Adaptive Partitioning and Scheduling for Enhancing WWW Application Performance

Daniel Andresen, Tao Yang, Oscar H. Ibarra, Omer Egecioglu
Department of Computer Science
University of California
Santa Barbara, CA 93106
{dandrese, tyang, ibarra, omer}@cs.ucsb.edu

Abstract

This paper studies runtime partitioning, scheduling and load balancing techniques for improving performance of on-line WWW-based information systems such as digital libraries. The main performance bottlenecks of such a system are caused by the server computing capability and Internet bandwidth. Our observations and solutions are based on our experience with the Alexandria Digital Library (ADL) testbed at UCSB, which provides on-line browsing and processing of documents, digitized maps and other geo-spatially mapped data via WWW. A proper partitioning and scheduling of computation and communication in processing a user request on a multi-processor server and transferring some computation to client-site machines can reduce network traffic and substantially improve system response time. We propose a partitioning and scheduling mechanism that adapts to resource changes and optimizes resource utilization, and demonstrate the application of this mechanism for on-line information browsing. We also provide a performance analysis and experimental results to study impact of resource availability and the effectiveness of our scheduling techniques.

1 Motivations

The number of digital library (DL) projects is increasing rapidly at both the national and the international levels (see, for example, [3, 16]) and they are moving rapidly towards supporting on-line retrieval and processing of major collections of digitized documents over the Internet via the WWW. Performance and scalability issues are especially important for DLs. Many collection items have sizes in the gigabyte range while others require extensive processing to be of value in certain applications. Critical performance bottlenecks that must be overcome to assure adequate access over the Internet involve server processing capability and network bandwidth. Considering that popular WWW sites such as AltaVista already have several millions of requests a day, the server performance must scale to match expected demands. While we expect network communication technology to improve steadily, particularly with the advent of ATM and B-ISDN, we still need to consider the minimization of network traffic in the design of a WWW system.

Our research is motivated by the above situation and develops solutions addressing performance issues of WWW-based applications. In [4, 5], we have studied issues in developing multiprocessor WWW servers dealing with this bottleneck using networked workstations connected with inexpensive disks. As the WWW develops and Web browsers achieve the ability to download executable content (e.g. Java), it becomes logical to think of transferring part of the server's workload to clients. Changing the computation distribution between a client and a server may also alter communication patterns between them, possibly reducing network bandwidth requirements. Such a global computing style scatters the workload around the world and can lead to significantly improved user interfaces and response times. However, blindly transferring workload 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 so-called "network computers", with little or no hard drive, and a minimal processor, but with Java and Internet networking protocols built in. Carefully designed scheduling strategies are needed to avoid imposing too much burden on these clients. At the server site, information on the current system load and disk I/O bandwidth affects the selection of a server node for processing a request. In addition to this, the impact of available bandwidth between the server and a client needs to be incorporated. Thus dynamic scheduling strategies must be adaptive to variations of client/server resources in multiple aspects.

In this paper, we propose a model for characterizing computation and communication demands of WWW-based information access requests and investigate a partitioning and scheduling scheme to optimize the use of multiprocessors, parallel I/O, network bandwidths and client resources. The scheduling decision adapts to dynamically changing server and client capabilities. We present analytical results on homogeneous environments examining the impact of client and server resource availability. The paper is organized as follows: Section 2 gives our computational model and examples of client-server task partitioning and mapping. Section 3 discusses an adaptive partitioning and scheduling scheme for a multiprocessor WWW server with client resources. Section 4 analyzes the scheduling performance in a homogeneous environment. Section 5 presents experimental results and verifies our analytical results. Section 6 discusses related work and conclusions.

2 WWW request processing

We first discuss the background of the World Wide Web (WWW) and present a model for WWW request processing, then give two applications to demonstrate the use of this model.

2.1 The Model

The WWW is based on three critical components: the Uniform Resource Locator (URL), the HyperText Markup Language (HTML), and the HyperText Transfer Protocol (HTTP). The URL defines which resource the user wishes to access, the HTML language allows the information to be presented in a platform-independent but still well-formatted manner, and the HTTP protocol is the application-level mechanism for achieving the transfer of information [8, 9, 17]. An HTTP request would typically activate the following sequence of events from initiation to completion. First, the client determines the host name from the URL, and uses the local Domain Name System (DNS) server to determine its IP address. The local DNS may not know the IP address of the destination, and may need to contact the DNS system on the destination side to complete the name resolution. After receiving the IP address, the client then sets up a TCP/IP connection to a well-known port on the server where the HTTP server process is listening. The request is then passed in through the connection. After parsing the request, the server sends back a response code followed by the results of the query. This response code could indicate the request service can be performed, or might redirect the request to another server. After the contents of the request are sent to the client, the connection is closed by either the client or the server [17]. The results of the request are normally described by HTML, which the client displays on its local machine. The current client-side browser such as Netscape supports Java and the results of the request can be a platform-independent program (called applet) runnable at the client machine to produce results to be displayed.

Our WWW server model consists of a set of nodes connected with a fast network as shown in Figure 1 and presented as a single logical server to the Internet. User requests are first evenly routed to processors via DNS rotation [4, 18]. Each server node may have its local disk, which is accessible to other nodes via remote file service in the OS. 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.

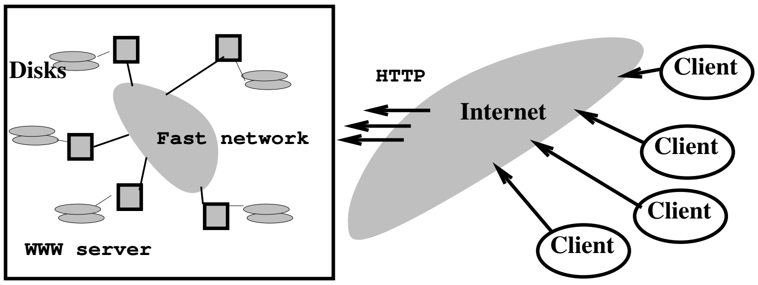


Figure 1: The architecture of a multiprocessor WWW server.

WWW applications such DLs involve extensive client-server interaction, and some of computation can be shifted to the client. In this paper we model the interaction between client and server using a task chain which is partially executed at the server (possibly as a CGI program [20]) and partially executed at the client (as a Java applet if applicable). A task consists of a segment of the request fulfillment, with its associated computation and communication. Task communication costs differ depending on whether task results must be sent over the Internet or can be transferred locally to the next task in the task chain. The following items appear in a task chain definition: 1) A task chain to be processed at the server and client site machines. A dependence edge between two tasks represents a producer-consumer relation with respect to some data items. 2) For each task, specify the input data edge from its predecessor, data items directly retrieved from the server disk and data items available at the client site memory. It should be noted that if a task is performed at a client machine, some data items may be available at this machine and slow communication from the server can be avoided. One such an example is wavelet-based image browsing to be discussed later.

Each task chain is scheduled onto one of the server nodes. Our challenge, then, is to select an appropriate node within the server cluster for processing, partitioning the tasks of the chain into two sets, one for the client and another for server, such that the overall request response time is minimized. This process is illustrated in Figure 2. In addition to considering the balancing between client and server machine load and capability, the network bandwidth between client and server affects the partitioning point. We assume that the local communication cost between tasks within the same partition (client or server) is zero while client-server communication delay is determined by the latency and current available bandwidth between them.

