Dynamic Processor Scheduling with Client Resources for Fast Multi-resolution WWW Image Browsing

Daniel Andresen, Tao Yang, David Watson, and Athanassios Poulakidas
Department of Computer Science
University of California
Santa Barbara, CA 93106
{dandrese, tyang, david, thanasis}@cs.ucsb.edu

Abstract

WWW-based Internet information service has grown enormously during the last few years, and major performance bottlenecks have been caused by WWW server and Internet bandwidth inadequacies. Utilizing client-site computing power and also multi-processor support at the server site can substantially improve the system response time. In this paper, we examine the use of scheduling techniques in monitoring and adapting to workload variation at client and server sites for supporting fast WWW image browsing. We provide both analytic and experimental results to identify the impact of system loads and network bandwidth on response times and demonstrate the effectiveness of our scheduling strategy.

1 Introduction

One of the fundamental roles for World-Wide Web (WWW) browsers and servers in today's environment is to provide a uniform interface for the on-line access of thousands of digitized documents, such as images. The main performance bottlenecks are server computing capability and Internet bandwidth. We examine these two issues under the scope of WWW-based digital library (DL) systems [12]. While multi-processor support for a server is critical for a popular WWW site [3], transferring part of the server's workload to the client is also possible since the current Web browsers have achieved the ability to download executable content. Taking advantage of multi-processor support with client resources can lead to significantly improved user interfaces and response times. An application of the server-client load shifting is in digital image browsing.

The current collections of the Alexandria Digital Library (ADL) project at UCSB [1] involve geographically-referenced materials, such as maps, satellite images, digitized aerial photographs, and associated metadata. The image size is large, from 10-100MB. In such a case, it is obviously impractical to send all the images matching a query in their entirety. The server must send low-resolution images for the user to further cull before high-resolution final versions are delivered. A technique for reducing the network bandwidth requirements in viewing high-resolution images is to let the client first browse low-resolution thumbnails, then reconstruct high-resolution images from the existing thumbnails with a small amount of additional data delivered from the server. This technique is called progressive image delivery and requires certain computing resources at client sites.

The main research challenge is the effective management and utilization of resources from multi-processor WWW servers and client-site machines. Blindly transferring load onto clients may not be advisable, since the byte-code performance of Java is usually 5-10 times slower than a client machine's potential. Also a number of commercial corporations are developing "network computers", with little or no hard drive, a minimal processor, but with Java and the Internet networking built in. A careful design of the scheduling strategy is needed to avoid imposing too much burden on these Net PCs.

In this paper we present a scheduling model for partitioning and mapping client-server computation based on dynamically changing server load and client-site computer capabilities. Our model incorporates multiple factors including disk, network, and computational abilities. We show how proper scheduling can lead to significantly improved response times over most current implementations in supporting WWW image browsing.

The paper is organized as follows: Section 2 gives the background on progressive image browsing. Section 3 presents the partitioning of wavelet-based image accessing operations. Section 4 discusses the scheduling strategy for a multi-processor WWW server and the cost modeling for a wavelet task. Section 5 gives an analysis of the response time performance for processing a fixed number of requests. Section 6 presents the experimental results. Section 7 discusses related work and conclusions.

2 Multi-resolution image browsing

We focus on browsing large digitized data objects, e.g. images in the ADL system. With current network speeds, it is quite infeasible to consider sending the full contents of an image file to users for the browsing purposes. The ADL has adopted progressive multi-resolution image delivery and subregion browsing as strategies to reduce Internet traffic when accessing map images [1]. This approach is based on the idea that users often browse large images via a thumbnail (coarse resolution), and desire to rapidly view higher-resolution versions and subregions of those images already being viewed. We briefly describe the techniques of wavelet image data retrieval and transformation for multi-resolution browsing.

Given an image, a forward wavelet transform produces a sub-sampled image of lower resolution called a "thumbnail", and three additional coefficient data sets. More formally, for the given quantized image I_1 of resolution $R \times R$[1], we specify the input and output of the forward wavelet transform as follows.

$$(I_2, C_1, C_2, C_3) = Forward\_Wavelet(I_1)$$

I_2 is the thumbnail of resolution $\frac{R}{2} \times \frac{R}{2}$, C_1, C_2 and C_3 are of resolution $\frac{R}{2} \times \frac{R}{2}$. Figure 1 depicts the result of wavelet transform.

