details of *NetBlitz*. For further information on the implementation and on real-life usage experience with *NetBlitz*, the reader is referred to [2]. Additional details on the systematic multiresolution framework can be found in [1].

2. *NetBlitz* Multiresolution System

In this section, we describe the design and implementation of *NetBlitz*, a multiresolution system for the WWW. *NetBlitz* dynamically generates different versions and delivers them to the user. This system has three key components: *version support* provided by the multiresolution engine (MRE), *caching algorithm*, and *user interface*. These components are described next.

2.1. Version Support

The support for versions in a multiresolution system consists of two items: dynamic generation of *automatic versions* (i.e., versions that can be generated in software from the original object, with no manual intervention) and *the mechanism for supplying the versions (automatic as well as non-automatic, or, materialized versions, which are provided by the CP) to the user*. We use a *proxy-based* architecture to provide version support in our system. We describe the architecture below.

2.1.1 System Architecture

Figure 1 shows the high-level architecture of our proxy based system. Briefly, a web proxy is a process intervening between the web server and the client (browser) that forwards the browser's requests to the server and the server's responses to the browser. It is commonly used to cache web documents and to provide secure firewalls for a set of users in a large organization (e.g., in a company or by an internet service provider) [22]. In our system, we generate different versions of objects on-the-fly *in the proxy* as they are sent by the web server to the browser. We chose to run generate versions at the proxy rather than the server or the client, because a) it addresses the bandwidth limitation of the last hop to the user, which is often the bottleneck and b) it can be implemented without requiring changes to the hundreds of servers and clients. The essential features of our design also extend to a server-based system.

In our implementation, we modified the widely used Apache proxy server [3] to provide multiresolution. The basic operation of our proxy is described in Algorithm2.1.

Algorithm 2.1 Multiresolution-Proxy(URL)

```
begin
    Extract the URL ($U$) and the version ($V$) from client request. /* details in Section 2.3. */
.   if (A valid copy of $< U, V >$ is present in the cache) then
        Return the cache copy to the client
    else
    begin
        $X$ := URL of the original object from which $V$ can be derived
        if ($X$ is not in the cache) then
            Fetch $X$ from the web-server.
        Derive $V$ from $X$ using the derivation function corresponding to the data type of $U$[2].
        Modify the relevant HTTP reply headers as necessary (e.g., mime-type and content-length)
        Return $V$ to the user.
        If the object is cacheable, cache the $X$ and $V$ versions (Section 2.2).
    end
end
```

Next, we discuss the implications of our multiresolution system on latency reduction.

The latency of a web request consists of some or all of the following components: *NetLatency* - the time taken to transfer data from the proxy to the client, *ServLatency* - the time taken to fetch the object from the server to the proxy in case the object is not in the proxy cache, and *ConvLatency* - the time taken to derive the version inside the proxy. *NetLatency* is obtained by dividing the volume of data transferred by the proxy-to-client network bandwidth (*NetBW*). *ConvLatency* is obtained by dividing the size of the source version by the bandwidth (*ConvBW*) of the derivation algorithm (i.e., number of bytes that can be converted per unit time).

[2]We use the extension and the mime-type of the object to determine the data type. This approach is already being used by most web browsers and web servers [10].

A natural question to ask is whether converting to a lower version always improves the latency. After all, generating a lower resolution takes some time. However, it is faster to transmit because of its smaller size. This creates an inherent conversion-transmission tradeoff. A simple formulation of this tradeoff is as follows.

Let r be the expected relative reduction in size. Assuming, a linear relationship to the size, it can be easily shown that converting to a lower resolution improves latency if and only if, *ConvBW* $>$ *NetBW* / r.

In other words, a multiresolution system is ideal if the client has a low bandwidth connection to the proxy, i.e., low *NetBW*. Similarly, multiresolution is most effective for data types that achieve significant compression (i.e., high r) and allow for fast transformations to different resolutions (high *ConvBW*). These observations are also validated by the experiments in Section 3.

In the next section, we describe our multiresolution-aware caching algorithms which further enhances the performance benefits of our system.

2.2. Multiresolution Cache Algorithm

An interesting side effect of providing both multiresolution and caching in a proxy is the presence of several versions of the same object in the cache. This could waste cache space unnecessarily because some of the versions may be derivable from others. We solve this problem via two new techniques which can be applied to most regular caching algorithms. First, we describe the basic operation of a caching algorithm.

