Multiprocessor Scheduling with Client Resources to Improve the Response Time of WWW Applications

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

Abstract

WWW-based information service has grown enormously during the last few years, and major performance bottlenecks have been caused by WWW server and Internet bandwidth inadequacies. Augmenting a server with multiprocessor support and shifting computation to client-site machines can substantially improve system response times and for some applications, it may also reduce network bandwidth requirements. In this paper, we propose adaptive scheduling techniques that optimize the use of a multiprocessor server with client resources by predicting demands of requests on I/O, CPU and network capabilities. We also provide a performance analysis under simplified assumptions for understanding the impact of system loads and network bandwidth when using our scheduling strategy. Finally we report preliminary experimental results to examine the system performance and verify the usefulness of the analytic model.

1 Introduction

Recently WWW-based computer technology has had an explosive growth in providing on-line access for images, files, and computing services over Internet. For example, most digital library (DL) systems [14] will be based on the WWW. The major performance bottlenecks for the advent of WWW technology are server computing capabilities and Internet bandwidth. Popular WWW sites such as *Alta Vista* [2] receive over twenty-eight million accesses a day, where the computational demand for each request is often less than that in many sophisticated WWW systems (e.g. DLs). In [4], a study is presented on developing multiprocessor Web servers in 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.

The main research challenge for such a computing environment is the effective management and utilization of resources from multiprocessor WWW servers and client-site machines. 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 the Internet networking built in. A careful design of scheduling strategies is needed to avoid imposing too much burden on these Net PCs. At the server site, information on the current system load and disk I/O bandwidth also 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 and we conducted a case study for the WWW-based image browsing application is in [5].

In this paper we propose a general scheduling model for partitioning and mapping client-server computation based on dynamically changing server and client capabilities. We present analytic results on homogeneous environments examining the impact of client and server resource availability. programmers in utilizing our model for their WWW applications. We are developing a software system called SWEB++ to support programmers in utilizing our model for their WWW applications. 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 scheduling strategy 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 analysis results. Section 6 discusses the related work and conclusions.

2 A model of WWW request processing

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, 15]. Each server node may have its local disk, which is accessible to other nodes via remote file service in the OS.

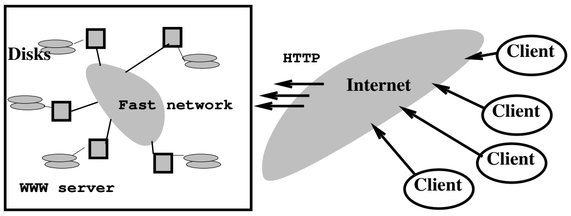


Figure 1: The architecture of a multiprocessor WWW server.

We model the interaction between client and server as a task chain that is partially executed at the server node (possibly as a CGI program [17]) and partially executed at the client site (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 can be scheduled onto one of the server nodes. Our challenge, then, is to select an appropriate node within the server cluster to process the request modeled by a task chain. Also we need to partition tasks in this chain into two sets, one for the client and another for server, such that the overall request response time is minimized as illustrated in Figure 2. For example, if the client happens to be underpowered and one server node is lightly loaded, then keeping things on that server node could be more reasonable in terms of request response time. In addition to client and server machine load and capabilities, network bandwidths between clients and a server also affect the partitioning point. We assume that local communication 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.

An example for text extraction from a Postscript file. 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 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 view the content of a Postscript document [13].

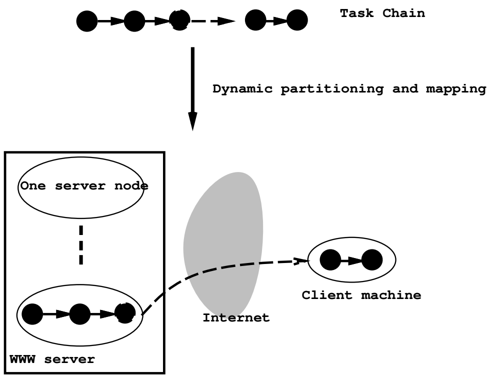


Figure 2: An illustration of dynamic task chain partitioning and mapping.

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 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.