2.2 Text extraction from a Postscript file

We demonstrate the use of above model in Postscript document browsing. The extraction of the plain text from a Postscript-formatted document is an application which can benefit strongly from client resources, but requires dynamic scheduling. This application is useful for 1) *Content replication.* The archives of an Internet/intranet site would be a logical place to locate useful related work for inclusion in another publication. Two standard options are either scanning a hard copy of the document and using OCR, or

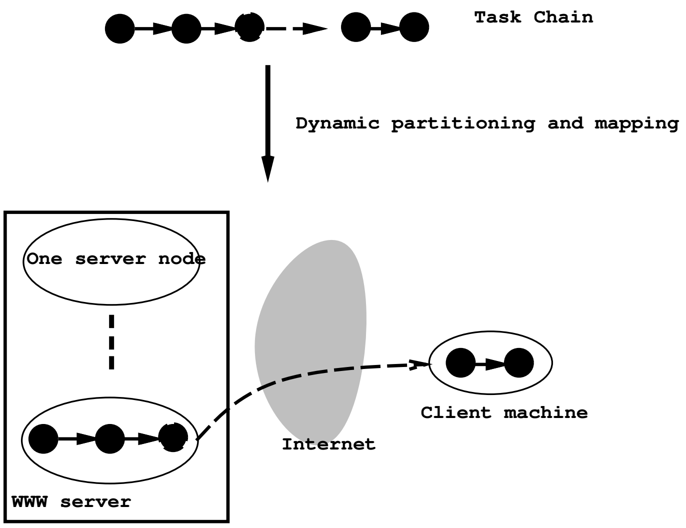


Figure 2: An illustration of dynamic task chain partitioning and mapping.

retyping the relevant sections. Automatic text extraction can be a much more efficient process. 2)*Non-Postscript-enabled browsers.* Many small WWW clients, such as personal digital assistants (PDAs) or NetPCs, do not have the capability to display Postscript files. Viewing the text can be the only available option to examine the content of a Postscript document [15].

Figure 3 depicts a task chain for processing a user request which extracts text from a subset of Postscript pages. The chain has two tasks that can be performed either at server or at client. 1) *Select the pages needed.* Eliminating unnecessary postscript pages reduces the computation needed for Step 2. The time to do this is small compared to the rendering time in the next task. 2) *Extracting the text.* A page is rendered via a software Postscript interpreter, and the text is extracted for the client. This step takes a significant amount of processing time. Thus there are three possible split points illustrated in Figure 3. We explain them as follows: D_1 - Send the entire Postscript file to the client and let it do everything. D_2 - Send over the relevant portions of the Postscript file for the client to process. D_3 - Extract the text on the server and send it over to the client for viewing.

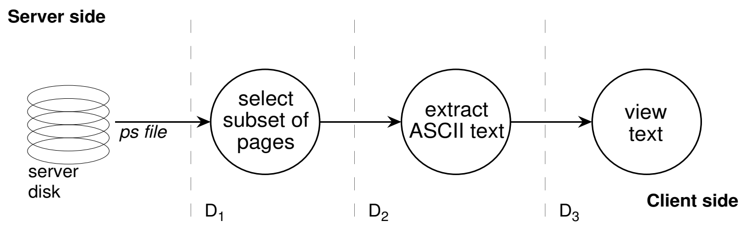


Figure 3: The task chain for text extraction from a Postscript file.

Dynamic scheduling is needed for balancing bandwidth and processing requirements. Postscript files are typically large, and so in most situations require a large amount of time to transmit. Text extraction dramatically reduces the size of the data to be transferred, but imposes a large computational burden. For example, extracting the text from a 25-page, 750KB technical paper takes about 50 seconds on a Sparc Ultra-1 workstation, with an output of about 70KB of text. Thus if the server does the processing, approximately 90% of the bandwidth requirements can be avoided, but this imposes a large amount of work on the server. The scheduler must determine a proper split point as a function of bandwidth and available processing capability.

2.3 Multi-resolution image browsing

Another application is browsing large digitized images in a DL 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. An image data file of size 100 MB will take about 8.5 minutes over a full T1 (1.544Mb/sec) connection. For the next generation of the Internet, e.g. T3 (45Mb/sec), TV set-top boxes (10Mb/sec), ATM and vBNS (155Mb/sec), the transmission time will significantly decrease but the demands for larger image files will continue increasing, especially when there are millions more users on the Internet. The ADL has adopted progressive multi-resolution image delivery and subregion browsing as strategies to reduce Internet traffic when accessing map images [3, 21]. 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 a 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 4 depicts the result of wavelet transform.

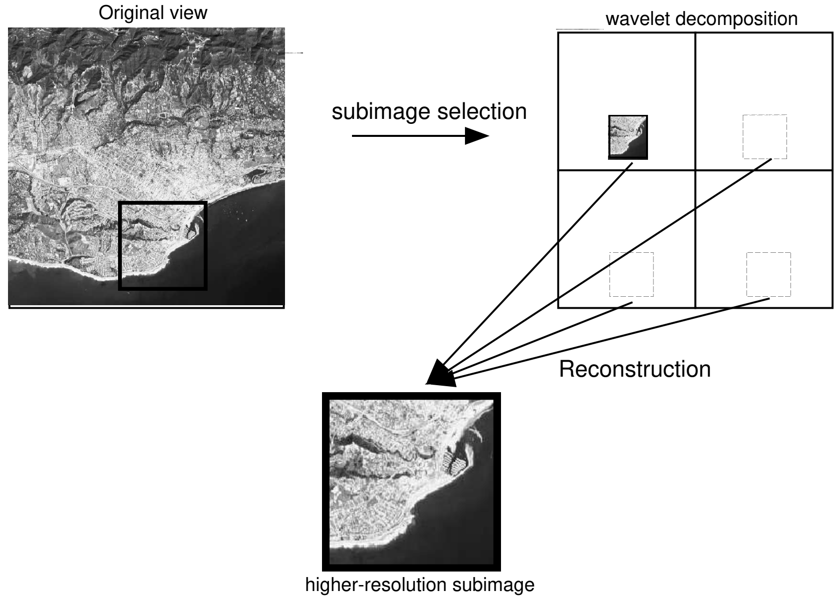


Figure 4: Reconstructing a high-resolution subregion from the thumbnail and coefficients.

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

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

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

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

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.

$$subregion(I_1) = Inverse\_Wavelet(subregion(I_2), subregion(C_1), subregion(C_2), subregion(C_3)).$$

A detailed definition of forward and inverse wavelet functions can be found in [11]. The time complexity of wavelet transforms is proportional to the image size. The wavelet transform can be applied recursively, namely thumbnail I_2 can be decomposed further to produce smaller thumbnails.

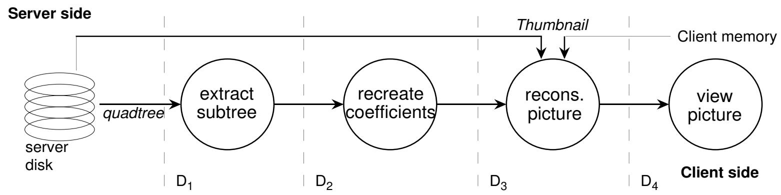


Figure 5: **A task chain for wavelet image enhancement.** Four cutoff points are depicted for possible computation partitioning between client and server. If the client performs the image reconstruction, the thumbnail is already available from the client memory and does need to be transmitted from the server.

The computation involved in multi-resolution image construction can be partially executed at a server and at a client also. The model of computation and communication described in [6, 21] uses the chain of tasks depicted in Figure 5: 1) *Fetching compressed wavelet data and extracting the subregion.* The wavelet image data is stored in a combined quad-tree/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 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 5 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 [21]. 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. D_1**: 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. D_2**: 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. D_3**: 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. D_4**: 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.

3 Partitioning and scheduling for request processing

Several factors affect response times for processing requests, including file locality, CPU/disk loads, and network resources. The load of each processing unit must be monitored so that requests can be assigned to relatively lightly loaded processors. Since data may need to be retrieved from disks, disk channel usages must be monitored. Simultaneous user requests accessing different disks can utilize parallel I/O channels to achieve a 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. We first present a cost model for predicting the response time in processing a request, then we discuss a strategy to select a server node and decide a good split point.

In our scheme request re-assignment is implemented using the HTTP "URL redirection" [4]. When client C sends a request to server S_0, S_0 returns a rewritten URL r' and a response code indicating the information is located at r'. C then follows r' to retrieve the resulting data. Most Net browsers and clients automatically query the new location, so redirection is virtually transparent to the user. This re-assignment approach requires a certain amount of time for reconnection. To avoid the dominance of such overhead, we include the redirection overhead in estimating the overall cost and a re-assignment is made only if the redirection overhead is smaller than the predicted time savings. For example, if a request involves a small file retrieval, typically no redirection occurs. DNS rotation is used to provide an initial load distribution [18]. Load re-balancing is effective under the assumption that our targeted WWW applications (e.g. DLs) involve intensive I/O and computation in the server and a study on the necessity of re-balancing after DNS rotation is in [4]. Another approach for implementing re-assignment is socket forwarding, which avoids the overhead of re-connection, but requires significant changes in the OS kernel or network interface drivers [2]. We have not used it for compatibility reasons.