Nearly every caching algorithm uses a metric representing the *utility* of caching an object. Typically, this utility metric is initialized when the object is fetched the first time and thereafter rises if the object is accessed or, otherwise falls with time. The object with the least utility is usually evicted to make space for a new object. The difference between various caching algorithms is in their metrics for utility. For example, in the commonly used LRU algorithm [20], the utility of an object is proportional to the recency of its last usage. On the other hand, in the recently proposed GreedyDual-Size algorithm, the utility of an object also depends on its size [5]. In our system, since different resolutions can be of different sizes, the cost measure used by the caching algorithm should *also account for the size* of the object. Hence, GreedyDual-

Size seems to be a good choice in this environment. This observation is also validated by our experiments.

Another issue to be addressed in our system is the assignment of utilities to various versions. We considered the following choices for the cache-resident version (S) and the newly generated version (D).
The options for the cache-resident version S are:

- *UPGRADE:* Upgrade its utility (since the request for V caused a hit of S).
- *NO-UPGRADE:* Leave its utility unchanged (since the request was not directly for S).

The options for the derived version D are:

- *SAVE:* Store it in the cache and determine its utility by treating it as a newly fetched object.
- *IGNORE:* Do not cache the new object at all (since it can be derived from S).

The tradeoffs of the resulting four combinations are studied in Section 3.4.

2.3. Interface Design

Multiresolution raises a very important issue in user interface design: given that there can be multiple versions of an object, how does the user select a version to view? Note that there is an added constraint that the browser and the HTTP protocol can not be modified in any way for this purpose. In this section, we briefly describe the mechanism we developed.

In our design of *NetBlitz*, each data-type has a default low cost version which is sent to the browser by the proxy the first time the document is requested by the browser[3]. If the user prefers to view another version, she has to select the object (either by "clicking" on a *link* to that object, or on the object itself) and a menu driven interface is displayed to guide her. This menu is generated by a small Javascript program which is appended at the bottom of every *HTML* page accessed via the proxy. Once the user selects a version, the script creates a URL containing a self-sufficient reference to the version — original URL of the object with embedded version information. The browser sends this URL to the proxy and the proxy responds back as described in Algorithm 2.1.

[3]Users who register with *NetBlitz* can customize their default resolution for the various data types which can be different from the system defaults.

Example 2.1 *Figures 2(a) and 2(b) show the high and low resolutions of the Bell Laboratories Home Page (*`http://www.bell-labs.com`*) respectively. Figure 2(b) also shows the menu with which the user can choose an alternative resolution (in this case, based on image quality, the default being low quality). Also, while the quality difference between the two versions are marginal, the space savings on the images are significant ($88kb$ (left) vs. $31kb$ (right)).*

3. Experimental Study

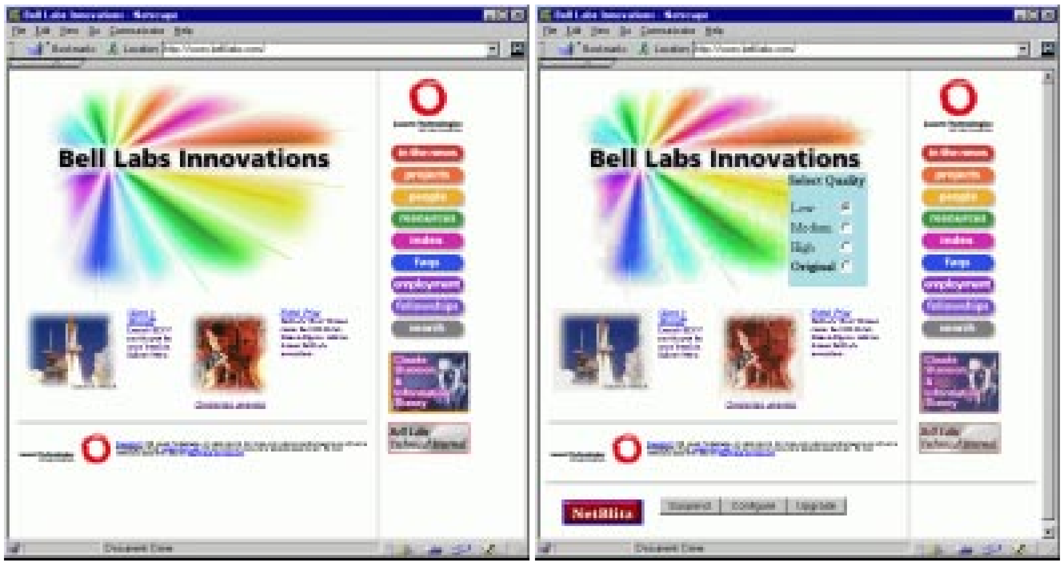
We conducted an extensive set of experiments to evaluate our multiresolution system. The two key elements studied in these experiments are a) *the performance gains due to multiresolution*, and, b) *the performance of various multiresolution-aware caching schemes*.