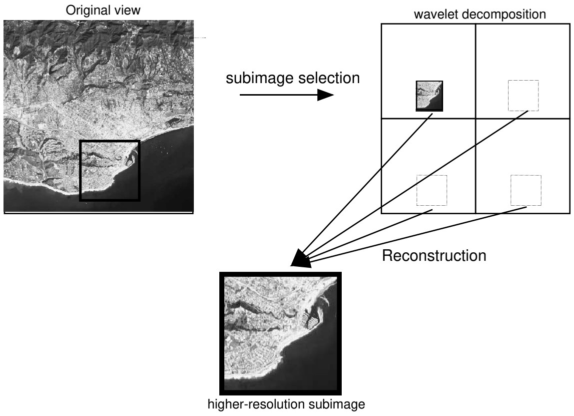

Figure 1: Reconstructing a high-resolution subregion from the thumbnail and coefficients.

The inverse wavelet transform (Inv_W) can be performed to re-construct the original image on-the-fly from the coefficient data sets and the thumbnail.

$$I_1 = Inv\_W(I_2, C_1, C_2, C_3).$$

If image thumbnail I_2 is available at the client site, then by requesting that the server sends C_1, C_2, C_3, image I_1 can be reconstructed at the client site. The image reconstruction is not time consuming, taking about 1.5 seconds for a 512×512 image on a SUN SPARC 5. The size of compressed data C_1, C_2, C_3 to be transferred is generally in the range of 10 to 100KBytes, which takes less than 1 second over a T1 link.

If a user wishes to access subregions of an image I_1, then the corresponding subregions in thumbnail I_2, C_1, C_2, C_3 can be retrieved and the reconstruction performed accordingly. We model such a process as follows.

$$\begin{aligned} subregion(I_1) = \quad & Inv\_W(subregion(I_2), subregion(C_1), \\ & subregion(C_2), subregion(C_3)). \end{aligned}$$

A detailed definition of forward and inverse wavelet functions can be found in [8]. The time complexity of wavelet transforms is proportional to the image size. The wavelet transform can be applied recursively, namely the thumbnail I_2 can be decomposed further to produce smaller thumbnails.

The actual wavelet implementation for data access and image reconstruction is a combination of Java and Computational Gateway Interface (CGI) programs [13]. Namely, an HTTP request activates a CGI program at the server, which computes the server's data using the predefined procedure. The results are sent to the client and then processed by the Java client-side application.

3 Task partitioning for progressive subregion image delivery

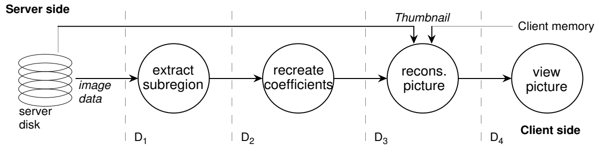

Figure 2: **A task chain for wavelet image enhancement.** Four cutoff points are depicted for possible computation partitioning between client and server. If the client performs the image reconstruction, the thumbnail is already available from the client memory and does need to be transmitted from the server.

The computation involved in multi-resolution image construction can be partially executed at a server and at a client also. Based on an implementation in [15], we model the computation and communication involved using a chain of subtasks depicted in Figure 2: 1) *Fetching compressed wavelet data and extracting the subregion.* The wavelet image data is stored in a combined quadtree/Huffman encoded form on a disk. These compressed files must be fetched. Then the appropriate subtree of a quadtree with its associated compressed coefficient data must be extracted in its compressed form. The compressed

[1]Rectangular shapes can also be supported while square images are used here for demonstration.

coefficient data is sent on to the next stage. 2) *Recreating the coefficients.* The compressed coefficients must be expanded to their original form. 3) *Reconstructing the pixels.* After the coefficients are available, the inverse wavelet function is called to create the new higher-resolution image from the thumbnail image. Notice that the thumbnail image needs to be fetched from the server disk if the reconstruction is conducted on the server. Otherwise, the thumbnail image is already available on the memory of the client machine. 4) *Viewing the image.* For our purposes, we assume the viewing of the image takes no computation time, and must be done on the client.

