Scalability Issues for High Performance Digital Libraries on the World Wide Web

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

Abstract

We investigate scalability issues involved in developing high performance digital library systems. Our observations and solutions are based on our experience with the Alexandria Digital Library (ADL) testbed under development at UCSB. The current ADL system provides on-line browsing and processing of digitized maps and other geo-spatially mapped data via the World Wide Web (WWW). A primary activity of the ADL system involves computation and disk I/O for accessing compressed multi-resolution images with hierarchical data structures, as well as other duties such as supporting database queries and on-the-fly HTML page generation. Providing multi-resolution image browsing services can reduce network traffic but impose some additional cost at the server. We discuss the necessity of having a multi-processor DL server to match potentially huge demands in simultaneous access requests from the Internet. We have developed a distributed scheduling system for processing DL requests, which actively monitors the usages of CPU, I/O channels and the interconnection network to effectively distribute work across processing units to exploit task and I/O parallelism. We present an experimental study on the performance of our scheme in addressing the scalability issues arising in ADL wavelet processing and file retrieval. Our results indicate that the system delivers good performance on these types of tasks.

1. Introduction

The number of digital library (DL) projects is increasing rapidly at both the national and the international levels (see, for example, [6, 2]). Many of the current projects are moving rapidly towards their goals of supporting on-line retrieval and processing of major collections of digitized documents over the Internet.

Performance and scalability issues are especially important for the Alexandria Digital Library (ADL) project [2, 14]. The fundamental goal of this project is to provide users with the ability to access and process broad classes of spatially-referenced materials from the Internet. Materials that are currently in the collections of ADL and accessible through the ADL World Wide Web (WWW) server include *geographically*-referenced items such as digitized maps, satellite images, digitized aerial photographs, and associated metadata. When fully developed, ADL will comprise a set of nodes distributed over the Internet supporting such library components as collections, catalogs, interfaces, and ingest facilities. ADL is currently building collections that will involve millions of items requiring terabyte levels of storage. Many collection items have sizes in the gigabyte range while others require extensive processing to be of value in certain applications. The catalog component alone contains a metadatabase of significant size.

Before such goals can be achieved, however, major issues of performance and scalability must be resolved, particularly for DLs supporting extensive collections or collections with large data items. Critical performance bottlenecks that must be overcome to assure adequate access over the Internet involve server processing capability and network bandwidth. 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 the current system. Additionally, the server performance must scale to match expected demands.

A strategy used in the ADL system for reducing network traffic is to provide the service of progressive image browsing and the subregion retrieval, to avoid unnecessary large image transmission. The trade-off, however, is that more processing is required at the server site. Considering that popular WWW sites such as Alta Vista, Lycos and Yahoo have been receiving over two million accesses per day (or 20-30 requests per second), and the ADL server involves much more intensive I/O and heterogeneous CPU activities, a multi-processor server becomes indispensable [2].

In this paper, we investigate the network bandwidth requirement in the ADL system using progressive image browsing retrieval and the computational and I/O demands for supporting such activities. We study the use of networks of existing, inexpensive workstations and disks to augment the processing and storage capabilities of DL servers. In particular, we have investigated to what extent recycling the idle cycles of processing units in networks of workstations, as well as retrieving files in parallel from inexpensive disks can significantly improve the scalability of a DL server responding to many simultaneous requests.

We have implemented our scheme on a cluster of SUN SPARC nodes connected by a Meiko CS-2 Elan network, and clusters of SUN and DEC workstations connected by Ethernet. Each processing unit (e.g. a SUN SPARC node) is linked to a local disk and is capable of handling a user request. There are a variety of resource constraints that can affect the performance of the server. These constraints include: the CPU speed and memory size of a single processing unit; the current system load; the transmission bandwidth between the processing unit and its local disk; the network latency and bandwidth between a processing unit and a remote disk when the accessed files are not stored in the local disk; and disk contention when multiple I/O requests access the same disk. By understanding these effects, we can achieve server scalability. In particular, by actively monitoring the run-time CPU, disk I/O, and network loads of system resource units, we can dynamically schedule user requests to nodes in a manner that provides the greatest overall processing efficiency.

Our strategies are based on our work for a scalable WWW server called SWEB [1]. We assume that the system contains a set of networked workstations (see Figure 3). Some of the workstations are connected to SCSI-II disks or mass storage subsystems. In this paper, the terms workstation unit, node, and processor are interchangeable. We assume that each CPU unit may be used by other applications and can leave and join the resource pool at any time.

The paper is organized as follows: Section 2 briefly describes the ADL Project. Section 3 discusses the scalability issues addressed in the ADL system for ameliorating network and server bottlenecks. Section 4 discusses scheduling and resource monitoring strategies in our multi-processor system. Section 5 presents experimental studies concerning multi-node performance and analyzing overhead and the effectiveness of resource scheduling. Section 6 discusses related work. Section 7 discusses conclusions and future work.

2. The Alexandria Digital Library on the WWW

Key aspects of the development strategy for ADL involve:

- developing a library whose components are distributed over the Internet and are accessible to many classes of users;
- following an evolutionary and incremental approach to both design and implementation;
- focusing on the design of *digitally supportable extensions* to traditional library functionality, and making ADL consistent with requirements of the library community;
- providing the user with access to the implicit information available in the DL collections, as well as to the explicit information;
- developing collections of *spatially-referenced* materials.