We modified the proxy simulator available from the University of Wisconsin [4] for most of our experiments. The simulation experiments were run using real-life web access data from the Digital Equipment Corporation's Web Proxy server [6]. These traces have been the basis of a number of earlier studies on proxy caching [21, 5, 7]. The experiments were run for 500,000 accesses. These requests were augmented to include a resolution field, which is described in the next section.

The various statistics for this trace are summarized in Table 2. The table shows that of all requests, only about half of them were for distinct objects. This translated to a maximum hit rate of 43% for the trace (obtained by having a large enough cache to hold all the objects). The size of all these distinct objects (*InfSize* $=$ 3.34 GB) was 62% of the total number of bytes requested in the trace (5.4 GB). Note that any cache size above *InfSize* would produce the same maximal hit rate.

3.1. Modeling Resolution

For each request in the web trace, we assign a *resolution* and *size* to the requested object. We assume all objects have R possible resolutions.



(a) Original Document (88KB) (b) Low Resolution (31KB)

Figure 2. Interface to choose resolutions (for images)

Number of Requests	500000
Unique Requests	281397
Maximal Hit Rate	0.437
Total Bytes	5.4 GBytes
Unique Bytes (*InfSize*)	3.34 GBytes

Table 2. Trace Statistics

We based the resolution distributions on a) the probability of accessing the lowest resolution (α) and b) skew in the Zipf distribution used to model the access frequencies of the remaining resolutions (θ). The parameters values used for the experiments are given in Table 3. Based on the values, three distributions were studied. In the *High Biased* (HB) case, the original document gets most of the requests while in *Low Biased* (LB) case accesses are biased towards the lowest resolution. The third distribution, *Uniform* (UN) accesses all resolutions with the same probability.

A Zipf distribution used used to derive the sizes of the resolutions. We studied two cases here. The first called ZIPF corresponds to an efficient compression algorithm. In this case, size of the data falls rapidly as we move from the original resolution to a lower resolution. In the other case, called ZINV, we “invert” the Zipf distribution, i.e., the sizes do not fall much except for the very low resolutions.

¿From the above discussion, it is clear that there are six combinations of the resolution-size distributions. However, ZIPF and ZINV provide similar performance for LB and HB distributions. Thus, we explore multi-resolution for four cases, namely, Uniform + Zipf (UN_ZIPF), High Biased + Zipf (HB_ZIPF), Low Biased + Zipf (LB_ZIPF), and Uniform + Zipf Inverse (UN_ZINV).

Intuitively, multiresolution would be most beneficial for the LB_ZIPF distribution. On the other hand, UN_ZINV would be expected to be the worst. Even though the UN_ZINV distribution is unlikely in real-life, we chose it to represent the adversarial case for multiresolution.

3.2. Cache Algorithms Studied

We extended several caching algorithms by applying the enhancements proposed in Section 2.2. However, since our goal is *not* to compare existing caching algorithms, we focus on only the GreedyDual-Size(GDS) recently proposed in [5]. The performance of other algorithms such as LRU are presented in [1]. GDS was chosen because it was shown in [4] to outperform other proposed algorithms for proxy caching. GDS is enhancement of LRU that tries to keep in the cache recently used objects that are a) expensive to get and b) use less cache space. This is achieved by associating a *utility* (H) equal to *cost/size* with each newly cached object, where *cost* is the cost of fetching the object and *size* is its size. When the cache overflows, the page with the lowest value of H, min_H is chosen for replacement and the H value of each resident object