3.1 A cost model for processing task chains

For each request, we predict the processing time and assign this request to an appropriate processor. Our cost model for a request is

$$t_s = t_{redirection} + t_{data} + t_{server} + t_{net} + t_{client}. \tag{1}$$

$t_{redirection}$ is the cost to redirect the request to another processor, if required. 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_{server} is the time for server computation required. t_{net} is the cost for transferring the processing results over the Internet. t_{client} is the time for any client computation required. We discuss the above terms as follows.

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

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. We currently ignore the startup costs for network disk I/O in our implementation, since if the request is large, other the transfer times dominate, and if the request is small, network overhead dominates.

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%.

- $t_{server} = CPU_{load} \frac{\text{No. of server operations required}}{{}^{CPU}\text{server speed}}$.

  The number of server operations required depends on how a task chain is partitioned. The cost estimation is based on the speed of the examined server node, the estimated CPU load on a destination node (CPU_{load}), and the estimated number of operations required. CPU_{load} represents the number of active jobs sharing the CPU, thus we multiply the required computation time by this factor to approximate the server CPU time. It should be noted that some estimated CPU cycles may overlap with network and disk time and the overall cost may be overestimated slightly, but this conservative estimation works well in our experience.

  The load estimation of remote processors is based on the periodic updating of information given by those remote processors. It is possible that a processor p_x is incorrectly believed to be lightly loaded by other processors, and many requests will be redirected to it. To avoid this unsynchronized overloading, we conservatively increase the CPU load of p_x by σ. This strategy is found to be effective in [22].

- $t_{client} = \frac{\text{No. of client operations required}}{{}^{CPU}\text{client speed}}$.

  The number of client operations required depends on how a task chain is partitioned. Here we assume the speed reported by the client machine includes client load factors.

- $t_{net} = t_{nstart} + \frac{\text{No. of bytes for client-server communication}}{\text{Net bandwidth}}$

  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. For the wavelet application, 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 the compressed 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}.

In the implementation, we need to collect three types of dynamic load information: CPU, disk, and network. CPU and disk activity can be derived from the Unix *rstat* utility, as well as some network information. A daemon (*loadd*) periodically updates the above information between the server nodes. Latency to the client can be approximated by the time required for the client to set up the TCP/IP connection over which the request (with the latency estimate) is passed. Client bandwidth is determined by the client, which measures the number of bytes per second received from the server for messages over a minimum size. The client passes both the latency and bandwidth estimates to the server as arguments to the HTTP request.

3.2 The procedure for dynamic chain partitioning and mapping

Given the arrival of HTTP request r at node x, the scheduler at processor x goes through the following steps:

1. **Preprocess a request.** The server parses the HTTP command, and expands the incomplete pathname. It also determines whether the requested document exists or it is a CGI program/task chain to execute. If it is not recognized as a task chain[2], the system will assign this request to the server node with the lowest load. Otherwise the following steps will be conducted.

2. **Analyze the request.** Given this task chain r, the system uses the algorithm in Figure 6 to select a partitioning and server node for the minimum response time. The complexity of this selection algorithm is $O(pd)$ where p is the number of server nodes and d is the number of all possible split points. No requests are allowed to be re-directed more than once, to avoid the ping-pong effect.

3. **Redirection and fulfillment.** If the chosen server node is not x, the request is redirected appropriately. Otherwise, a part of this chain is executed at this server node and the remaining part of a task chain will be further executed at the client machine.

```
Estimate the client CPU load and speed.
Determine the locality of server files to be accessed.
For each available server node
   Estimate the load, CPU speed, local disk available bandwidth and remote disk accessing bandwidth.
   For each possible client-server split point
      For each task executed on client, delete all server file input edges if those files are available in client memory.
      Size of server data = summation of all file input
      Size of client-server communication = the output data size of the last server task plus any server data needed for all client tasks.
      No. of server operations needed = summation of computation for all server tasks
      No. of client operations needed = summation of computation for all client tasks.
      Evaluate t_s using Equation (1) and its itemized definition.
   EndFor
   Select the best split point with the minimum t_s.
EndFor
Select the best server node with the minimum cost.
```

Figure 6: The procedure for scheduling a task chain.

4 A performance analysis

It is difficult to analyze the performance of our scheme for general cases. We make a number of assumptions to provide three facets of analysis: maximum sustained requests per second (MRPS), expected redirection ratios, and predicted task chain split points. In this way, we can analyze the impact of system resource availability from different aspects, for example, the number of server-nodes, available Internet bandwidth

[2] A simple file retrieval can also be considered as a task chain as discussed later.

and the CPU speed ratio between client and server machines. We first present our framework, then demonstrate its use for three sample task chains: text extraction, wavelet processing, and file fetches. We will present experimental evidence to support our analysis.

4.1 The framework

Main assumptions. We assume that the system is homogeneous in the sense that all nodes have the same CPU speed and 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. All requests are uniform, i.e. involving the same task chain. Each server node processor receives a uniform number of requests, and produces a stable throughput of information requested. In estimating the available local disk or remote disk accessing bandwidth, we assume a linear model such that the bandwidth is uniformly shared among processed tasks. We neglect items such as disk and network contention, as well as memory subsystem artifacts such as paging at this time.

Request activity models. We examine the performance assuming that each node receives r requests at each second for a period of L. We denote this as model (r, L). Two instances of this model are considered.

- $(r, 1)$. This reflects the system performance in responding to a burst in user requests, which occurs frequently in many WWW sites [7, 13]. All requests are assumed completed in the same length of time and each server processor is dealing with the same number of requests until all finish. All task chains are partitioned uniformly on the same edge.
- (r, ∞). We examine MRPS when $L = \infty$, which represents the maximum sustained performance when the system enters a steady state and receives a stable amount of requests for a long period of time. Thus all requests can be assumed to be completed in the same length of time and each server processor is dealing with the same number of requests. All task chains are partitioned uniformly on the same edge.

We define the following terms:

- p – the number of server nodes.
- R – the total number of requests received for all processors per second.
- 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 and a client.
- B_s – the maximum aggregated network bandwidth available from the server to all clients.

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:

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

Assume the uniform split point is known. We further define:

- H – the average response processing time for each request (the time from when the client launches a request to the time when the client receives the desired data).
- F_s – the average size of server disk files needed.
- F_n – the average size of server-client data needed to be transferred.
- Z – the average number of requests processed simultaneously at each processor. For example, $Z = r$ for model $(r, 1)$.
- E_{server} – the average total task time needed at the server site.
- E_{client} – the average total task time needed at the client site.

Then

$$H = c * \frac{F_s}{\frac{b_1}{Z}} + (1-c) * \frac{F_s}{\frac{b_2}{Z}} + (A + d(A+O))Z + E_{server}Z + F_n/B + S * E_{client}. \tag{2}$$

Notice that $B \leq \frac{B_s}{Z*p}$.

4.1.1 Expected redirection ratios for $(r, 1)$ and (r, ∞)

The scheduler minimizes the response time for each request. The expected redirection ratio can be derived by comparing the costs of performing the redirection to the expected benefits.

For H in Equation (2), we can rewrite the formula in the following format:

$$d * Z * (\frac{F_s}{b_1} - \frac{F_s}{b_2} + A + O) + \text{constants independent of } d.$$

We present two scenarios for d:

- $\frac{F_s}{b_1} - \frac{F_s}{b_2} + A + O < 0, i.e.\ \ A + O < F_s(\frac{1}{b_2} - \frac{1}{b_1})$. This corresponds to the case of a large file access with a slow local network. In this case, H is minimized with d at its maximum possible value. Note we have a constraint such that
$$1 - c = 1 - \frac{1}{p} - d \geq 0$$
then $d \leq 1 - 1/p$. Thus $d = 1 - 1/p$.

- $A + O \geq F_s(\frac{1}{b_2} - \frac{1}{b_1})$. This corresponds to the case where we have a fast interconnection network and can fetch a file remotely almost as quickly as a local access; or when the redirection overhead is too high. Then $d = 0$, which minimizes H.