Figure 2 depicts the above processing steps and four possible cutoff points for partitioning this chain for the server and client. We discuss the possible computation and communication scenarios for four partitioning points below. Notice that we also need to consider that the data sent from the server to the client may be compressed first for transmission, then decompressed at the client site. **D1: The client starts from subregion extraction**. The entire compressed image data needs to be transferred, but the image thumbnail does not need to be transmitted. The transmitted wavelet data is not further compressible. **D2: The client starts from coefficient data recreation**. A part of compressed image data is retrieved on the server based on the subregion position. The image thumbnail does not need to be transmitted. The transmitted subimage data is not further compressible. **D3: The client starts with image reconstruction**. Coefficient reconstruction is conducted at the server site. But the derived subregion coefficient data must be further compressed otherwise the size of uncompressed coefficient data is similar to that of the original subregion image and it would be more efficient to send the original image. Thus the overhead of server compression and client decompression must be incorporated. The image thumbnail does not need to be transmitted. **D4: The client does not do any computation**. The image thumbnail needs to be retrieved from the server disk. The result of image reconstruction is not compressible further.

4 Processor scheduling for WWW requests and cost modeling for wavelet tasks

Our WWW server consists of a set of workstation nodes connected with a fast local network as shown in Figure 3 and is presented as a single logical server to the Internet. The user requests are first evenly routed to processors via DNS rotation [13]. The server nodes in the system communicate with each other and redirect requests to the proper node by actively monitoring the usages of CPU, I/O channels, and the interconnection network.

Our scheduling strategy for redirecting a request to an appropriate server node is to find a processor that minimizes the overall response time. In [3], we have proposed

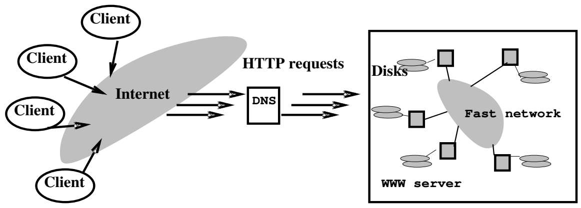

Figure 3: The architecture of a multi-processor WWW server.

a prediction model for approximating the processing time of a given request on a processor. In this model, we consider several factors that affect the response time. These include loads on CPU, disk, and network resources. The load of a processing unit must be monitored so that requests can be distributed to relatively lightly loaded processors. Disk channel usage must also be observed and simultaneous user requests accessing different disks can utilize parallel I/O to achieve higher throughput. The local interconnection network bandwidth affects the performance of file retrieval since many files may not reside on the local disk of a processor, so remote file retrieval through the network file system will be involved. Local network traffic congestion could dramatically slow the request processing.

For a wavelet task chain, we need to not only select an appropriate server node for processing, but also partition this chain into two parts. One part is executed on an appropriate server node, another part is executed on the client. The optimum partitioning point is the one which minimizes the overall processing time, which requires the dynamic consideration of client/server machine loads and capabilities, and the network bandwidth between client and server. We assume that the local communication delay between subtasks within the same machine (client or server) is zero while client-server communication delay is determined by the current available bandwidth and latency between them.

The cost for processing a wavelet chain is modeled as:

$$t_s = t_{redirection} + t_{data} + t_{server} + t_{net} + t_{client}.$$

$t_{redirection}$ is the cost to redirect the request to another processor, if required. t_{server} is the time for server computation required. t_{client} is the time for any client computation required. The values of t_{server} and t_{client} depend on how a task chain is partitioned. t_{data} is the server time to transfer the required data from the server disk drive, or from the remote disk if the file is not local. t_{net} is the cost for transferring the processing results over the Internet. We discuss t_{data} and t_{net} in more detail.

$$\bullet t_{data} = \begin{cases} t_{lstartup} + \frac{\text{data size}}{b_{disk} \times \delta_1} & \text{If local,} \\ t_{rstartup} + \frac{\text{data size}}{min(b_{disk} \times \delta_1, b_{lnet} \times \delta_2)} & \text{If remote.} \end{cases}$$

The requested file is the compressed image data. If image reconstruction is conducted at the server then the thumbnail image should also be included. If reconstruction is conducted at the client then the thumbnail image is already available and thus it is not included.

If the file is local, the time required to fetch the data is simply the file size divided by the available bandwidth of the local storage system, b_{disk}, plus some startup overhead $t_{lstartup}$. We also measure the disk channel load δ_1. If there are many concurrent requests, the disk transmission performance degrades accordingly. If the data is remote, then the file must be retrieved through the interconnection network. The local network bandwidth, b_{lnet}, and load δ_2 must be incorporated, plus the startup overhead t_{rstart}. Experimentally, we found on the Meiko approximately a 10% penalty for a remote NFS access, and on the SUN workstations connected by Sparc/Ethernet the cost increases by 50%-70%.