Cache Size	1% (% *InfSize*)
NetBW	19.2, . . ., 400 Kbs
ConvBW	250 Kbs
Resolutions	
Num. Resolutions R	4
HB_ZIPF	$\alpha = 0.1, \theta = 5$
LB_ZIPF	$\alpha = 0.9, \theta = 5$
UN_ZIPF	$\alpha = 0.1, \theta = 0$
UN_ZINV	$\alpha = 0.1, \theta = 0$

Table 3. Parameter Settings

is reduced by min_H. When a page is accessed, its H value is restored to its original value. There are many choices for the *cost* function depending on the criteria being optimized. For example, setting it to a constant (e.g., 1) optimizes for *hit rate* or, setting it to the downloading latency minimizes average latency. The trace driven studies in [5] show that GDS variant that optimizes for hit rate, called GD-Size(1) in that paper, provides the best overall performance. We, thus, use GD-Size(1) for the studies in Section 3.4 and refer to it as GDS in the rest of the paper.

As pointed out in Section 2.2, there are four possible combinations to be studied: SAVE and IGNORE with UPGRADE and NO_UPGRADE. In this paper, we only study the UPGRADE case with the SAVE and IGNORE option.

3.3. Parameter Settings

Table 3 shows the various parameters settings for these experiments. Many of the settings here are self explanatory. The *NetBW* was varied from 19.2kbs (low-end modem) to 128kbs (ISDN) speeds to cover the bandwidth range of wired and wireless dial-in connections. We chose a *ConvBW* of 250 Kbs based on our proxy conversion logs. However, since it is strongly dependent on the processing power of the proxy, it can be expected to improve in the future.

In each of the experiments, the performance of the resolution proxy is compared to a standard proxy that does not provide resolution. We refer to this as the *Default* scheme in our discussion.

We use two primary metrics to evaluate various techniques — *hit rate* and *average latency*. Hit rate refers to the ratio of the accesses that were satisfied by the proxy cache to the total requests. Average latency refers to the difference between the time the request was made and the time when it was satisfied - averaged over the entire trace.

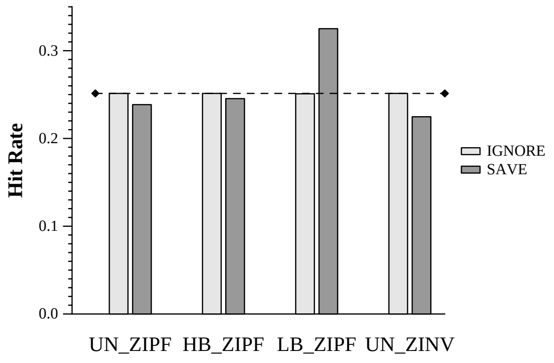


Figure 3. Comparing Hit Rate

3.4. Performance Results

In this section we present the results of a subset of our experiments. These experiments have been chosen because they are representative samples that highlight the inherent tradeoffs of multiresolution. An extensive quantitative study of other factors impacting multiresolution is available in [1]. First, we study the performance of the GDS algorithm in the multiresolution context, in terms of hit-rate and amount of data converted. Then, we study the effect of multiresolution on latency. All these experiments study the performance in the context of a multiresolution proxy. As pointed out earlier, the gains of performing multiresolution at the proxy are greatest when the client-proxy connection is the bottleneck. Thus, we finally study the benefits due to multiresolution if it were also done at the server. This addresses the case where the bandwidth bottleneck lies in the proxy-server path (i.e., the client has a fast connection to the proxy).

Expt. 1: Hit Rate Study
In this set of experiments, we compare the performance of LRU and GDS in combination with the SAVE and IGNORE options (Section 2.2). The cache size is 1% of *InfSize*, or about 33 MB.

Figure 3(a) shows the hit rate for GDS. The hit rate for the IGNORE heuristic is equal to the hit rate of the *default* case because it ejects the lower resolution and stores only the original document in the cache like a normal cache.

On the other hand, the SAVE heuristic gives a lower hit rate for the UNIFORM access patterns and the HB_ZIPF case because it retains the newly created version. Since the cache size is small (1%), saving a lower resolution wastes space since it is unlikely to be used in the near future. The gains in hit rate for LB_ZIPF, however, are impressive. This is due to two factors: a) the lower resolution requires much lesser space and b) GDS quickly evicts the original document since it has a larger size and it is rarely accessed. As the cache size increases, the gains from this strategy would be expected to diminish since cache space would no longer be a premium.