4.1.2 MRPS for (r, ∞)

The MRPS is achieved when the entire server system enters a steady state. Then r requests uniformly arrive at each server node at each second and the throughput for each individual processor is also r. For this stage, let H_s be the part of response time spent at the server site for each request. Then each request will be processed at a server node for H_s seconds. Since r new requests come in at each second, the number of requests processed at each node simultaneously is $z = rH_s$.

Based on Equation (2), we have

$$H_s \geq c * \frac{F_s}{\frac{b_1}{rH_s}} + (1 - c) * \frac{F_s}{\frac{b_2}{rH_s}} + (A + d(A + O))rH_s + E_{server}rH_s$$

$$r \leq \frac{1}{c * \frac{F_s}{b_1} + (1 - c) * \frac{F_s}{b_2} + A + d(A + O) + E_{server}}$$

$$R \leq \frac{p}{c * \frac{F_s}{b_1} + (1 - c) * \frac{F_s}{b_2} + A + d(A + O) + E_{server}}.$$

We also notice that since the throughput of p server nodes is R, the total data output per second for the entire system is $R * F_n$. This is restricted by the available output bandwidth of the system. Namely $R * F_n \leq B_s$. Thus $R \leq \frac{B_s}{F_n}$. Using the above two conditions, we have a bound for the MRPS of the entire system as:

$$R^* = \min(\frac{p}{c * \frac{F_s}{b_1} + (1 - c) * \frac{F_s}{b_2} + A + d(A + O) + E_{server}}, \frac{B_s}{F_n}) \text{ where } c = d + 1/p.$$

Naturally the MRPS may be maximized by having the client do everything possible to minimize E_{server}. But such a strategy may increase F_n, which also limits the system throughput. With advent of improved Internet network technology, we anticipate that the client-server bandwidth will be improved steadily in the near future, and the MRPS will be achieved by choosing the split point which lets the client do as much as possible. In this manner the maximum possible amount of calculation is spread to the clients.

4.1.3 The expected split point for $(r, 1)$

For this case, $Z = r$, and the H formula in (2) is simplified as follows.

$$H = c * \frac{F_s}{\frac{b_1}{r}} + (1 - c) * \frac{F_s}{\frac{b_2}{r}} + (A + d(A + O))r + E_{server}r + F_n/B + S * E_{client}$$

With different split points, all four terms F_s, F_n, E_{server} and E_{client} may change, affecting the optimum choice. We select the split point minimizing H. Partitioning also affects the MRPS that can be reached. In this case a philosophical decision must be made regarding the goal of the server - minimizing individual response times or preserving server resources for future requests. In our current research we assume the former policy.

4.2 Case analysis

In the following subsections we present an analysis for three applications: file fetch, Postscript text extraction, and wavelet-based image access.

4.2.1 File fetches

We study a simple case, the file fetch, to demonstrate the use of the above framework. The task chain is shown in Figure 7 and has only two tasks, sending the file and viewing it. The computational cost for sending the data is $g * F$ where F is the file size to be fetched. The location of the two tasks is fixed, and the split point is fixed between these two tasks.

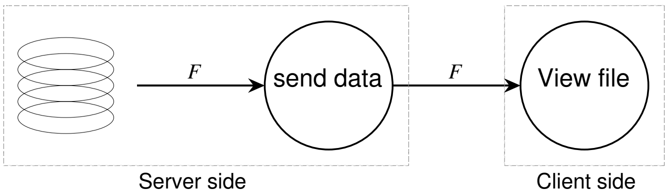


Figure 7: The task chain for a file fetch.

Expected Redirection Ratio for $(r, 1)$ and (r, ∞).

- If $A + O < F(\frac{1}{b_2} - \frac{1}{b_1})$, $d = 1 - 1/p$.
- If $A + O \geq F(\frac{1}{b_2} - \frac{1}{b_1})$, $d = 0$.

MRPS for (r, ∞). Then the upper bound for the sustained MRPS of the entire system at the steady state is:

$$R^* = \min\left(\frac{p}{(\frac{1}{p} + d)\frac{F}{b_1} + (1 - \frac{1}{p} - d)\frac{F}{min(b_1, b_2)} + A + d(A + O) + g * F}, \frac{B_s}{F}\right).$$

The expected split point. Fixed.

4.2.2 Postscript text extraction

Here we analyze the task chain illustrated in Figure 3. First we define some additional terms:

- F_1 – the average size of the Postscript file.
- k_1 – the average fraction of F_1 actually needed for the pages requested.

- k_2 – the average ratio of Postscript page size to its text. So $k_1 \times k_2 \times F_1$ is the size of the text extracted from the Postscript file.
- g – the constant ratio for the cost of sending disk files, e.g., gF_1 is the time for sending data F_1.
- E_1 – the average server CPU time for extracting the pages requested from a Postscript file.
- E_2 – the average server CPU time for extracting the text from the selected pages.

There are three possible partitions shown above and we mark the response time for these partitions as H_1, H_2, and H_3. We choose the one with the minimum processing time, i.e.,

$$H = \min(H_1, H_2, H_3). \tag{3}$$

Partition D_1:

$$F_s = F_1,\ F_n = F_1,\ E_{server} = gF_1,\quad E_{client} = E_1 + E_2.$$

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

Partition D_2:

$$F_s = F_1,\ F_n = k_1 * F_1,\ E_{server} = E_1 + gk_1F_1,\quad E_{client} = E_2.$$

$$H_2 = c * \frac{F_1}{\frac{b_1}{r}} + (1-c) * \frac{F_1}{\frac{b_2}{r}} + (A + d(A+O))r + r * (E_1 + gk_1F_1) + S * E_2 + k_1 * F_1/B.$$

Partition D_3:

$$F_s = F_1,\ F_n = k_1 * k_2 * F_1,\ E_{server} = E_1 + E_2 + gk_1k_2F_1,\quad E_{client} = 0.$$

$$H_3 = c * \frac{F_1}{\frac{b_1}{r}} + (1-c) * \frac{F_1}{\frac{b_2}{r}} + (A + d(A+O))r + r * (E_1 + E_2 + gk_1k_2F_1) + k_1 * k_2 * F_1/B.$$

Expected Redirection Ratio for $(r, 1)$ and (r, ∞). We determine the redirection ratio d for different partitions in order to minimize the response time.

For H_1, H_2, and H_3, $F_s = F_1$. According to our analysis in Section 4.1.1, based on the difference between $A + O$, $F_1(\frac{1}{b_2} - \frac{1}{b_1})$. d can be selected as either 0 or $1 - 1/p$. We have two cases:

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

The primary selection criteria between H_1 and H_2 will be the bandwidth between the client and server. Between H_2 and H_3, bandwidth is still important, but S becomes very significant.

MRPS for (r, ∞). The expected MRPS depends on the three choices of partitioning. We calculate the MRPS bound R_i^* for each partitioning and choose the maximum one as the bound.