We comment briefly on these key strategic points.

Since we are initially focusing on users who access ADL from the WWW, primary access to ADL is from WWW browsers connected to the ADL WWW server. Z39.50 clients are also able to connect with the ADL SQL/Z39.50 query engines. 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, while the HTTP protocol is the application-level mechanism for achieving the transfer of information [8]. The WWW supports general types of multimedia information systems while a DL system provides more advanced features for browsing, searching, and delivering digitized documents.

A major reason for adopting an evolutionary and incremental approach to design and development stems from the rapidity of developments in Internet technology. The first increment in the development of ADL involved the design and construction of a stand-alone "rapid prototype" (RP) system [5]. A second, and now completed, increment provided an augmented version of the functionality of the RP over WWW. The third increment is focused on developing a greatly enhanced catalog component based on a general model of metadata and supporting catalog interoperability with other DLs.

The approach of providing digitally-supportable extensions to traditional libraries is represented in the high-level architecture of ADL shown in Figure 1. Each of these components may be distributed over the Internet. This architecture involves the four major components of traditional libraries, namely a catalog component, a collection component, an ingest component, and a user interface component.

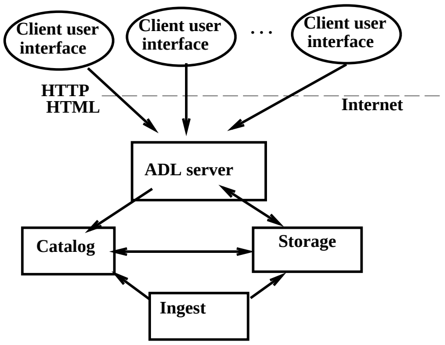

Figure 1. The ADL architecture.

- The *storage component* of ADL contains a collection of *digital objects*. The collections on which ADL is initially focused include spatially-referenced materials, such as digitized maps, digitized aerial photographs, and images from many domains of application [5]. An important aspect of the ADL collection is that individual items are typically very large. Satellite images are frequently 100 MB in size, and sizes of up to two GB are not uncommon.

- The *catalog component* of ADL permits users to make a mapping between their requirements for information and the most appropriate set of information that can be accessed from the library's collection of items. Since it is important that spatially-referenced items be accessible by means of a spatial reference, each item is represented in the catalog by a *spatial footprint*, which is a pointset characterizing the spatial extent of the item in the space over which it is defined. Footprints are represented in an extensible metadata model for spatially-referenced information (currently combining the FGDC and USMARC standards [5]) and are indexed to support efficient search over the catalog holdings. The metadata model also incorporates extensions involving gazetteers (i.e. mappings between named geographic features and the footprints of their spatial extent) and preselected image "textures features". Both the gazetteer and the image texture features are used to support content-based search.
- The *ingest component* involves the digitization of non-digital items; the extraction of catalog metadata from items that are admitted to the collections (which initially may be in either digital or non-digital form); and the application of transformations, such as wavelet decompositions, to ingested items. The metadata extraction is in accordance with the metadata model of the catalog. Currently available metadata includes approximately 450K frame-level records for a NASA/Ames database; approximately 350K sheet-level records for Geodex topographic series; approximately 100K USMARC map records from MELVYL; and catalog records for selected items, such as WWW sites for spatial data in digital form and aerial photographs for four local counties in California.
- The goal of the *interface component* is to provide easy access to a core set of functionality for a heterogeneous user population. This component contains a browser based on HTTP/HTML to support user access via the WWW. Current implementations of HTTP/HTML impose significant limitations on browsers, such as statelessness and the general reliance on small, fast transactions. In particular, HTML lacks mechanisms for presenting spatial data in vector form, and provides weak support for the entry of spatially-indexed information. The development of the WWW interface component has involved attempts to overcome such limitations, and the system provides external viewers and "helper apps" for the display of spatially-indexed materials of both raster and vector type. These limitations are being overcome with the current generation of programmable browsers.

3. Scalability of the ADL

As noted above, digitized data objects in ADL are typically very large. 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 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 of users on the Internet. The ADL has adopted progressive multi-resolution and subregion browsing strategies to reduce Internet traffic in accessing map images. This approach is based on the following ideas:

- Users often make the selection of materials of interest without browsing the image information at fine-grain levels of resolution; in particular, they initially need information only at coarse levels of resolution. Delivering images at coarse grain resolution substantially reduces the size of data transferred between the client and the ADL server.

  For current computer monitors, it is reasonable to assume that one would usually view an image of resolution 512 × 512, or at most $1K \times 1K$ to fit a screen. Compressed 512 × 512 color images have size of 100K-300KBytes and take around ten seconds to transfer over a T1 link.
- Users should be able to rapidly view higher-resolution versions of those images already being viewed to assist their selection. It is desirable to have a method that can construct a higher resolution image from the lower resolution image with only a small amount of additional data. If such construction can be performed at the client site, then since the lower solution image is already available at the client site, only the difference data needs to be transferred from the ADL server over the Internet. Note that the size of difference data is usually small, taking less than 1 second to deliver in a T1 link.
- Satellite map images usually have high resolutions (e.g. $2K \times 2K$ and $10K \times 10K$), and such high resolutions can not be viewed on a regular screen. A reasonable user requirement is to browse a subregion of an image to identify details of interest. Popular subregion resolutions are likely to be around 512 × 512. Thus supporting subregion browsing can also significantly reduce network bandwidth requirements.