Expt. 2: Data Conversion Volume

The last experiment studied the cache hit rate in the multiresolution context. However, hit rate is only one of the many factors affecting performance in this environment. Another important metric is the volume of the data that the proxy has to convert to a lower resolution. This is shown in Figure 4 for cache size of 1% for the four distribution combinations.

The benefits of the SAVE heuristic are obvious here. For all the cases shown SAVE has to convert much fewer bytes. Since the volume of conversion has a direct influence on the load on the proxy, any savings here can provide significant performance improvement.

The HB_ZIPF distribution requires the least conversion since most of the requests are for the original document. The greatest gains from the SAVE heuristic are for LB_ZIPF — by saving the low resolution copy it can provide data to the client without any conversion. The larger the cache, more objects at lower resolution can be stored and thus, the gains from the SAVE heuristic improve for larger cache sizes.

For the two uniform cases, the cache is not large enough to store all the resolutions and consequently, the required resolutions are typically served using the original document, leading to the same amount of conversion.

This points to an interesting tradeoff — while more resolutions provide more flexibility to the client, they make cache management much harder. Thus, any gains to be made with having small-sized resolutions (as in UN_ZIPF) can get wiped out if there are too many actively requested resolutions.

Expt. 3: Latency Study

Finally, we study the performance metric directly relevant to the user: the average latency of requests. In this section, we limit our study to the LB_ZIPF and the UN_ZINV distributions for the SAVE heuristic due to space constraints. The tradeoffs for the IGNORE heuristic are similar.

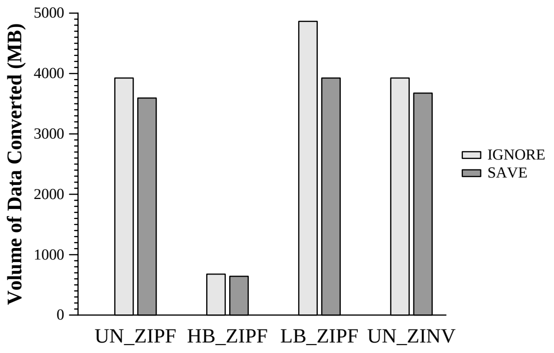


Figure 4. Data Converted by Proxy

Figure 5 shows the average latency of each request (y-axis) for different network speeds (x-axis). Note that the x-axis is in logscale. The two graphs are for GDS and a cache size of 1% and for a *ConvBW* of 250Kbps.

As expected, with increasing network bandwidth, the latency improves until it starts to flatten out. Recall that of the three components to latency, *NetLatency*, *ConvLatency* and *ServLatency*, the only variable for each line is *NetLatency*; the other two are independent of the network bandwidth[4]. Consequently, with increasing network speeds, the delay in transmitting pages from the proxy to the client becomes relatively insignificant causing the average latency to level off.

Figure 5 shows the tremendous performance gains due to multiresolution at the proxy. For the LB_ZIPF distribution, the improvement in latency is close to 50% for the modem speeds (19.2 - 56.6 Kbps). As the bandwidth increases, the client to proxy connection becomes less of a bottleneck, minimizing the gains. Yet, SAVE gets a significant improvement (20%) in response time for ISDN speeds (128 Kbps).

A surprising result is that of the UN_ZINV distribution. While its access pattern is unlikely to occur in practice, we chose it as an adversarial worst case for multiresolution. However, with the SAVE heuristic, it outperforms the default case, further making a case for providing multiresolution.

Expt. 4: Network Bandwidth Study

As the experiments so far have shown, doing multiresolution at the proxy can provide tremendous gains for environments with a weak to moderate proxy-client connection. However, in many settings such as intranets, clients have high-speed access to the proxy; the

[4]The next experiment studies the contribution of each of these components to the total latency.