In our current implementation, $t_{lstartup}$ and $t_{rstartup}$ are neglected because if the files are large, these costs are relatively small and if the files are small, redirection $t_{redirection}$ and network overhead will dominate.

$$\bullet t_{net} = t_{nstart} + \frac{\text{client-server communication size}}{\text{Net bandwidth}}.$$

This term is used to estimate the time necessary to return the results back to the client over the network. The number of bytes required again depends on how the partitioning is conducted. If the server does image reconstruction, then the entire subregion image needs to be shipped. If the client only does image reconstruction, the server only needs to send compressed the coefficient data. t_{nstart} is the startup time for network connection and is ignored in the current setting for similar reasons to those given for t_{lstart} and t_{rstart}.

Given this cost prediction for an image browsing request, the system compares all server nodes and enumerate all possible partitioning choices for the corresponding chain, then selects a partitioning and a server node to reach the minimum response time.

5 An analysis for homogeneous client-server systems

It is difficult to analyze the performance of this scheme for general cases. We make a number of assumptions to examine the impact of system resources on the selection of partitioning points. While our scheme works for processing a sequence of requests, we study the response time in simultaneously processing a fixed number of requests. Our result reflects the scheduling performance for responding to a burst of requests, which occurs frequently in many WWW sites [5, 9]. In the following analysis, we assume that the system is homogeneous in the sense that all nodes have the same CPU speed and the initial load, and each node has a local disk with the same bandwidth. We assume that all clients are uniformly loaded with the same machine capabilities. We assume that all requests have the same type of image operations for reconstructing an $n \times n$ subimage. Each server node processor receives a uniform number of requests, and produces a stable throughput of information requested.

We define the following terms:

- R – requests received for the entire system.
- p – the number of server nodes.
- r – requests received per processor. $r = \frac{R}{p}$. Notice that we assume that the r requests arriving at each node after the division via DNS are uniform.
- A – the average overhead in preprocessing a request, and deciding a redirection.
- b_1 – the average bandwidth of local disk access.
- b_2 – the average bandwidth of remote disk access.
- c – the average probability of a processed request accessing a local disk.
- S – the average slowdown ratio of the client CPU compared to the server node. $S = \frac{^{CPU}\text{server speed}}{^{CPU}\text{client speed}}$.
- d – the average redirection probability.
- O – the average overhead of redirection.
- B – the average network bandwidth available between the server cluster and the client.
- H – the average processing time for each request.
- F_1 – the average size of the compressed wavelet data (quadtree and coefficients).
- k – the average fraction of F_1 actually needed for a subregion.
- g – the constant ratio for the server cost of sending disk files, e.g., gF_1 is the time for sending data F_1.
- F_2 – the size of the thumbnail. The image size is $n/2 \times n/2$.
- E_1 – the average server CPU time for extracting the subregion information.
- E_2 – the average server CPU time for creating the subregion coefficient data.
- E_3 – the average server CPU time for image reconstruction.
- E_c – the average server CPU time for coefficient data compression.
- E_d – the average server CPU time for coefficient data decompression.

Among the r requests arriving at each node, we assume the probability of accessing one of the server disks is equal to 1/p. Then $r * 1/p$ requests are accessing the local disk. Among those r requests, dr of them will be redirected to other nodes but dr requests will be redirected from other nodes to this node (we also assume that redirection is uniformly distributed because of the homogeneous system). Our experiments show that in such cases, the redirected requests tend to follow file locality. Thus the total number of requests processed at each processor after redirection is r requests per second. Among them, the total number of requests accessing the local disk is r/p from the original arrival tasks, plus an additional d redirected requests. Then the probability of accessing a local disk for those r requests is:

$$c = \frac{\frac{r}{p} + d * r}{r} = \frac{1}{p} + d.$$

Let H be the average response time for each request (the time from when the client launches a request to the time the client receives desired data). Then

$$H = t_{sys} + t_{data} + W_1 + W_2 + W_3 + W_4 + W_5 + t_{net}.$$

where t_{sys} is the overhead for possible redirection, HTTP connection and parsing. t_{data} is server time spent for reading the compressed data and thumbnail file if needed. W_1 is time spent for extracting a subregion. W_2 is time spent for recreating the wavelet coefficient data. W_3 is for the wavelet image reconstruction. W_4 is the overhead for compressing/uncompressing data transmitted between client and server. W_5 is the server time required to send data to the client. t_{net} is the network time for client-server data transmission.