To support these features, the ADL system is using a wavelet-based hierarchical data representation for multi-resolution decomposition of images. Images and their subregions can be browsed in different levels of resolution and can be delivered in a progressive manner [2]. We briefly describe the techniques of wavelet image data retrieval and transformation below.

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

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

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

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

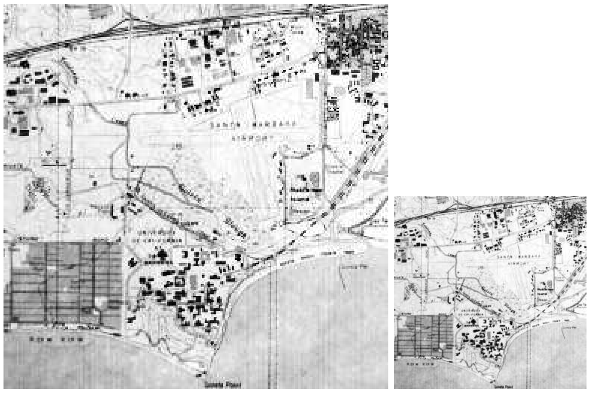

Figure 2. *left* **A map image I_1.(right) The thumbnail I_2 after applying the forward wavelet transform.(left)**

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 ADL sends C_1, C_2, C_3, image I_1 can be reconstructed at the client site. The image reconstruction is not time consuming, taking about 1.5 seconds for a 512 × 512 image on a SUN SPARC 5. The size of compressed data C_1, C_2, C_3 to be transferred is 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 [4]. The time complexity of wavelet transforms is proportional to the image size. The wavelet transform can be applied recursively, namely the thumbnail I_2 can be decomposed further to produce smaller thumbnails $I_2, I_3, \cdots$. The ingest component of the ADL performs the forward transformation to decompose data images into thumbnails and the coefficient data set.

The ADL system uses the above wavelet technique and at runtime, the following operations will be frequently invoked on the ADL server.

- *Retrieval of regular files and thumbnail images:* when a client requests a thumbnail for the initial browsing, the server retrieves the corresponding file from the ADL storage.
- *Retrieval of image coefficient data for progressive image browsing:* when a client further requests images with higher resolutions, the server retrieves compressed image coefficient data from permanent storage and delivers it to the client as the client machine performs the inverse wavelet to construct images of higher resolutions from the existing thumbnail and new coefficient data. If the client machine does not have such a capability, the server performs the image reconstruction.
- *Retrieval of image subregions:* after a client identifies an interesting point in an image, the request is made for an enhanced resolution view of the subregion surrounding that point. The appropriate image data is then retrieved and sent to the client.

If a user finds it necessary to access the original large image, e.g., for scientific applications, the ADL will direct this request to a large storage server (currently a 1TB robotic tape storage device at the San Diego Supercomputer Center). This file will be delivered via ftp since large images (with size 10-100MB), take a long time to transfer over the Internet. We do not address the issue of ftp delivery of large documents in this paper.

It should be noted that there are other operations performed in the ADL server. For example, content-based database queries to find suitable images are important, so the speed of the database server and its supporting mass storage is vital [2, 12]. We assume that the database functionality is provided by a separate computer within ADL, and so focus our attention on the problem of delivering data, whether simple files or wavelet data, to the user as quickly as possible over the Net. We also focus on scalability issues in supporting the multi-resolution and subregion browsing of images.

While we have addressed issues relating to the network bandwidth bottleneck, the ADL server itself can be another bottleneck in document delivery. For example, the computation and disk I/O performed in the ADL for decompressing and accessing subregions of images involve a substantial amount of time and occupy disk I/O channels for long periods.

There are several aspects in assessing the scalability of a DL system. When there are many requests coming in, the typical situation (such as one in the current WWW servers) is that the server's response for an individual request becomes slow. For a single-workstation server, there is an upper bound for the number of requests per second (RPS) that the server can handle. For example, a SPARC 10 can handle 4 requests per second for delivering files of size 1MB-2MB. When the system limit is reached, the requests fail due to congestion at the server. Our overall objective for the system is to reduce and sustain the response time under large numbers of simultaneous requests. We define the response time for a request as the length of time from when a request is initiated until all requested information arrives at the client for that transaction. Another performance goal is to have the RPS limit of the system as large as possible since the access activities of current popular WWW sites already indicate that ≥ 20 requests per second can be expected.

We performed an experiment to determine the network bandwidth requirements after adopting the progressive image browsing strategy, and examine the ADL server requirements. Table 1 gives the compressed size for a number of greyscale images. Each image was eight bits per pixel, and was a square image. We use the compression algorithm developed in [11] whose compression ratio is approximately 90%. The compressed full images are still sizeable. Using progressive image delivery can reduce network demands since full resolution images are not always required. For example, with the "pentagon" image (a satellite photo of the Pentagon), the 128 × 128 pixel thumbnail requires just 3.3K, and only 6.2K of additional data is needed to view this image at a 256 × 256 resolution. This indicates our multi-resolution/subregion browsing strategy significantly reduces network bandwidth requirements. But retrieving subregions from the compressed images imposes processing cost. On a Sun SPARC station 10, the current implementation takes 2 seconds of CPU time for extracting 8KB of compressed coefficient data from a 2K x 2K pixel greyscale image. To support 20 users per second, at least 40 high-end workstations need to be employed.