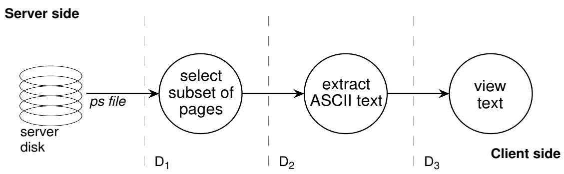


Figure 3: The task chain for text extraction from a Postscript file.

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 technical paper with 750KB 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.

Another example for multi-resolution image browsing. This application comes from the Alexandria Digital Library system (ADL) [3]. ADL has adopted a progressive multi-resolution and subregion browsing strategy to reduce Internet traffic in accessing map images. This approach is based on the idea that users often browse large images via a

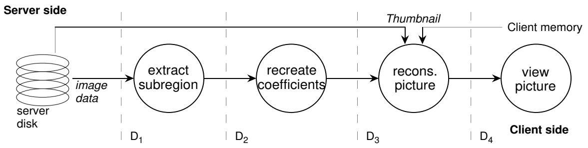


Figure 4: A task chain for image reconstruction.

thumbnail (at coarse resolution), and desire to rapidly view higher-resolution versions and subregions of those images already being viewed. To support these features, the ADL system is using a wavelet-based hierarchical data representation for multi-resolution images [9]. Figure 4 depicts a task chain for processing a user request in accessing an image or subimage with a higher resolution, based on an implementation in [18]. Tasks in this chain are: 1) Fetch and extract the subregion indexing data when a user wants to review a subregion. 2) Use the indexing data to find coefficient data to be used in constructing a higher resolution subregion image. 3) Use the coefficient data and thumbnail image to perform a wavelet inverse transform in producing a higher resolution image. Thus there are four possible split points between a client and a server as shown in Figure 4. D_1 **-** Send the entire compressed image to the client and let it do everything. D_2 **-** Send over the relevant portions of the compressed image for the client to process. D_3 **-** Recreate the coefficients from their compressed representation and send them to the client for reconstruction. D_4 **-** Recreate the image requested by the client and send the pixels to the client. For split point D_3, the thumbnail file is available at client machine, thus the data access from the server disk for thumbnail data is eliminated.

3 Request Scheduling

Several factors affect response times for processing requests, including file locality, CPU/disk loads, and network resources. 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 partitioning point. This cost model is based on our previous SWEB work [4] with extensions to incorporate the impact of client resources and network bandwidth and guide the client-server partitioning. Notice that it is not easy to model the cost associated with processing a WWW request accurately and factors such as caching and contention are not considered. Our experiments show that the current cost function does reflect the impact of multiple parameters on the overall system response performance, and that the multi-node system delivers good performance based on such a heuristic.

In [4], we proposed using *URL redirection* to implement the request re-assignment. This 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.

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

$$t_s = \ t_{redirection} + t_{data} + t_{server} + \ t_{net} + t_{client}. \qquad (1)$$

$t_{redirection}$ is the cost to redirect the request to another processor, if required. t_{data} is the server time to transfer the required data from a server disk drive. If data files are local, the time required to fetch is approximated as the file size divided by the available bandwidth of the local storage system. If data files are remote, then each file must be retrieved through the interconnection network. Then t_{data} is approximated as the file size divided by the available remote accessing bandwidth. If there are many concurrent requests, local data transmission performance degrades accordingly. At run-time, the system needs to monitor the disk channel load of each local disk and also available remote data accessing bandwidths.

t_{server} is the time for server computation required, which is $CPU_{load}\frac{\text{No. of server operations required}}{^{CPU}\text{server speed}}$. 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 server 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 experiments.

t_{net} is the cost for transferring processing results over the Internet, which is the size of client-server communication volume divided by available network bandwidth. t_{client} is the time for any client computation required, which is the number of client operations required divided by client speed. Here we assume the speed reported by the client machine includes client load factors. Again client-server communication and client-side computation requirements depend on how a chain is partitioned. For the wavelet example, 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.

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[1], 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 5 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

[1] A simple file retrieval can also be considered as a task chain [6].

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 5: The procedure for scheduling a task chain.