For the case of accessing local disk, the disk bandwidth is shared by r requests with probability of c. For the case of accessing a remote disk, the network bandwidth is shared by r requests with probability of $(1-c)$. Thus we have

$$t_{data} = c * \frac{F}{\frac{b_1}{r}} + (1-c) * \frac{F}{\frac{b_2}{r}}$$

where $F = F_1 + F_2$ if the server does the reconstruction, or $F = F_1$ otherwise.

The computation cost is the non-overlapped CPU cycles for processing a request. Notice that CPU cycles are shared by r requests. $t_{sys} = (A + d(A+O))r$. And,

$$W_1 = \begin{cases} r * E_1 & \text{if server extracts a subregion,} \\ S * E_1 & \text{if client does it.} \end{cases}$$

$$W_2 = \begin{cases} r * E_2 & \text{if server recreates coefficient data,} \\ S * E_2 & \text{if client does it.} \end{cases}$$

$$W_3 = \begin{cases} r * E_3 & \text{if server does image reconstruction,} \\ S * E_3 & \text{if client does it.} \end{cases}$$

$$W_4 = \begin{cases} E_c * r + E_d * S & \text{if cutoff point is D3,} \\ 0 & \text{else.} \end{cases}$$

$W_5 = r * g * F_n$, and $t_{net} = F_n/B$ where

$$F_n = \begin{cases} F_1 & \text{if cutoff point is D1,} \\ k * F_1 & \text{if cutoff point is D2,} \\ k * F_1 & \text{if cutoff point is D3,} \\ n^2 & \text{if cutoff point is D4.} \end{cases}$$

There are 4 possible partitions and we mark the response time for these partitions as H_1, H_2, H_3 and H_4. We choose the one with the minimum processing time.

Partition D1:

$$\begin{aligned} H_1 = \; & c * \tfrac{F_1}{\frac{b_1}{r}} + (1-c) * \tfrac{F_1}{\frac{b_2}{r}} + (A + d(A+O))r \\ & + r * gF_1 + S * E_1 + S * E_2 + S * E_3 + F_1/B. \end{aligned}$$

Partition D2:

$$\begin{aligned} H_2 = \; & c * \tfrac{F_1}{\frac{b_1}{r}} + (1-c) * \tfrac{F_1}{\frac{b_2}{r}} + (A + d(A+O))r \\ & + r * (E_1 + gkF_1) + S * (E_2 + E_3) + k * F_1/B. \end{aligned}$$

Partition D3:

$$\begin{aligned} H_3 = \; & c * \tfrac{F_1}{\frac{b_1}{r}} + (1-c) * \tfrac{F_1}{\frac{b_2}{r}} + (A + d(A+O))r \\ & + r * (E_1 + E_2 + E_c + gkF_1) \\ & + S * (E_3 + E_d) + k * F_1/B. \end{aligned}$$

Partition D4:

$$\begin{aligned} H_4 = \; & c * \tfrac{F_1+F_2}{\frac{b_1}{r}} + (1-c) * \tfrac{F_1+F_2}{\frac{b_2}{r}} + (A + d(A+O))r \\ & + r * (E_1 + E_2 + E_3 + gn^2) + n^2/B. \end{aligned}$$

We can further determine the redirection ratio d for different partitions in order to minimize the response time. A detailed analysis can be found in [2]. Due to space restrictions, the results are summarized as follows:

- **Case a:** When $A + O \geq (F_1 + F_2)(\frac{1}{b_2} - \frac{1}{b_1})$, $d = 0$ for all H_1, H_2, H_3 and H_4.
- **Case b:** When $A + O < F_1(\frac{1}{b_2} - \frac{1}{b_1})$, $d = 1 - 1/p$ for all H_1, H_2, H_3 and H_4.
- **Case c:** When $F_1(\frac{1}{b_2} - \frac{1}{b_1}) \leq A + O < (F_1 + F_2)(\frac{1}{b_2} - \frac{1}{b_1})$, $d = 0$ for H_1, H_2 and H_3, and $1 - 1/p$ for H_4.