Image	*Dim.*	*Full*	*Thumb*	×2	×4
lena	256^2	7K	1K	1.5K	2.3K
textures	512^2	47K	3.3K	6K	14K
pentagon	1024^2	102K	3.3K	6.2K	16.4K
spot_reg1	2048^2	440K	13K	27K	75K

Table 1. Wavelet compressed data size for progressive delivery. "Full" is the compressed full image size. The thumbnail size is 1/8 of the original image dimensions.

To reduce and sustain the response time under large numbers of simultaneous requests, our strategies involve:

- Utilizing multiple networked commodity workstations and disks to build a scalable server. The computing environment can be heterogeneous and workstation/processor units with different speeds and different loads at any time.
- Developing sophisticated dynamic scheduling algorithms for exploiting task and I/O parallelism adaptive to the runtime change of system resource loads and availabilities. The system needs to provide good system resource estimation to assist the scheduler. The scheduler needs to incorporate multiple system performance parameters to assign user requests to a proper unit for efficient processing. The overhead involved for current resource load assessment and scheduling should be minimized.

4. System Resource Monitoring and Request Scheduling

In this section we first discuss the monitoring of system resources, introduce the functional modules of our scheduler, and we then present an algorithm for determining the processor assignment of a given ADL request.

There are several factors that affect the response time in processing ADL requests. These include loads on CPU, disk, and network resources. The load of a processing unit must be monitored so that requests can be distributed to relatively lightly loaded processors. An ADL request following the HTTP protocol is handled through a TCP/IP connection and then subsequently through a forked subprocess. This subprocess may retrieve a file or invoke a Computational Gateway Interface (CGI) program implementing an ADL-specific operation. In processing most ADL requests, image data needs to be retrieved from disks, so disk channel usage must be observed. Simultaneous user requests accessing different disks can utilize parallel I/O to achieve higher throughput. The local interconnection network bandwidth affects the performance of file retrieval since many files may not reside in the local disk of a processor. Therefore remote file retrieval through the network file system will be involved. Local network traffic congestion could dramatically slow the request processing.

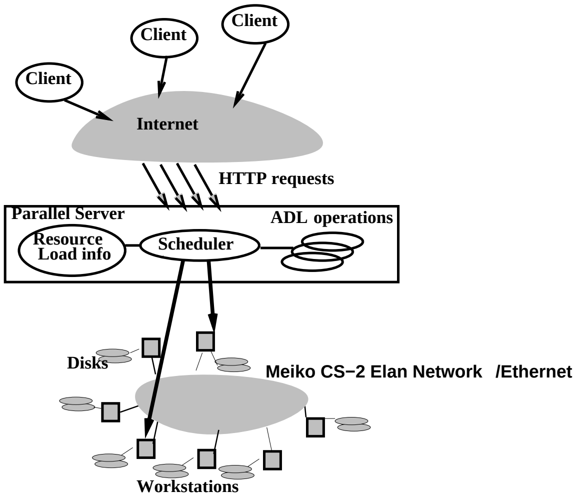

Figure 3. ADL multi-node server architecture.

Thus a critical component of our system is a load daemon running at each processor to detect its own CPU, disk, and network load, and periodically broadcasts this information to other processors. Our experiments show that such overhead is insignificant.

4.1. Internal scheduler structure

There are two approaches to designing the scheduler. One is to have a centralized scheduler running on one processor such that all requests go through this processor. The scheduler monitors the usages of all system resources, makes assignment decisions based on this information, and routes requests to appropriate processors. Our main reason for not adopting this approach is that the central distributor becomes a single point of failure, making the entire system vulnerable.

The current version of our system uses a distributed scheduler. The user requests are first evenly routed to processors via Domain Name System (DNS) rotation. DNS rotation provides the initial assignment of HTTP requests and is used in the NCSA multi-workstation server [10]. In this scheme, multiple real machines are mapped to the same IP name. When a client requests the network ID of the machine name (e.g., www.cs.ucsb.edu), the DNS at the server site rotates the network IDs, picking one (e.g., 1.1.1.1) to send back to the client. The rotation on available workstation network IDs is in a round-robin fashion. This functionality is available in current DNS systems. The major advantages of this technique are simplicity and ease of implementation [10].

The DNS is subject to caching problems when attempting to do dynamic load balancing, and it assigns requests without consulting dynamically-changing system load information. Thus our scheduler conducts a further re-direction of requests. Each processor in the ADL server contains a scheduler and those processors collaborate with each other to exchange system load information. After a request is routed to a processor via DNS, the scheduler in that processor makes a decision regarding whether to process this request or redirect it to another processor. An HTTP request is not allowed to be re-routed more than once in order to avoid the ping-pong effect.

The functional structure of the scheduler at each processor is depicted in Fig. 4. It contains a daemon based on NCSA httpd code for handling httpd requests, with a *broker* module that determines the best possible processor to handle a given request. The broker consults with two other modules, the $oracle$ and the $loadd$. The $oracle$ is a miniature expert system, which uses a user-supplied table to characterize the CPU and disk demands for a particular task. The $loadd$ daemon is responsible for updating the system CPU, network and disk load information periodically (every 2-3 seconds), and marking those processors which have not responded in a preset period of time as unavailable. When a processor leaves or joins the resource pool, the $loadd$ daemon will be aware of the change.