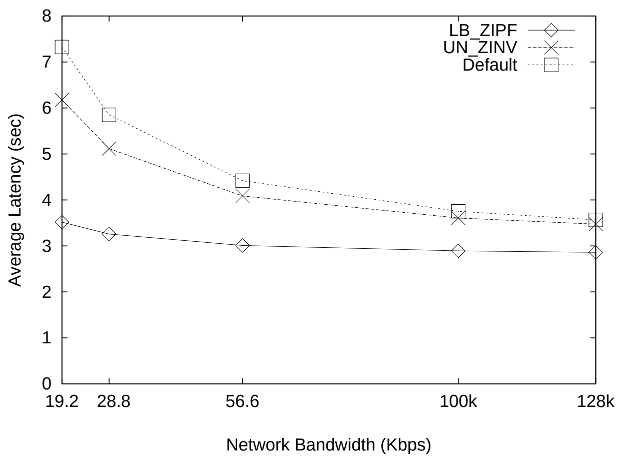


Figure 5. Average latency vs. Network BW

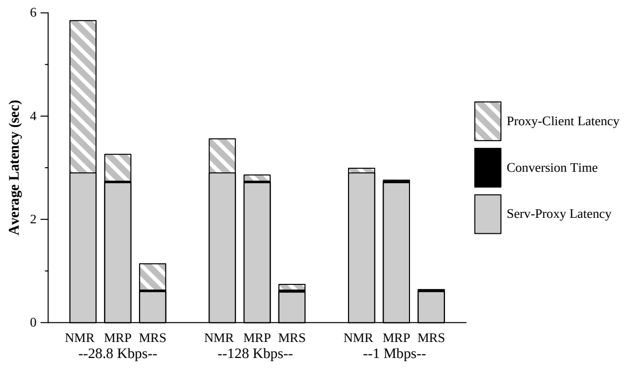


Figure 6. Latency components

bottleneck there is the connection from the proxy to the server. Doing multiresolution at the proxy can provide only marginal gains in such cases, instead multiresolution should be pushed all the way to the server. In this section, we study the performance in this scenario, namely doing multiresolution at the server (MRS) and compare it to the two cases measured earlier – multiresolution at the proxy (MRP) and the default case of no multiresolution (NMR).

Figure 6 studies the contribution of each of the three components of total latency – *NetLatency*, *ConvLatency* and *ServLatency* for three different bandwidth values of the client-proxy connection. Note that the NMR case has no conversion cost (*ConvLatency*).

As expected, the contribution of the *NetLatency* reduces as the client-proxy bandwidth increases. However, in all the cases, *ServLatency* is a significant portion of the total latency. Thus, by doing multiresolution at the server (MRS), one can additionally enhance performance significantly independent of the client-proxy bandwidth.

However, there are many non-technical issues that need to be addressed before it is feasible in practice to do multiresolution at the server. Some of these issues were highlighted briefly in Section 2.1.1. Consequently, in the current scenario, proxies provide the best scope of minimizing the data glut and the network bottleneck in the WWW.

4. Implementation Status and Experience

NetBlitz currently runs on a Sun Sparc Ultra-2 machine. It is a modification of the publicly available Apache [3] proxy server. Currently it supports various image formats and Word, Postscript and PDF document formats. *NetBlitz* has been put to wide use for over six months in the Information Sciences Research Center at Bell Labs and has served over half a million requests to date. *NetBlitz* allows users to customize the default resolutions for the various data types and tabulates the data usage for each client along with the bandwidth savings due to *NetBlitz* for the data types it converts. Further details on *NetBlitz*'s implementation can be found in [2].

5. Related Work

The proxy architecture was proposed in the original HTTP specification for providing secure firewalls [22] and has been extended to provide interesting features such as caching [5] and anonymizing user accesses [10]. The closest work to ours is the work by Fox *et al.* to provide *dynamic distillation* of images and video over the web [9]. The authors proposed a proxy-based scheme for dynamically transforming the contents of the data to lower resolutions. A similar approach for transcoding images in the mobile context has been taken in [12]. However, their work does not consider the implications of providing multiresolution on other system components such as caching and deals solely with versions that can be generated automatically. There is considerable interest in the web community on *content negotiation* [14]. Unlike our approach where the user is presented with the list of options, the focus there is to augment the HTTP protocol so that the client and the server can *automatically* negotiate the best (materialized) version for the client.

Web caching has been studied extensively in the literature ([21, 18, 5, 16] to name a few), though not in conjunction with multiresolution. A taxonomy of the various caching strategies proposed for the web is presented in [21] and [5]. As pointed out earlier, our goal is to incorporate multiresolution given *any* web caching algorithm and in that sense, our problem is orthogonal.