4 An analysis for homogeneous client-server environments

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 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 requests. 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, 11]. 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 at 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 at 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_s – the average total task time needed at the server site.
- E_c – the average total task time needed at the client site.

Then

$$H = c*\frac{F_s}{\frac{b_1}{Z}}+(1-c)*\frac{F_s}{\frac{b_2}{Z}}+(A+d(A+O))Z+E_sZ+F_n/B+S*E_c. \quad (2)$$

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

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.

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

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

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

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

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:

$$\min(\frac{p}{c * \frac{F_s}{b_1} + (1-c) * \frac{F_s}{b_2} + A + d(A+O) + E_s}, \frac{B_s}{F_n})$$

where $c = d + 1/p$. Naturally the MRPS may be maximized by having the client do everything possible to minimize E_s. 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.

The expected split point for $(r, 1)$. For this case, $Z = r$, and the H formula in (2) is simplified as follows.

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

With different split points, all four terms F_s, F_n, E_s and E_c may change, affecting the optimal 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 studies

In [6], we have conducted case studies using the above framework for simple file fetches and wavelet-based image browsing. Here we present some results for the text extraction chain 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. Then

$$H = \min(H_1, H_2, H_3). \quad (3)$$

Let $Y = c * \frac{F_1}{b_1} + (1-c) * \frac{F_1}{b_2} + A + d(A+O)$. We can show that

$$H_1 = r * Y + r * gF_1 + S * E_1 + S * E_2 + F_1/B.$$

$$H_2 = r * Y + r * (E_1 + gk_1F_1) + S * E_2 + k_1 * F_1/B.$$

$$H_3 = r * Y + r * (E_1 + E_2 + gk_1k_2F_1) + k_1 * k_2 * F_1/B.$$

Expected Redirection Ratio for $(r, 1)$ and (r, ∞). For H_1, H_2, and H_3, $F_s = F_1$, then using the result in Section 4.1, d can be selected as either 0 or $1 - 1/p$ based on the difference between $A + O$, $F_1(\frac{1}{b_2} - \frac{1}{b_1})$.

MRPS for (r, ∞). The expected MRPS R^* depends on the three choices of partitioning.

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

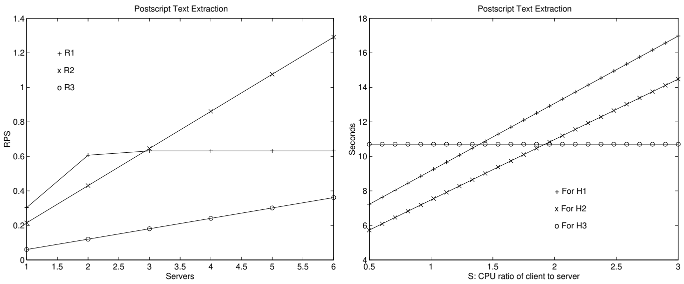
We calculate the MRPS bound R_i^* for each partitioning as:

$$R_1^* = \min(\frac{p}{Y + gF_1}, \frac{B_s}{F_1}).$$

$$R_2^* = \min(\frac{p}{Y + E_1 + gk_1F_1}, \frac{B_s}{k_1F_1}).$$

$$R_3^* = \min(\frac{p}{Y + E_1 + E_2 + gk_1k_2F_1}, \frac{B_s}{k_1k_2F_1}).$$

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 6(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 6: (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 6(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.

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 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. We also present experiments supporting the analysis 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

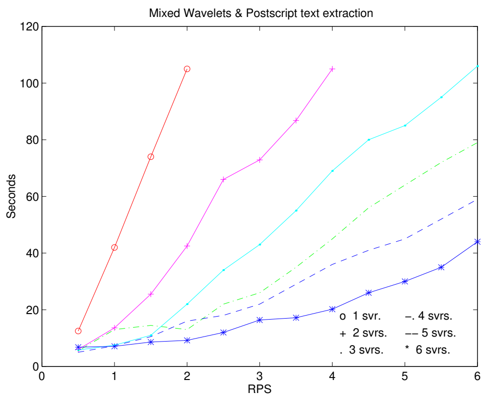


Figure 7: Average request response times as p and/or RPS changes in processing mixed wavelet and Postscript requests.