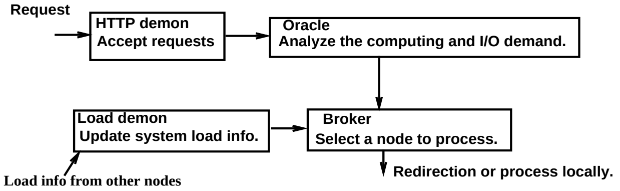
Request
HTTP demon
Accept requests
Oracle
Analyze the computing and I/O demand.
Load demon
Update system load info.
Broker
Select a node to process.
Redirection or process locally.
Load info from other nodes

Figure 4. The functional modules of a scheduler for each processor.

After a processor accepts a HTTP request through TCP connection, the system assigns a request to a particular processor. We discuss how this request can be *transparently* routed to this processor. The best situation would be to modify the UNIX sockets package to change the semantics of the "accept" system call, but such modification requires a substantial change of the UNIX operating system kernel. The approach that we implemented is based on the fact that the HTTP protocol allows for a response called "URL Redirection". When client C sends a request to server S_0, S_0 returns a rewritten URL r' and a response code indicating that 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.

The primary advantages of URL redirection are the simplicity of implementation and universal compatibility. An simple flow-of-control modification can be made within a WWW server, where the main complexity lies in the routines to determine the optimal server for a particular request. Furthermore, the approach lies well within our design parameters; it does not require a modification of the HTTP protocol, is reasonably efficient, and is able to support sophisticated optimization algorithms. The primary disadvantage of URL redirection in practice is the added overhead of an additional connect/pass request/parse/respond cycle after the redirection occurs. We will show that such overhead is more than negated by improved performance overall.

4.2. The processor assignment of ADL requests

In [1], we designed an algorithm that decides the routing for a general HTTP request. In this subsection, we discuss the strategies for the ADL system.

In the previous work on load balancing (e.g. [13]), usually one factor (CPU load) is considered. A processor can be classified as lightly loaded and heavily loaded based on the CPU load. One purpose of such a classification is to update load information *only* when a classification changes. Such a strategy reduces unnecessary overhead. In our problem context, it is hard to classify a processor as heavily or lighted loaded since there are several load parameters. A processor could have a light CPU load but its local disk may receive many access requests from the network file system.

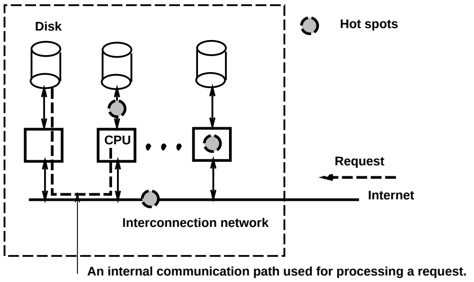
Disk
Hot spots
CPU
Request
Internet
Interconnection network
An internal communication path used for processing a request.

Figure 5. Resource hot spots monitored.

As listed above, several system load parameters affect such a scheduling decision and should be considered together. Figure 5 shows hot spots in system resources to be monitored and depicts a communication path used in processing an ADL request. This path starts from one CPU and goes through the local interconnection network and then through another disk (remote file accessing). We discuss two cases in explaining how the aggregated resource load factors affect the choice of processor assignment. Assume that a request is to access a file on the disk of Processor A (called disk A in short).

- Given a slow interconnection network, this request should be assigned to processor A if A is not heavily loaded. If this request is assigned to another processor B, the remote file access from B to disk A takes a long time, which slows down the response performance.

But if processor A is heavily loaded, then this request should be assigned to a node whose CPU is lightly loaded since computation cost takes a part of the response time. Also if disk A's channels are fully occupied, the available bandwidth may be smaller than the network bandwidth. In this case disk locality becomes less significant, since remote accessing speed is not the key factor dominating the performance.

- Given a fast interconnection network, remote access will not create a bottleneck for request processing, thus a processor which has a light CPU load should be selected. Of course the load of network, disk channels and CPUs should be consistently monitored to let the scheduler be aware of the change.

Multiple system load parameters can be used together in deciding the assignment of a request. Since our goal is to minimize the response time for each request, we design the decision-making heuristic based on the following estimated response time function for processing each request:

$$T_s = T_{redirection} + T_{data} + T_{CPU} + T_{net}$$

$T_{redirection}$ is the overhead to redirect the request to another processor, if required. T_{data} is the time to transfer the required data from the disk drive, or from the remote disk if the file is not local. T_{CPU} is the time to fork a process and perform disk I/O to handle a HTTP request, plus any known associated computational cost. In the ADL environment, we identify a set of wavelet-related functions and characterize their CPU demands. For example, accessing a wavelet data subregion costs CPU cycles proportional to the subregion size in the ADL implementation. T_{net} is the cost for transferring the processing results over the Internet. Since we are interested in getting the result out of the server as soon as possible, T_{net} is considered constant in our scheduling scheme. Furthermore, this time is likely to be the same across the set of nodes and can be ignored.