Then,

$$H = \min(H_1, H_2, H_3, H_4).$$

The above formula can help us to understand the partitioning selection. For example in Figure 4, we illustrate the modeled response times and their relationship to server load, client capabilities, and network bandwidth using the following parameters based on our experimental results: n=512, k = 0.25, R = 6, p = 6, E_1 = 1.4 sec, E_5 = 0.4 sec, E_6 = 2.6 sec, b_1 = 1100000 byte per second, b_2 = 1000000 bytes per second, A = 0.001 sec, O = 0.1 sec, g = 2.5×10^{-7} sec/byte, F_1 = 56000 bytes, $F_2 = (n/2)^2$, E_c = 0.9 sec, E_d= 0.9 sec. Figure 4 (a) plots the results for H_1, H_2, H_3, and H_4, when S is set to range from 0.5 to 3. For (b), B ranges from 10000 to 150000 bytes/sec while S is 2. From (a) and (b), we can see that if the server is very fast or the Internet communication is very fast, then D4 is the best partition and the server does everything. Otherwise, the best partition is at D2, and it is advisable to send over the compressed data for the client to process.

6 Experimental Results

We have implemented a prototype of our dynamic scheduling scheme with client resources on a Meiko CS-2 distributed memory parallel machine based on our previous SWEB work [3]. The Meiko CS-2 is essentially a workstation cluster with a fast network interconnect. Each node has a scalar processing unit (a 40Mhz SuperSparc chip) with 32MB of RAM running SUN Solaris 2.3. Our primary experimental testbed consists of six Meiko CS-2 nodes as our server, each of which is connected to a dedicated 1GB hard drive on which the test files reside. Disk service is available to all other nodes via NSF mounts. The client machines are loaded with our custom Java applet library implementing some of the basic operations, including wavelet reconstruction. To avoid Internet bandwidth fluctuations, clients are located within the campus network

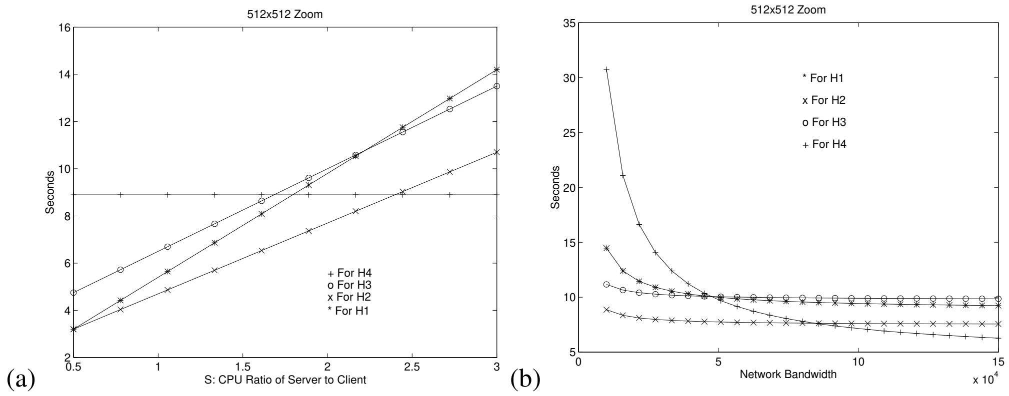

Figure 4: Impact of system resources on partitioning. (a) Client power varying, (b) Network bandwidth varying.

to assist in experimental stability over multiple runs. We primarily examine the performance of wavelet subimage retrieval. We select an operation which extracts a 512×512 subregion at full resolution from a $2K \times 2K$ map image, representing the user zooming in on a point of interest at a higher resolution after examining at an image thumbnail.

The overhead of our scheduling and load monitoring is quite small for all experiments. Analyzing a request takes about the range of 2-4ms, and load monitoring takes about 0.1% of CPU resources. These results are very similar to those reported in [3].

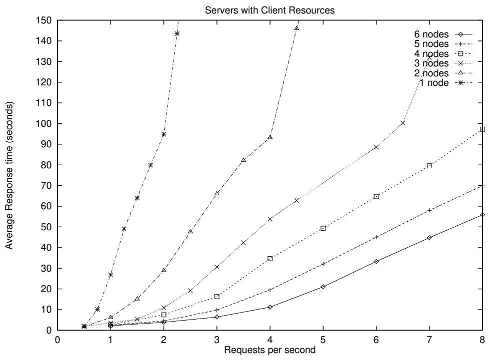

Figure 5: Request response times with client resources as the number of servers and RPS change.

Performance improvement using server and client resources. We examine the performance of the multi-processor server with client resources. We run a test for a period of 30 seconds, at each second R requests are launched from clients. Figure 5 shows the average response times for 1, 2, 3, 4, 5 and 6 server nodes with client computing resources. We can see that the response times drop significantly by using multiple server nodes. For example, with RPS=1, the average response time for the 6-node servers with client resources is less than 5 seconds while the time for the 1-node server is about 25 seconds.