6. Conclusions

Many emerging environments are increasingly facing a problem where the requirements of applications easily outstrip the system capabilities. In this paper, we study a *systematic multiresolution-based* approach to addressing this issue. The goal of multiresolution is to improve system performance at the expense of systematically lowering the "quality" of the response at user's request. We propose a novel framework for providing multiresolution and highlight the broad problems that need to be addressed in designing such a system.

We then apply these concepts to the WWW and describe the design and implementation of a proxy-based prototype which automatically transforms objects to lower resolutions. By shrinking sizes of huge images and large documents on-the-fly, a multiresolution proxy provides a seamless solution to mitigate the problem of "world-wide-wait". The system also provides a uniform interface which allows the user to intuitively choose among the various versions of an object. Our experimental results show that extending traditional cache management to take into account multiresolution further enhances performance supporting our claim of the benefits of *systematic multiresolution*.

References

[1] S. Acharya, H. F. Korth, and V. Poosala. Systematic multiresolution and its application to the World Wide Web. Technical report, Bell Laboratories, April 1998.

[2] S. Acharya, H. F. Korth, V. Poosala, and S. Seshadri. NetBlitz: A Web content customization proxy. Technical report, Bell Laboratories, October 1998.

[3] Apache HTTP Server Project. WWW, URL: `http://www.apache.org`, February 1998.

[4] P. Cao. Web Cache Simulator. WWW, URL: `http://www.cs.wisc/edu/~cao/webcache-simulator.html`, Feb 1998.

[5] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. Technical Report CS-TR-97-1343, University of Wisconsin, Madison, May 1997.

[6] Digital Equipment Corporation, Digital's web proxy traces. WWW, URL: `ftp://ftp.digital.com/pub/DEC/traces/proxy`, Jan 1997.

[7] B. M. Duska, D. Marwood, and M. J. Feeley. The measured access characteristics of World-Wide-Web client proxy caches. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Dec. 1997.

[8] J. Foley, S. Feiner, J. Hughes, and A. vanDam. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990.

[9] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to network and client variability via on-demand dynamic distillation. In *Proceedings of ASPLOS*, pages 160–170, Cambridge, Massachusetts, Oct. 1–5, 1996.

[10] P. Gibbons, Y. Matias, and A. Mayer. How to make personalized web browsing simple, secure, and anonymous. In *Proceedings of Financial Cryptography*, February 1997.

[11] P. Gibbons, Y. Matias, and V. Poosala. The Aqua (Approximate Query Answering) Project White Paper. Technical report, Bell Labs, 1997. Contact poosala@research.bell-labs.com for a copy.

[12] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubias. Dynamic adaptaion in an image transcoding proxy for mobile web browsing. *IEEE Personal Communications*, December 1998.

[13] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. *Proc. of ACM SIGMOD Conf*, 1996.

[14] K. Holtman and A. Mutz. Transparent content negotiation in HTTP. WWW, URL: `http://www.ietf.org/ids.by.wg/http.html`, February 1998.

[15] Y. Ioannidis and V. Poosala. Histogram-based techniques for approximating set-valued querie s. Technical report, Bell Labs, 1998. Contact poosala@bell-labs.com for a copy.

[16] T. Kroeger, D. Long, and J. Mogul. Exploring bounds of web latency reduction from caching and prefetching. In *USENIX Symposium of Internet Technologies and Systems*, 1998.

[17] J. Mogul, T. Berners-Lee, L. Masinter, P. Leach, R. Fielding, H. Nielsen, and J. Gettys. Hypertext transfer protocol – HTTP/1.1. WWW, URL: `http://www.ietf.org/ids.by.wg/http.html`, November 1997.

[18] J. Pitkow and M. M. Recker. A simple yet robust caching algorithm based on dynamic access patterns. In *Proceedings of 2nd. International WWW Conference*, pages 1039–1046, October 1994.

[19] R. L. Read, D. S. Fussell, and A. Silberschatz. A multiresolution relational data model. In *Proceedings of the 18th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Vancouver*, Aug. 1992.

[20] A. Silberschatz, J. Peterson, and P. Galvin. *Operating System Concepts, 3rd Edition*. Adison Wesley Publishing Company, 1991.

[21] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for world-wide web documents. In *Proceedings of the ACM SIGCOMM*, pages 293–305, August 1996.

[22] N. J. Yeager and R. E. McGrath. *Web Server Technology*. Morgan Kaufmann, 1996.