Having determined the estimated time for each node to fill the request using the formula above, the broker selects a node with the minimum estimated completion time. If the chosen node x is not the processor for this broker, the request is redirected to x. At node x, this request is processed in the normal HTTP manner, with any CGI's executed as needed.

It should be noted that it is not easy to model the cost associated with processing an ADL request accurately. We still need to investigate further the design of such a function. Our experiment shows 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 acceptable performance based on such a heuristic function, adapting to dynamically-changing system resource loads. In our experiments, we show that the overhead for scheduling and system monitoring is insignificant compared to the system resources used for request fulfillment and other activities.

5. Experimental Studies

Our primary experimental testbed consists of a workstation cluster (Meiko CS-2) at UCSB. Each node has a scalar processing unit (a 40Mhz SPARC Viking chip) with 32MB of RAM running Solaris 2.3. We mainly use six CS-2 nodes, each of which is connected to a SCSI-II 1GB disk on which test data files reside. The reason that we use 6 Meiko nodes is that these were the available resource with disks dedicated to our research. Under these configurations our experiments still demonstrate the scalability of the system and effectiveness of our scheduling techniques. Disk service is available to all other nodes via NFS mounts. These nodes are connected via a modified fat-tree network called Elan with a peak bandwidth of 40MB/s. The complete ADL SWEB server is based on NCSA httpd 1.3 source with extensive modifications to support scheduling functionality. Each custom client accesses the broker, and is then redirected (if necessary) to the appropriate node to fulfill the request. All software uses the sockets library built on TCP/IP. Because current Meiko CS-2 communication routines are not optimized for TCP/IP we were only able to achieve approximately 5-15% of the peak communication performance on the Meiko.

We consider two major types of requests for the ADL: file fetches (e.g. thumbnails) and wavelet data retrieval for multi-resolution/subregion browsing. The second type usually involves a certain amount of CPU activity in locating appropriate image data and performing on-the-fly compression. We believe that these types of requests will consume the majority of resources within the current ADL WWW prototype. As we discussed before, direction of database queries (farmed out to a dedicated database machine) and dynamically generated interface pages for the client (which may prove significant) consume other CPU cycles. We will address these activities in our future work, but we expect that our results in this paper shall be valid for those activities as well.

Our clients were primarily situated within UCSB. This was due to the instability of available Internet bandwidth making it difficult to obtain consistent results. Furthermore, our goal is to get the data out of the server as fast as possible, and providing bandwidth is outside the scope of our research. We expect that over the next few years, the available bandwidth of the Internet will increase substantially and will be comparable to our current local environment.

It should be noted that the results we report involve the average performance of multiple iterations over an extended period. The test performance is affected by dynamically-changing system loads since the machines are shared by many active users at UCSB.

We report our experiments to examine the performance of our system in the following aspects: scalability of overall performance, the effectiveness of scheduling strategies, and a discussion of scheduling overhead costs imposed.

5.1. The maximum number of requests per second

The first experiment was run to determine how many requests per second could be processed in delivering regular files such as thumbnails or text. This depends on the average file sizes requested and the number of nodes. The maximum number of serviced RPS (or MRPS in short) is determined by fixing the average file size and increasing the RPS until requests start to fail, which indicates that the system limit is reached. The duration of the test which simulates the burst of simultaneous requests also affects the experimental results. Requests arriving in a short period can be queued and processed gradually. But requests continuously generated in a long period cannot be queued without actively processing them since new requests are continuously arriving. We have chosen 30 seconds as the test duration since in practice bursts of requests arrive periodically.

File sizes in bytes	1K	1.5M
Single server	45	9
Multi-node server	82	45

Table 2. MRPS for a test duration of 30s on six nodes clustered by Meiko CS-2 Elan.

Table 2 shows the MRPS numbers obtained for a test duration of 30 seconds on the 6-node Meiko. For small files, the MRPS can reach 45 for a single node, but only 9 for 1.5MB files. We also conducted a test for a duration of 120 seconds, in which MRPS drops to 4 for a single node. The multi-node server can significantly speedup the MRPS as shown in Table 2. We also tested the MRPS on the workstations clustered by the Ethernet. Effective bandwidth of our Ethernet is much smaller than the CS-2 Elan network, the MRPS for processing 1.5 MB files is about twice as small as on the Meiko.

No. servers	1	2	4	6
MRPS	2	4	8	12

Table 3. MRPS for accessing 512×512 image subregion data at a duration of 30s on Meiko.

File accesses do not involve additional computations on the server. Accessing subregion data involves some computation to locate the correct location of data in compressed large images. We examine the performance of accessing 512×512 subregion data from a set of $2K \times 2K$ map images at the server site and delivering the results to the client for constructing a 512×512 image. Figure 3 shows the MRPS for a duration of 30 seconds on Meiko for such browsing activities. Linear speedups are obtained using multiple nodes. Again, it should be noted that the single node performance reaches 2 MRPS for a test of 30 seconds. For a longer test, the requests cannot be queued and thus the MRPS drops.

5.2. Response time and drop rates

We examine the response time when we vary the number of server nodes. The system starts to drop requests if the RPS reaches the limit. For small file requests (1K) with RPS=16, the multi-node server performs much better than one-node server but the response remains constant (around 0.5 second) when using 2 or more processors, because none of the theoretical or practical limits on bandwidth or processing power have been reached.