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 7 shows the average response times with client resources for mixed types of wavelet and Postscript 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. 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 resource for text extraction. 30s. period, $p = 6$.

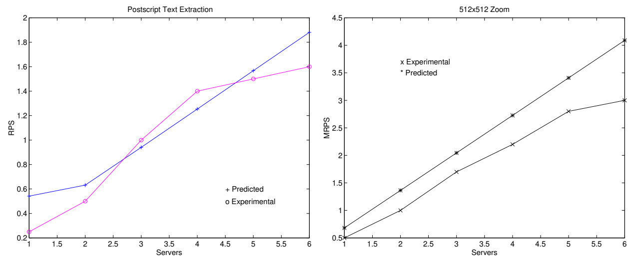
We also note a significant increase in the maximum number of requests per second (MRPS) a server system can complete over short periods by using client resources. If we consider a response time of more than 60 seconds as a failure in the case of wavelets, then the MRPS for the system with and without client resource in processing a sequence of wavelet-based requests is summarized in Table 3. Using client resources improves MRPS of a server by approximately 5 to 6 times.

5.3 Verification of analytical results

We further conduct experiments on how the theoretical results presented in Section 4 match the system behavior in terms of MRPS and split points under the specified assumptions. A verification for redirection ratios is in [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 3: Bursty MRPS for processing wavelets.



(a) (b)

Figure 8: 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 [11, 16], which indicates most "long" bursts on the Internet are actually relatively short. Thus the sustained RPS required in practice are for a period shorter than ∞. Notice that 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 8. 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 reasonable. It should be noted that the previous load balancing research [19] normally use simulations to verify performance analysis and it is quite difficult to predict and match actual performance in a real experimental setting.

Expected Split Points. In Figure 9, 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 9(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 9(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 9(a) and (b), the theoretical model matches the scheduler's selections.

Server/Client CPU Speed Ratio

Client/Server Bandwidth (bytes/sec)	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)

Server/Client CPU Speed Ratio

Client-server bandwidth (bytes/sec)	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 9: Effects of CPU speed and network bandwidth on (a) text extraction and (b) wavelet chain partitioning decisions. $R = 18$.

As observed in Figure 9(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 9(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 concluding remarks

The evolution of the WWW brings tremendous changes for Web-based computing. Compared to the previous SWEB work [4], the main contributions of this work are an adaptive scheduling model 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 proper utilizing of server and client resources can significantly reduce application response times. Our current work is developing a software system called SWEB++ [6] which implements and supports the use of our scheduling scheme when programming WWW applications. The system contains a user interface for task specification, a module composer, and run-time performance monitor and scheduling support.

Several projects are related to our work. Projects in [10, 12] are working on global computing software infrastructures. Scheduling issues in heterogeneous computing using network bandwidth and load information are addressed in [8]. The above work deals with an integration of different machines as one server and does not have the division of client and server. Recent work on mobile computing (e.g. [1]) also uses bandwidth and data locality information to dynamically migrate computation. 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 [13] for filtering multi-media data but it does not consider the use of client resources for integrated computing.

Acknowledgments

This work was supported in part by NSF IRI94-11330, NSF CCR-9409695, NSF CCR-9702640, 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, Omer Egecioglu, Cong Fu, Oscar Ibarra, Terry Smith, and the Alexandria Digital Library team for their valuable suggestions.

References

[1] A. Acharya, M. Ranganathan, J. Saltz, "Sumatra: A Language for Resource-aware Mobile Programs," To appear in Mobile Object Systems, J. Vitek and C. Tschudin (eds), Springer Verlag Lecture Notes in Computer Science.

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

[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, 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, April, 1997.

[6] D. Andresen, T. Yang, Adaptive Scheduling with Client Resources to Improve WWW Server Scalability, UCSB TRCS96-27.

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

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

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

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

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

[12] K. Dincer, and G. C. Fox, Building a world-wide virtual machine based on Web and HPCC technologies. To Appear in Supercomputing'96, ACM/IEEE, Nov 196.

[13] 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.

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

[15] 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.

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

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

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

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