For D_1,

$$R_1^* = \min\left(\frac{p}{c * \frac{F_1}{b_1} + (1-c) * \frac{F_1}{b_2} + A + d(A+O) + gF_1}, \frac{B_s}{F_1}\right).$$

For D_2,

$$R_2^* = \min(\frac{p}{c * \frac{F_1}{b_1} + (1-c) * \frac{F_1}{b_2} + A + d(A+O) + E_1 + gk_1F_1}, \frac{B_s}{k_1F_1}).$$

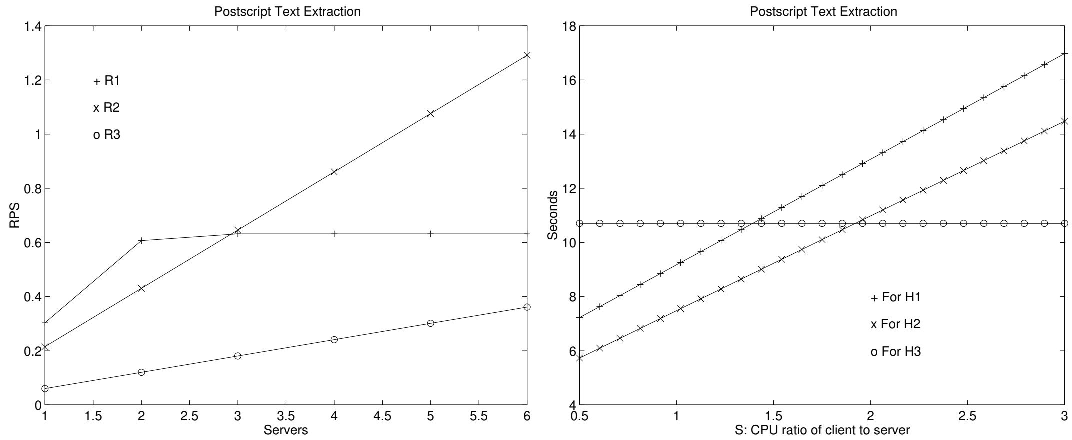
For D_3,

$$R_3^* = \min(\frac{p}{c * \frac{F_1}{b_1} + (1-c) * \frac{F_1}{b_2} + A + d(A+O) + E_1 + E_2 + gk_1k_2F_1}, \frac{B_s}{k_1k_2F_1}).$$

So

$$R^* = \max(R_1^*, R_2^*, R_3^*).$$

Using the above formulae, we can illustrate the impact of server load, client capabilities, and network bandwidth. The following parameters are used: $R = 12$, $E_1 = 0.4$ sec, $E_2 = 3.5$ sec, $b_1 = 1100000$ byte per second, $b_2 = 1000000$ bytes per second, $B_s = 1000000$ bytes per second, $g = 2.5 \times 10^{-7}$, $k_1 = 0.1$, $k_2 = 0.1$, $A = 0.001$ sec, $O = 0.1$ sec, and $F_1 = 1.6$MB. The values of R_1^*, R_2^*, R_3^* are depicted in Figure 8(a) when $S = 1$ and p varies from 1 to 6. Notice that R^* is the maximum among those three values. When p is small ($p \leq 3$), R_1^* leads to the highest R^* value because in this case the server has the minimum workload. When p is getting larger, R_2^* begins to play a role because bandwidth B_s limits the R_1^* value while for R_2^*, B_s is not a limiting factor. As a result, the value of R^* scales up as p increases.



(a) (b)

Figure 8: (a) Values of R_1^*, R_2^*, and R_3^* when $S = 1$ and p varies. (b) Values of H_1, H_2, and H_3 when $p = 6$ and S varies.

Expected Partition Points for (r, ∞)**.** The formula in Equation (3) helps us to compare and determine partitioning points in different scenarios. Using the same parameters, Figure 8(b) shows results of H_1, H_2 and H_3 when $p = 6$ and S is set to range from 0.5 to 3. For the minimum response time, the best split point is D_2 when S is small, and is D_3 when S is larger than 1.9, which indicates that with a slow client, it is better to retain the workload on the server. Similarly we can show that with a very slow Internet connection, D_3 is the optimum choice while D_2 is the best choice if B is reasonably large.

4.2.3 Wavelet chain processing

We now analyze the wavelet task chain illustrated in Figure 5 for retrieval of a subregion with size $n \times n$. First we define some additional terms:

- 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 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 subtree 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.

We mark the response time for the four partitions as H_1, H_2, H_3 and H_4. We choose the one with the minimum processing time, i.e.,

$$H = \min(H_1, H_2, H_3, H_4). \tag{4}$$

Partition D_1:

$$F_s = F_1,\ F_n = F_1,\ E_{server} = gF_1,\quad E_{client} = E_1 + E_2 + E_3.$$

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

Partition D_2:

$$F_s = F_1,\ F_n = k * F_1,\ E_{server} = E_1 + gkF_1,\quad E_{client} = E_2 + E_3.$$

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

Partition D_3:

$$F_s = F_1,\ F_n = k * F_1,\ E_{server} = E_1 + E_2 + E_c + gkF_1,\quad E_{client} = E_d + E_3.$$

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

Partition D_4:

$$F_s = F_1 + F_2,\ F_n = n^2,\ E_{server} = E_1 + E_2 + E_3 + gn^2,\quad E_{client} = 0.$$

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

Expected Redirection Ratio for $(r, 1)$ and (r, ∞). We determine the redirection ratio d for different partitions in order to minimize the response time.

For H_1, H_2, and H_3, $F_s = F_1$. For H_4, $F_s = F_1 + F_2$. According to our analysis in Section 4.1.1, based on the difference between $A + O$, $F_1(\frac{1}{b_2} - \frac{1}{b_1})$, and $(F_1 + F_2)(\frac{1}{b_2} - \frac{1}{b_1})$, d can be selected as either 0 or $1 - 1/p$. We have three cases:

- **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 $F_1(\frac{1}{b_2}-\frac{1}{b_1}) \le A+O < (F_1+F_2)(\frac{1}{b_2}-\frac{1}{b_1})$, $d=0$ for H_1, H_2 and H_3, and $1-1/p$ for H_4.

MRPS for (r,∞)**.** The expected MRPS depends on the four choices of partitioning. We calculate the MRPS bound R_i^* for each partitioning and choose the maximum one as the bound.

For D_1,

$$R_1^* = \min(\frac{p}{c * \frac{F_1}{b_1} + (1-c) * \frac{F_1}{b_2} + A + d(A+O) + gF_1}, \frac{B_s}{F_1}).$$

For D_2,

$$R_2^* = \min(\frac{p}{c * \frac{F_1}{b_1} + (1-c) * \frac{F_1}{b_2} + A + d(A+O) + E_1 + gkF_1}, \frac{B_s}{kF_1}).$$

For D_3,

$$R_3^* = \min(\frac{p}{c * \frac{F_1}{b_1} + (1-c) * \frac{F_1}{b_2} + A + d(A+O) + E_1 + E_2 + E_c + gkF_1}, \frac{B_s}{kF_1}).$$

For D_4,

$$R_4^* = \min(\frac{p}{c * \frac{F_1+F_2}{b_1} + (1-c) * \frac{F_1+F_2}{b_2} + A + d(A+O) + E_1 + E_2 + E_3 + gn^2}, \frac{B_s}{n^2}).$$

We assume that $n^2 >> k * F_1$ (the size of the original image is normally larger than that of compressed wavelet data). In comparing R_1^*, R_2^* R_3^* and R_4^*, we need to consider that d is different for Cases (a), (b) and (c). For Cases (a) and (b), d is chosen to be the same among all partitions, thus it is easy to show that $R_4^* \le R_2^*$ and $R_3^* \le R_2^*$.

For Case (c), d is the same for D_2 and D_3. Thus $R_3^* \le R_2^*$. Between R_4^* and R_2^*, we can plug in d values, and we have.

$$R_2^* = \min(\frac{p}{1/p * \frac{F_1}{b_1} + (1-1/p) * \frac{F_1}{b_2} + A + E_1 + gkF_1}, \frac{B_s}{kF_1}).$$

$$R_4^* = \min(\frac{p}{\frac{F_1+F_2}{b_1} + +A + (1-1/p)(A+O) + E_1 + E_2 + E_3 + gn^2}, \frac{B_s}{n^2}).$$

Since, $F_1(\frac{1}{b_2}-\frac{1}{b_1}) \le A+O$ in Case (c),

$$\frac{F_1+F_2}{b_1} + (1-1/p)(A+O) > \frac{F_1}{b_1} + (1-1/p)F_1(\frac{1}{b_2}-\frac{1}{b_1}) = 1/p * \frac{F_1}{b_1} + (1-1/p) * \frac{F_1}{b_2}.$$

Thus $R_4^* < R_2^*$ for Case (c).

In summary, only partition D_1 or D_2 could lead to the highest MRPS for all three cases. Then,

$$R^* = \max(R_1^*, R_2^*).$$

Expected Partition Points for (r,∞)**.** Formula (4) can help us to determine the partitioning points for different scenarios. For example, we can compare H_1, H_2, H_3, H_4 as follows.