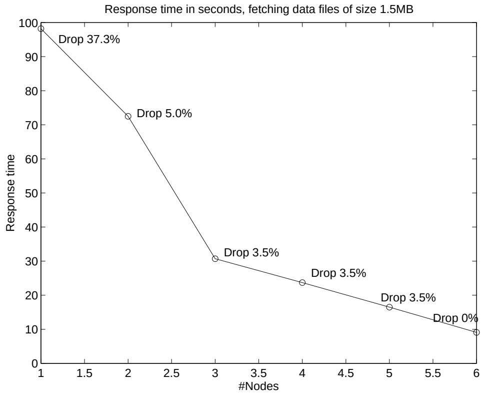

Figure 6. The response times in seconds and drop rates for Meiko. RPS=8; 30 seconds.

Figure 6 shows the response times for delivering a set of image data files of size 1.5MB with RPS=8 when we vary the number of server nodes. When the number of processors increases, the multi-node system provides substantially better performance. Under especially heavy loads, which tend to occur during peak hours at popular sites, a single server tends to drop almost half or more of the connections made, whereas a multi-node server might have a larger overall average response time but fill every request. From Figure 6 a superlinear speedup can be inferred. We have also tested in the Ethernet implementation and have similar super-linear speedups. For a sequence of wavelet-based requests (accessing 512×512 subregions from $2K \times 2K$ satellite images) we again note a super linear speedup as shown in Figure 7.

The super linear speedup in terms of response times reflect the fact that the total size of memory in the multi-node server is much larger than on a one-node server. Furthermore, the multi-node server accommodates more requests within main memory while one-node server spends more time in swapping between memory and the disk. The network overhead for accessing remote files is distributed among multiple nodes rather than concentrated at a single node.

5.3. The effectiveness and overhead of scheduling

Our scheduling strategy takes into consideration the locality of requested files, and also the current resource loads. We compare our approach with a strategy that uniformly distributes the requests

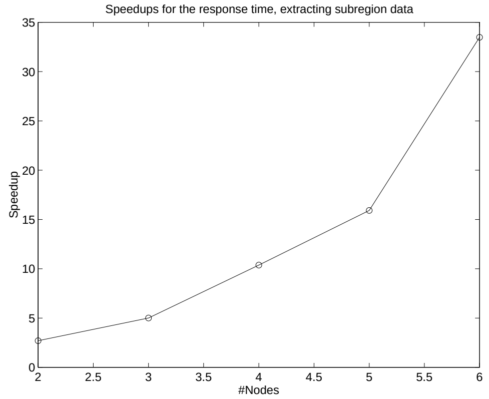

Figure 7. Wavelet image extractions, speedup for response times. RPS=3, 30sec. 0% drop rate for ≥ 2 nodes.

to nodes (called round robin). Through such a comparison, we examine the benefits of scheduling in an ADL environment with heterogeneous computing and I/O activities.

We tested the ability of the system to handle ADL requests for accessing image and text files with mixed sizes from 100 bytes to 1.5MB. Fig 8 shows the performance improvement in terms of response times. For heavily loaded situations, our scheduling strategy considers the aggregative impact of system resources over the ADL operations and has an advantage of 18-54% over round robin for heavily-loaded situations with varying request requirements. For lightly loaded cases, the round robin performs well since CPU/disk/network resources are not saturated. For such cases, load imbalance does not affect the performance of processing. The added overhead in our strategy only slows down transfers by 1%, which is insignificant. A series of tests with wavelet subregion image data accesses were performed, also indicating similar results.

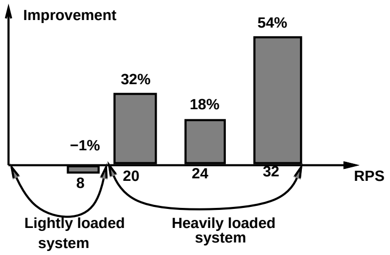

Figure 8. Effectiveness of scheduling. Improvement ratio of intelligent scheduling over round-robin. Meiko, 30 sec.

In order to determine the overhead of scheduling imposed on the system, we instrumented our code to determine the overhead cost for a request. For the case of fetching a 1.5MB image file over a fairly heavily loaded system, the total time from request initiation to receiving the last byte of information was approximately 5.4 seconds. Of that time, the server spent 4.9 seconds transmitting data, with approximately 10% of the total time spent acquiring the connection, handling the redirection, and spent merely 0.005 seconds in the scheduling algorithm. The algorithm requires approximately 0.004 seconds for analysis when the URL has not been previously redirected, and about 0.001 seconds when it has. For wavelet subregion image extraction, the numbers are virtually identical, with the exception that the time spent actually fulfilling the request is typically larger. The results indicate that the overall overhead introduced by the scheduling algorithm and load monitoring is insignificant.

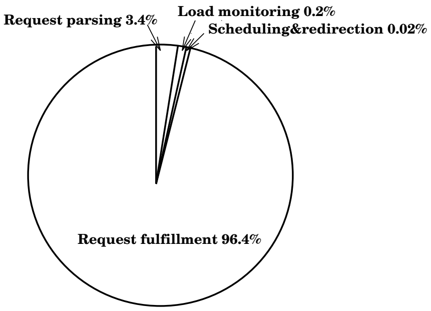

Figure 9. CPU time distribution at a server node. The test is for 512×512 subregion extraction.