With and without client resources. We also compare the improvement ratio of the response time with client resources over the response time without client resources in Table 1. As the server load increases steadily, the response time improvement ratio increases dramatically. We also note a significant increase in the number of requests per second (RPS) a server system can complete by using client resources. The detailed results are in [2].

	RPS=0.1	0.5	1	1.5	2
p=6	142 %	164%	192%	679%	4029%
p=5	96%	183%	191%	1838%	7893%

Table 1: Response time improvement ratio with and without client resource.

Load balancing with hotspots. "Hotspots" are a typical problem with DNS rotation, where a single server node exhibits a higher load than its peers [13, 14]. We examine how our system deals with hotspots. We directed a fixed number of requests to a subset of nodes in our server cluster, giving a wide range of load disparities. Without our scheduler, those nodes would have to process all of those requests. Our scheduling algorithm can effectively deal with temporary hotspots at various nodes by redirecting requests to other nodes in the group. The result is shown in Figure 6. The X axis shows the range of processors which receive requests. The upper curve of Figure 6 shows the average response time in seconds when no scheduling is performed and the request processing is limited to the fixed subset of nodes. The lower curve shows the average response time with presence of our scheduler. In that case the load is evenly distributed among all processors by redirection, and response times are very uniform.

Dynamic scheduling with varying client capabilities. Various clients will have differing capabilities in terms of network bandwidth and processing capabilities. We illustrate the effect of these on the scheduling decisions of our scheme and examine if the theoretical results presented in Section 5 match the system decisions under the specified assumptions. Figure 7 shows how the system makes the decision in terms of cutoff points for the subimage extrac-

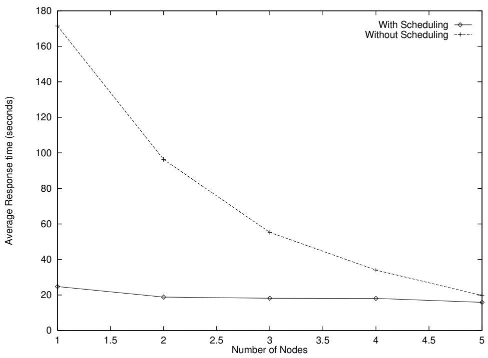

Figure 6: System performance with request concentration at fixed server subsets. Tests for a period of 30 sec., 4 RPS.

tion when processing R concurrent requests where R=18, and when artificially adjusting the server/client bandwidth and CPU ratio reported by the client. Each coordinate entry in Figure 7 is marked with the decision of scheduler. We overlaid the theoretical predictions from the formula of Section 5 on the system results. For each entry, if the choices for all requests agree with the theoretical prediction, we mark the actual selected cutoff decision, otherwise we mark the percentage of disagreement. For example, In Figure 7, when the bandwidth is 100,000 bytes/sec. and $S = 4$, the percentage of disagreement with the theoretical model is 6% among all processed requests. We can see the theoretical model closely matches the system's selections. As bandwidth decreases, the scheduler would increase the percentage of requests who make use of client CPU for data decompression and image reconstruction, minimizing the data sent over the network. On the other hand, as client CPU speeds decrease, we expect the server to do more processing.

Client-server bandwidth (bytes/sec)	Server/Client CPU Speed Ratio 0.25	0.5	1	2	4	10	20	40
100	D2	D2	D2	D2	D2	D2	D2	D2
500	D2	D2	D2	D2	D2	D2	D2	D2
1,000	D2	D2	D2	D2	D2	D2	D2	D2
10,000	D2	D2	D2	D2	D2	D2	**D4**	**D4**
100,000	D2	D2	D2	76	**6**	**D4**	**D4**	**D4**
150,000	D2	D2	D2	77	**D4**	**D4**	**D4**	**D4**
250,000	D2	D2	D2	93	**D4**	**D4**	**D4**	**D4**
500,000	D2	D2	D2	95	**D4**	**D4**	**D4**	**D4**
750,000	D2	D2	D2	88	**D4**	**D4**	**D4**	**D4**
1,000,000	D2	D2	D2	98	**D4**	**D4**	**D4**	**D4**
bytes/sec	***Predicted D2 region***				***Predicted D4 region***			

Figure 7: Effects of CPU speed and network bandwidth on decisions.