For H_1, H_2, and H_3, d is always the same. Notice that $r = R/p$. Thus we have

$$H_2 - H_1 = (R/p - S)E_4 - (1-k)F_1(\frac{1}{B} + g)$$

$$H_3 - H_2 = (R/p - S)E_5 + E_c * R/p + E_d * S.$$

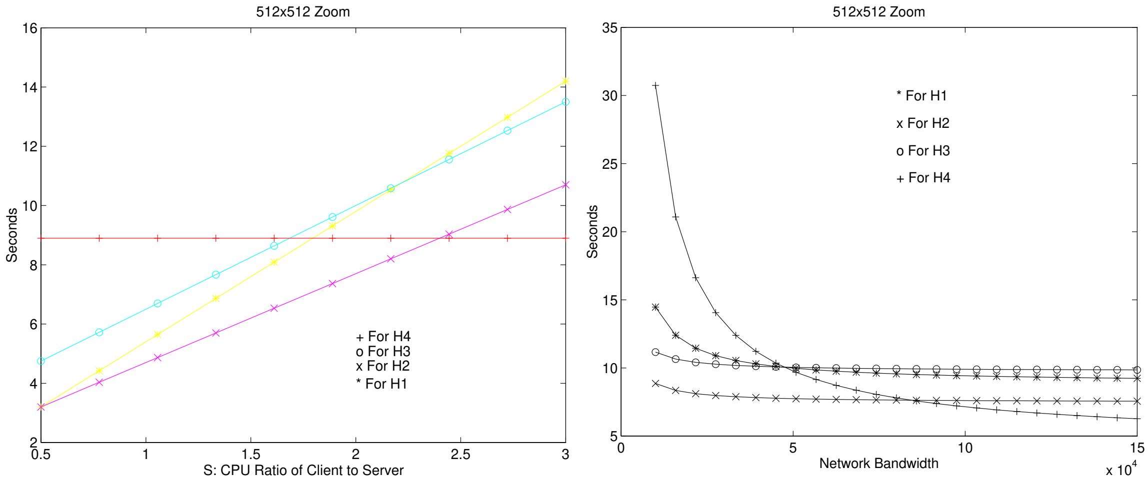
Between H_4 and H_2, if d is the same for Cases (a) and (b):

$$H_4 - H_2 = (R/p - S)(E_3 + E_2) + (n^2 - kF_1)(\frac{1}{B} + g) + F_1(\frac{(1-c)}{b_2} + \frac{c}{b_1}).$$

For Case (c):

$$H_4 - H_2 = (1 - 1/p)(A + O - F_1(\frac{1}{b_2} - \frac{1}{b_1}))R/p + (R/p - S)(E_2 + E_3) + (n^2 - kF_1)l(\frac{1}{B} + g).$$

If $R/p \geq S$ but B is very small, then $H_3 \geq H_2$, $H_4 \geq H_2$, and $H_1 \geq H_2$. Namely, H_2 will be the minimum and D_2 will be chosen. Intuitively, if the server is very busy, it is always better for the client to do as much as possible. But since the Internet is slow, D_2 is better than D_1 since D_2 involves less client-server communication.



(a) (b)

Figure 9: Image reconstruction for a subregion. (a) Client power varying, (b) Network bandwidth varying; each server node is twice as powerful as a client.

We graph the values of H_1, H_2, H_3, and H_4, illustrating their relationship to server load, client capabilities, and network bandwidth. We utilize the following parameters based our experimental data: n=512, k = 0.25, R = 6, p = 6, $g = 2.5 \times 10^{-7}$, $E_1 = 1.4$ sec, $E_2 = 0.4$ sec, $E_3 = 2.6$ sec, $b_1 = 1100000$ byte per second, $b_2 = 1000000$ bytes per second, A = 0.001 sec, O = 0.1 sec, $F_1 = 56000$ bytes, $F_2 = (n/2)^2$, $E_c =$ 0.9 sec, and E_d= 0.9 sec. Figure 9 shows two results of H_1, H_2, H_3, and H_4, when B ranges from 10000 to 150000 bytes/sec and S is set to range from 0.5 to 3. For points on the left side of the figure, it is advisable to send over the compressed subtree data for the client to process if the server is too busy, otherwise the server should send over the completed image. On the right, the conclusion is similar when the client-server bandwidth increases steadily.

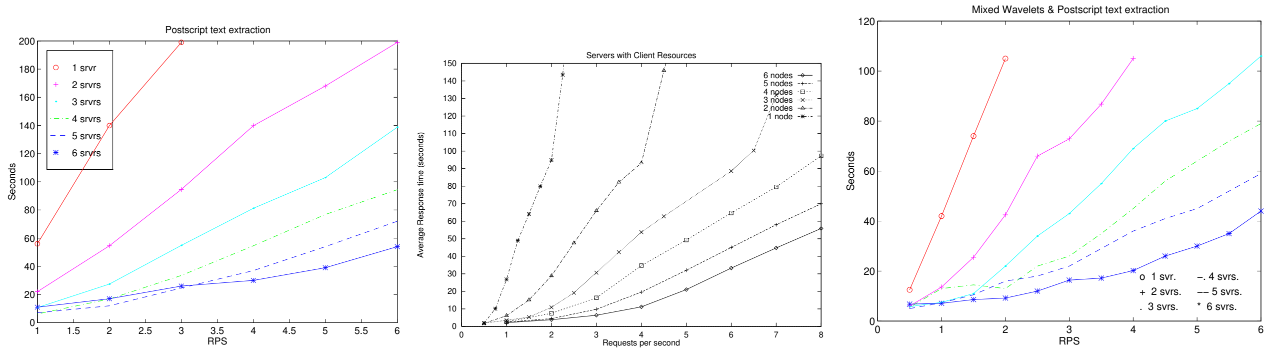
5 Experimental Results

We have implemented a prototype of our scheduling scheme on a Meiko CS-2 distributed memory machine. The Meiko CS-2 can be viewed as a workstation cluster connected by the Elan fast network. Each node has a a 40Mhz SuperSparc chip with 32MB of RAM running SUN Solaris 2.3. The TCP/IP layer communication on the Meiko can achieve approximately 3-15% of the peak performance (40MB/s). Our primary experimental testbed consists of six Meiko CS-2 nodes as our server. Each server node is connected to a dedicated SCSI 1GB hard drive on which test files reside. Disk service is available to all other nodes via NFS mounts.

We first examine the performance impact of utilizing multiple servers and client resources, then demonstrate that our scheme can successfully balance processor loads when few nodes receive more requests compared with others. We also present experiments supporting the analytical model presented in Section 4. We primarily examine the scheduling performance on three applications: file fetches, text extraction for postscript documents, and wavelet-based subimage retrieval. Each text extraction request consists of extracting one page of text from a 45-page Postscript file with size 1.5MB. The Postscript code for the single page is approximately 180KB, and the extracted text is about 2.5KB. The wavelet operation we choose is to extract 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. All results are averaged over multiple runs, and the test performance is affected by dynamically-changing system loads since the machines are shared by many active users at UCSB. The client machines are loaded with our custom library implementing some of the basic operations, including wavelet reconstruction. Clients are located within the campus network to avoid Internet bandwidth fluctuations over multiple experiments.

The overhead for monitoring and scheduling is quite small for all experiments. Analyzing a request takes about 2-4ms, and monitoring takes about 0.1% of CPU resources. These results are consistent with those in [4].

5.1 The impact of adding multiple servers



(a) (b) (c)

Figure 10: Request response times as p and/or RPS changes. (a) Postscript text extraction, (b) Wavelet subimage retrieval. (c) Mixed wavelet and postscript text extraction. The period is 30 seconds.

We examine how average response times decrease when p increases for a test period of 30 seconds, and at each second R requests are launched from clients ($RPS = R$). Figure 10 shows the average response times in seconds with client resources for processing a sequence of wavelet, Postscript or mixed requests. We can see from the experimental results that response times decrease significantly by using multiple server nodes, and this is consistent across all types of loads. The extreme slope for the one-node server is due to the nonlinear effects of paging and system overhead for a very high system load.