We also investigated the distribution of CPU cycles at the server site to identify the amount contributed for scheduling decisions and resource information collection. Figure 9 shows the server load distribution for processing requests that access subregions of $2K \times 2K$ images from the ADL map collection in the previous experiment (see Figure 7). 3.4% of the CPU is used for parsing the HTML commands, but less than 0.2% time is used for collecting load information and making scheduling decisions. We investigated the overhead for fetching files, and the percentage of overhead for load information collection and scheduling again is very small (less than 0.1%).

6. Related Work

Numerous other initiatives to create high-performance HTTP servers have been reported. The *Inktomi* server at UC Berkeley is based on NOW technology [3], and the research focus is on information searching and indexing. NCSA [10] has built a multi-workstation HTTP server based on round-robin domain name resolution (DNS) to assign requests to workstations. The round-robin technique is effective when HTTP requests access HTML information of relatively uniform size and the load and computing power of workstations is relatively comparable. Our assumption is that the computing power and other machine resources can be heterogeneous. They can be used for other computing needs, and can leave and join the system resource pool at any time. Thus scheduling techniques which are adaptive to the dynamic change of system load and configuration are desirable. Heterogeneous computing is studied in [15]. We have not addressed scheduling on processors with different architectures in this paper; however, our techniques can be extended to such cases.

Our dynamic scheduling scheme is closely related to the previous work on load balancing on distributed systems, for which a collection of papers is available in [13]. In these studies, task arrivals may temporarily be uneven among processors and the goal of load balancing is to adjust the imbalance between processors by appropriately transferring tasks from overloaded processors to underloaded processors. The task resource requirements are unknown, and the criteria for task migration are based on a single system parameter, i.e., the CPU load. We call them single-faceted scheduling strategies. In WWW applications, there are multiple parameters that affect the system performance, including CPU loads, interconnection network performance and disk channel usages. The optimal HTTP request assignment to processors does not solely depend on CPU loads. Thus we develop a multi-faceted scheduling scheme that can effectively utilize the system resources by considering the aggregate impact of multiple parameters on system performance. In [7], resource requirements are predicted and suggested to guide the load sharing. They discuss multiple factors, but utilize only the CPU factor in predicting response times. With ADL applications we can take advantage of specific domain knowledge to accurately predict multiple resource requirements for each request.

7. Conclusions and Future Work

We have discussed issues involved for developing a scalable DL system on the WWW. Multi-resolution and subregion image browsing substantially reduces the network bandwidth requirements, but imposes additional data indexing complications and computational demands at the server site. We have implemented a multi-node scheme to strengthen the processing capabilities of the ADL server. Our experiments indicate that our scheduling scheme effectively monitors and utilizes the multiple system resources, while the overhead required to support such a scheme is extremely low.

In future work, we plan to migrate the multi-node scheme into the production environment as the primary server for the Alexandria Digital Library project. We feel this will introduce many interesting research problems relating to scheduling wildly varying demands in a heterogeneous environment. We will also investigate the incorporation of other Alexandria-specific functions such as map image querying, searching and on-the-fly HTML generation.

Acknowledgments

This work was supported in part by funding from NSF, ARPA, and NASA under NSF IRI94-11330 and a startup fund from University of California at Santa Barbara. Thanasis Poulakidas and Ashok Srinivasan provided the wavelets implementation and compression data used. We would like to thank the Alexandria Digital Library team for many valuable discussions and suggestions. We would also like to acknowledge the student researchers who contributed to this project, including Vegard Holmedahl and David Watson.

References

[1] D. Andresen, T. Yang, V. Holmedahl, O. Ibarra, SWEB: Towards a Scalable WWW Server on MultiComputers, *Proc. of Intl. Symp. on Parallel Processing*, IEEE, April, 1996. HTTP://www.cs.ucsb.edu/Research/rapid_sweb/SWEB.html.

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

[3] E. Brewer, Personal communication, http://inktomi.berkeley.edu, Jan., 1996.

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

[5] C. Fischer, J.Frew, M. Larsgaard, T.R. Smith and Q.Zheng. Alexandria Digital Library: Rapid Prototype and Metadata Schema. Proceedings of Advances in Digital Libraries 1995, Vol. ,pp. , 1995.

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

[7] K. Goswami, M. Devarakonda, R. Iyer, Prediction-based Dynamic Load-sharing Heuristics, *IEEE Trans. on Parallel and Distributed Systems*, 4:6, pp. 638-648, 1993.

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

[9] Lycos Usage: Accesses per Day, http://lycos.cs.cmu.edu/usage-day.html, June 17, 1995.

[10] 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.

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

[12] S. Prabhakar, D. Agrawal, A. El Abbadi, A. Singh, T. Smith, Content based placement and browsing, CS Tech Report, UCSB, 1995.

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

[14] T. R. Smith and J. Frew. Alexandria digital library. *CACM*, 38(4):61-62, April 1995.

[15] R. Wolski, C. Anglano, J. Schopf, F. Berman, Developing Heterogeneous Applications Using Zoom and HeNCE, *Proceedings of the Heterogeneous Computing Workshop, HCW '95*, pp. 12-21, Santa Barbara, CA, IEEE, April, 1995.