7 Related work and concluding remarks

We have presented a dynamic scheduling and cost model for processing image browsing requests by utilizing both client and WWW server resources. We have demonstrated the effectiveness of our scheme in adapting to different client-server capabilities for minimizing response times. Shifting computation from a server to its clients essentially scatters the workload around the world. This relates the global computing and application-level scheduling projects [6, 7, 10]. Those projects deal with an integration of different machines as one virtual machine. Our experience in using bandwidth and load information for scheduling could be useful to this research. Addressing client configuration variation is discussed in [11] for filtering multi-media data in order to reduce network bandwidth requirements but does not consider the use of client resources for integrated computing. Our current work is to generalize this work to support other applications on the WWW for adaptive server/client scheduling [2].

Acknowledgments

This work was supported in part by funding from NSF IRI94-11330, NSF CCR-9409695, NSF CDA-9529418 and a grant from NRaD. We would like to thank Omer Egecioglu, Oscar Ibarra, Terry Smith, Cong Fu, Norbert Strubel, and the ADL image processing team for many valuable discussions and suggestions.

References

[1] D.Andresen, L.Carver, R.Dolin, C.Fischer, J.Frew, M.Goodchild, O.Ibarra, R.Kothuri, M.Larsgaard, B.Manjunath, D.Nebert, J.Simpson, T.Smith, T.Yang, Q.Zheng, “The WWW Prototype of the Alexandria Digital Library”, *Proceedings of ISDL'95: International Symposium on Digital Libraries,* Japan August 22 - 25, 1995.

[2] D.Andresen, T.Yang, 'Adaptive Scheduling with Client Resources to Improve WWW Server Scalability', *Dept. of Computer Science Tech Rpt. TRCS96-27* , U.C. Santa Barbara, 1996.

[3] D.Andresen, T.Yang, V.Holmedahl, O.Ibarra, “SWEB: Towards a Scalable World Wide Web Server on Multicomputers”, *Proc. of 10th IEEE International Symp. on Parallel Processing (IPPS'96)*, April, 1996, Hawaii. pp. 850-856.

[4] D. Andresen, T. Yang, O. Egecioglu, O.H. Ibarra, T.R. Smith,“Scalability Issues for High Performance Digital Libraries on the World Wide Web”, Proc. of the 3rd Forum on Research and Tech. Advances in Digital Libraries (ADL96), pp. 139-148, May, 1996.

[5] M. Arlitt, C. Williamson, Web Server Workload Characterization: The Search for Invariants, Proc. SIGMETRICS Conference, Philadelphia, PA, May, 1996.

[6] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao, “Application-Level Scheduling on Distributed Heterogeneous Networks”, Proc. of Supercomputing '96, 1996.

[7] H. Casanova, J. Dongarra, NetSolve: A network server for solving computation science problems, Proc. of Supercomputing'96, ACM/IEEE, Nov., 1996.

[8] E.C.K. Chui, Wavelets: A Tutorial in Theory and Applications, Academic Press, 1992.

[9] M. Crovella, A. Bestavros, “Self-Similarity in World Wide Web Traffic Evidence and Possible Causes”, *Proc. SIGMETRICS96*, Philadelphia, May, 1996.

[10] K. Dincer, and G. C. Fox, Building a world-wide virtual machine based on Web and HPCC technologies. *Proc. of Supercomputing'96,* ACM/IEEE, November, 1996.

[11] A. Fox, E. Brewer, “Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation”, Computer Networks and ISDN Systems, Volume 28, issues 711, p. 1445. May, 1996.

[12] E. Fox, Akscyn, R., Furuta, R. and Leggett, J. (Eds), Special issue on digital libraries, *CACM*, April 1995.

[13] E.D. Katz, M. Butler, R. McGrath, A Scalable HTTP Server: the NCSA Prototype, *Computer Networks and ISDN Systems.* vol. 27, 1994, pp. 155-164.

[14] D. Mosedale, W. Foss, R. McCool, “Administering Very High Volume Internet Services”, *1995 LISA IX*, Monterey, CA, September, 1995.

[15] A. Poulakidas, A. Srinivasan, O. Egecioglu, O. Ibarra, and T. Yang, Experimental Studies on a Compact Storage Scheme for Wavelet-based Multiresolution Subregion Retrieval, Proceedings of NASA 1996 Combined Industry, Space and Earth Science Data Compression Workshop, Utah, April, 1996.