We more closely examine the relative scale-up ratio of performance from $i-1$ nodes to i nodes. This ratio is defined below where $H(i)$ is the response time using i nodes.

$$\Delta_{i-1}^{i} = \frac{\frac{H(i-1)}{H(i)}}{\frac{i}{i-1}}$$

The denominator indicates the server resource's increasing speed. If $H(i-1)$ is too large, then we use ∞ as the result. If $\Delta_{i-1}^{i} = 100\%$, it indicates a perfect performance scale-up. $\Delta_{i-1}^{i} > 100\%$ indicates a super-linear scale-up. This is possible because increased memory and disk resources reduce paging. Table 1 shows the relative scale-up ratio for processing a sequence of wavelet requests, which demonstrates that the system achieves a reasonable speedup with added multiprocessor resources. The scale-up results are similar for processing file fetching and text extraction requests.

	Δ_1^2	Δ_2^3	Δ_3^4	Δ_4^5	Δ_5^6
RPS=2	250%	200%	93%	106%	125%
RPS=4	∞	109.1%	125.3%	133.3%	175.9%
RPS=6		∞	102.9%	118.2%	117.2 %

Table 1: Relative scale-up ratios with client resources for wavelets.

5.2 The impact of utilizing client resources

We compare the improvement ratio of response time $H(i)$ with client resource over the response time $H'(i)$ without using client resource (i.e. all operations are performed at server). This ratio is defined as $H'(i)/H(i) - 1$. The comparison result for p=6 is shown in Table 2 for processing a sequence of Postscript text extraction requests. Table 3 is for wavelets. As the server load increases steadily, the response time improvement ratio increases dramatically.

RPS	0.5	1	2	3	4
Without (sec)	73.3	132.5	160	294	407
With (sec)	12.8	13.4	15.2	20.6	32.6
Imp. ratio	617%	889%	953%	1327%	1148%

Table 2: Average response time with and without client resources for text extraction. 30s. period, $p = 6$.

We also note a significant increase in the maximum number of requests per second (MRPS) a server system can complete over short periods by using client resources. If we consider a response time of more than 60 seconds as a failure in the case of wavelets, then the MRPS for the system with and without client resource in processing a sequence of wavelet-based requests is summarized in Table 4. Using client resources improves MRPS of a server by approximately 5 to 6 times.

RPS	1.0	1.5	2.0	2.5
Without client resources (sec)	15.97	57.24	123.76	213.4
With client resources (sec)	4.78	5.61	6.45	7.73
Improvement	234%	918%	1818%	2660%

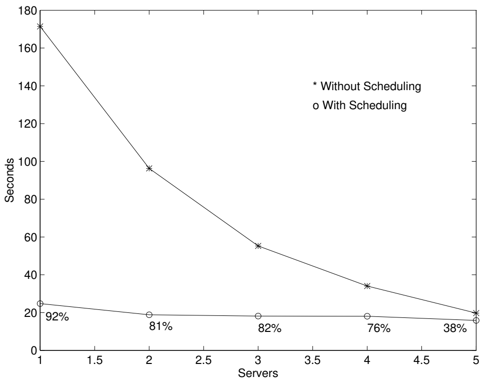
Table 3: Average response time with and without client resources for wavelets. 30s. period, $p = 6$.

	p=1	2	3	4	5	6
With client resources	1.5	3.0	4.5	6.0	7.5	9.0
Without	0.3	0.6	0.8	1.2	1.4	1.5

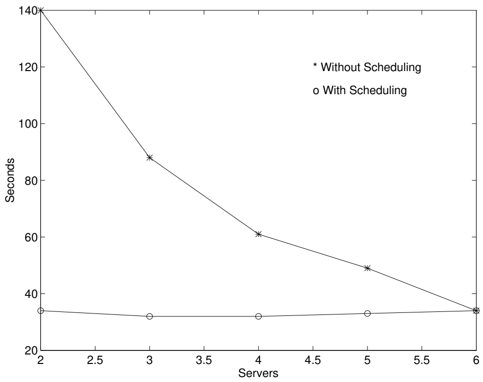
Table 4: Bursty MRPS for processing wavelets.

5.3 Load balancing with "hot spots"

"Hot spots" is a typical problem with DNS rotation, where a single server exhibits a higher load than its peers. Various authors have noted that DNS rotation seems to inevitably lead to load imbalances [18, 19]. We examine how our system deals with hot-spots by sending a fixed number of requests to a subset of nodes in our server cluster, giving a wide range of load disparities. Without our scheduler, the selected nodes would have to process all of those requests. The scheduler can effectively deal with temporary hot-spots at various nodes by redirecting requests to other nodes in the cluster. The result is shown in Figure 11 for extracting a 512×512 pixel wavelet subimage (left) and extracting the text from a Postscript page (right). The X axis shows the range of processors which receive requests. The upper curves of Figure 11(a) and (b) show 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 curves show the average response time with presence of our scheduler. We also mark the redirection rate for the wavelets requests. We note that redirection rates drop dramatically as load disparity goes down. This trend matches our expectation, because where the system tends to be homogeneous and the load is evenly balanced, our analytic model in Section 4 predicts a redirection rate of zero for both of the above experiments due to the relatively small file sizes.



(a) 512×512 wavelet subimage.

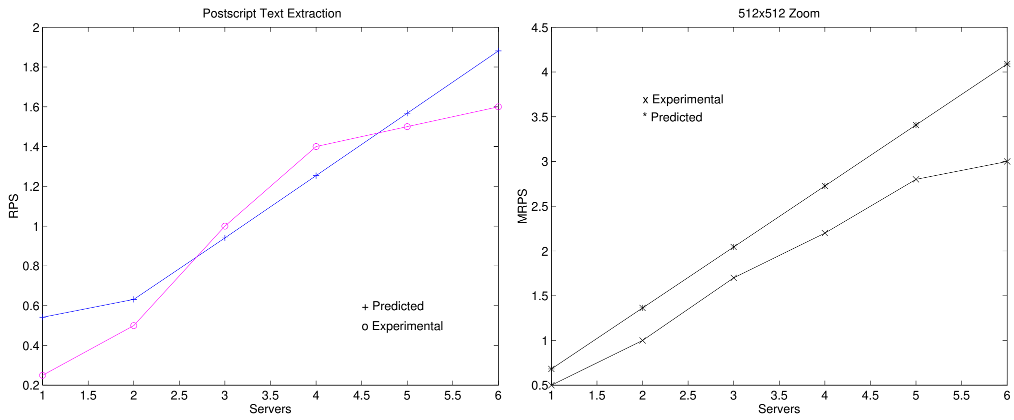


(b) Postscript text extraction.

Figure 11: System performance with request concentration at fixed server subsets. Tests for a period of 30 sec., 4 RPS. The indicated percentage is the redirection rate.

5.4 Verification of analytical results

We further conduct experiments on how the theoretical results presented in Section 4 match the system behavior under the specified assumptions.



(a) (b)

Figure 12: Sustained MRPS, experimental vs. predicted results. The test period is 120s. (a) Postscript. (b) Wavelets.

Sustained MRPS bound. For Postscript and wavelet requests, we have theoretical MRPS predictions in Section 4. We ran experiments to determine the actual MRPS by testing for a period of 120 seconds and choosing the highest RPS such that the server response times are reasonable and no requests are dropped. We chose the period of 120 seconds based on [13, 19], which indicate most "long" bursts on the Internet are actually relatively short. Thus the sustained RPS required in practice are for a period shorter than ∞. For this experiment, the clients are simulated within the Meiko machine. The aggregated bandwidth (B_s) of the 6-node server to the other client nodes is high, and does not create a bottleneck. The results are shown in Figure 12. In general, the predicted MRPS bounds reasonably match the trend of actual MRPSs. There is some discrepancy, which is caused by the following reasons: 1) paging and OS overheads are neglected in our model, and 2) there are other background jobs running in the system. Still, the overall accuracy of prediction is very reasonable. It should be noted that the previous load balancing research [22] normally use simulations to verify performance analysis and it is quite difficult to predict and match actual performance in a real experimental setting.

Expected redirection ratio. Experiments in Section 5.3 already indicate that the trend of redirection matches the theoretical prediction for the 512 × 512 wavelet extraction. We further examine the impact of varying file sizes on the redirection ratio. Figure 13 shows the predicted and actual redirection ratios when the fetched file size varies. Utilizing the formula presented in Section 4.2.1, the difference between $\frac{F}{b_1} - \frac{F}{b_2}$ and $A + O$ determines the theoretical switching point. We use the following data: $A + O \approx 0.1s$, $b_1 = 1.1MB/s$, $b_2 = 1.0MB/s$. Then we can determine that at $F \approx 1.1MB$, d will shift from 0 to $1 - 1/p$. We find that the actual redirection rate curve is quite close to that predicted, although slightly shifted due to other background system activity affecting algorithm parameters.

For wavelets, at the present time we do not have compressed wavelet files large enough to approach the predicted switch point. Thus the predicted redirection is a flat line and matches the actual ratio very well. Similarly for Postscript text extraction, our largest test files are very close to the crossover point. For example, utilizing a 1.5MB file gives a predicted $(\frac{1.5MB}{1.1MB/s} - \frac{1.5MB}{1MB/s}) - (0.1 + 0.001) = 35ms$. time advantage for redirection. Such a small predicted advantage is overwhelmed by other factors during the scheduling process, such as CPU load, network bandwidth and reconnection cost.

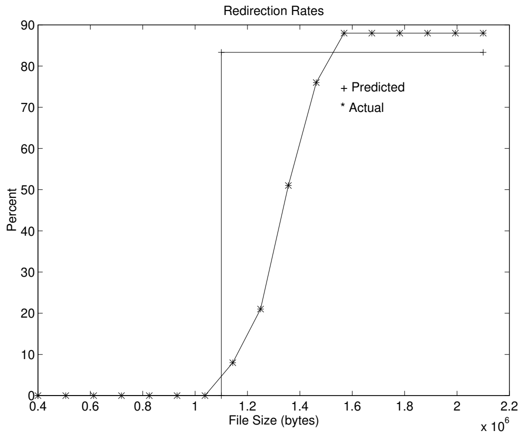


Figure 13: Redirection rates for file fetching. Experimental vs. predicted results.

Expected Split Points. In Figure 14, we compare the theoretically predicted split points with the actual decision made by the system scheduler in the Postscript and wavelet experiments. The system with six nodes is processing R concurrent requests (R= 18) when we artificially adjust the server/client bandwidth and CPU ratio reported by the client. Each coordinate entry in Figure 14(a) and (b) is marked with the decision of scheduler and the theoretical prediction. For each entry, if the choices for all requests agree with the theoretical prediction, we mark the actual selected split decision, otherwise we mark the percentage of disagreement. For example, in Figure 14(b), when available bandwidth B is 10,000 bytes/second and server/client CPU ratio $S = 2$, D_2 is the selected processing option for all requests, matching the analytical model's result. When $B = 100,000$ and $S = 2$, the percentage of disagreement with the theoretical model is 76% among R requests processed; however, this percentage is only for one entry (corresponding to one setting). For most of entries in Figure 14(a) and (b), the theoretical model matches the scheduler's selections.

Client/Server Bandwidth (bytes/sec) \ Server/Client CPU Speed Ratio	**0.25**	**0.5**	**1**	**2**	**4**	**10**	**20**
100	D3	D3	D3	D3	D3	D3	D3
500	D3	D3	D3	D3	D3	D3	D3
1,000	D3	D3	D3	D3	D3	D3	D3
10,000	90	D3	D3	D3	D3	D3	D3
100,000	6	15	D3	D3	D3	D3	D3
150,000	D2	D2	D3	D3	D3	D3	D3
250,000	D2	D2	D3	D3	D3	D3	D3
500,000	D2	D2	D3	D3	D3	D3	D3
750,000	D2	D2	D3	D3	D3	D3	D3
1,000,000	D2	D2	D3	D3	D3	D3	D3
	Predicted D2 region		***Predicted D3 region***				

(a)

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**
	Predicted D2 region				***Predicted D4 region***			

(b)

Figure 14: Effects of CPU speed and network bandwidth on (a) text extraction and (b) wavelet chain partitioning decisions. $R = 18$.

As observed in Figure 14(a), when there is no speed advantage for the client CPU, and the server is unloaded, the server is instructed to complete all computations (D_3). In the first few columns where the client is faster than the server node, Internet bandwidth plays the deciding part, since partition D_2 requires more bandwidth than D_3. On the borderline area between 10,000 and 100,000 bytes/second, the server decision is largely determined on real-time background tasks and network bandwidth fluctuations, which leads to a partial disagreement with the analytical model. For Figure 14(b), as client CPU speeds decrease, the server does more processing (D_4). But when client-server bandwidth decreases, the scheduler increases the percentage of client involvement for data decompression and image reconstruction (D_2), to minimize the size of data sent over the network.

6 Related work and conclusions

Several projects are related to our work. Projects in [12, 14] are working on global computing software infrastructures. Scheduling issues in heterogeneous computing using network bandwidth and load information are addressed in [10]. The above work deals with an integration of different machines as one server and does not have the division of client and server. Our current project focuses on the optimization between a server and clients and currently uses tightly coupled server nodes for a WWW server, but results could be generalized for loosely coupled server nodes. Addressing client configuration variation is discussed in [15] for filtering multi-media data but it does not consider the use of client resources for integrated computing.

Compared to the previous SWEB work [4], the main contributions of this work are an adaptive partitioning and scheduling scheme for processing requests by utilizing both client and multiprocessor server resources, and analytic results for supporting our scheduling scheme. The assumptions in the analysis are simplified, but the results help us understand the performance impact of several system resources and corroborate the design of our techniques. The experimental results show that properly utilizing server and client resources can significantly reduce application response times.

Acknowledgments

This work was supported in part by NSF IRI94-11330, NSF CCR-9409695 and a grant from NRaD. We would like to thank David Watson who helped in the implementation, Thanasis Poulakidas for assisting us in using the wavelet code, Cong Fu, Terry Smith, and the Alexandria Digital Library team for their valuable suggestions.

References

[1] The AltaVista Main Page, http://altavista.digital.com/.

[2] E. Anderson, D. Patterson, E. Brewer, "The Magicrouter, an Application of Fast Packet Interposing", submitted to the Second Symposium on Operating System Design and Implementation (OSDI96), 1996. Also available at http://HTTP.CS.Berkeley.EDU/~eanders/262/262paper.ps.

[3] 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", *Proc. of ISDL'95: International Symposium on Digital Libraries,* Japan, 1995.

[4] 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)*, pp. 850-856. April, 1996.

[5] 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 IEEE Forum on Research and Tech. Advances in Digital Libraries (ADL96)*, pp. 139-148, May, 1996.

[6] D. Andresen, T. Yang, D. Watson, A. Poulakidas, Dynamic Processor Scheduling with Client Resources for Fast Multi-resolution WWW Image Browsing, in *Proc. of the 11th IEEE International Parallel Processing Symposium (IPPS'97)*, Geneva, pp. 167-173, April, 1997.

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

[8] T. Bemers-Lee and D. Connolly, "Hypertext Markup Language - 2.0", http://www.w3.org/hypertext/WWW/MarkUp/html-spec/html-spec_toc.html, June, 1995.

[9] T. Bemers-Lee, "Uniform Resource Locators", http://www.w3.org/hypertext/WWW/Addressing/URL/, 1996.

[10] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks", *Proc. of Supercomputing '96*, ACM/IEEE, Nov., 1996.

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

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

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

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

[15] 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.

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

[17] Hypertext Transfer Protocol (HTTP): A protocol for networked information, http://www.w3.org/hypertext/WWW/Protocols/HTTP/HTTP2.html, June 26, 1995.

[18] 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.

[19] D. Mosedale, W. Foss, R. McCool, "Administering Very High Volume Internet Services", *1995 LISA IX*, Monterey, CA, 1995.

[20] NCSA development team, "The Common Gateway Interface", http://hoohoo.ncsa.uiuc.edu/cgi/, June, 1995.

[21] A. Poulakidas, A. Srinivasan, O. Egecioglu, O. Ibarra, and T. Yang, "A Compact Storage Scheme for Fast Wavelet-based Subregion Retrieval", To appear in the Proceedings of COCOON '97, Shanghai, China, August, 1997.

[22] B. A. Shirazi, A. R. Hurson, and K. M. Kavi (Eds), Scheduling and Load Balancing in Parallel and Distributed Systems, IEEE CS Press, 1995.