Scalable Web Server Design for Distributed Data Management

Scott M. Baker Bongki Moon

Technical Report 98-8

Abstract

Traditional techniques for a distributed web server design rely on manipulation of central resources, such as routers or DNS services, to distribute requests designated for a single IP address to multiple web servers. The goal of the Distributed Cooperative Web Server (DCWS) system development is to explore application-level techniques for distributing web content. We achieve this by dynamically manipulating the hyperlinks stored within the web documents themselves. The DCWS system effectively eliminates the bottleneck of centralized resources, while balancing the load among distributed web servers. DCWS servers may be located in different networks, or even different continents and still balance load effectively. DCWS system design is fully compatible with existing HTTP protocol semantics and existing web client software products.

August 1998

Department of Computer Science
The University of Arizona
Tucson, AZ 85721
{bakers,bkmoon}@cs.arizona.edu

1 Introduction

With the explosive popularity of the internet and the world wide web (WWW), there is a rapidly growing need to provide unprecedented access to globally distributed data sources through the internet. Web accessibility will be an essential component of the services that future digital libraries should provide for clients. This need has created a strong demand for database access capability through the internet [19], and high performance scalable web servers [16, 23]. As most popular web sites are experiencing overload from an increasing number of users accessing the sites at the same time, it is desired that scalable web servers should adapt to the changing access characteristics and should be capable of handling a large number of concurrent requests simultaneously, with reasonable response times and minimal request drop rates.

Any distributed system requires some techniques to distribute the work load across multiple server computers. In some scalable web server systems [22], this work load distribution is done at the *packet level* by using some form of custom routing. The router must intercept inbound packets, translate the incoming IP address into an address of a server computer, and place the packet out on a local area network. The packet router is expected to be a bottleneck as all packets must pass through it.

On the other hand, the work load distribution may also be achieved by using a *custom domain name service (DNS)* to rotate the one host name amongst many IP addresses [16]. However, this is a very coarse-grained solution since DNS mappings may be cached by multiple levels within the hierarchy of services. DNS mappings do have a time-to-live (TTL) parameter which specifies how long the information is considered valid. A trade-off exists with the TTL parameter in that a low TTL will yield more control over the distribution at the sacrifice of creating a bottleneck on the DNS server, while a high TTL will yield less control over the distribution while causing less load on the DNS server.

A collection of web documents may be viewed as a directed graph, where each document is a node and each hyperlink (or image reference) is a directed link from one node to another. If there is a way to distribute this graph amongst many server computers in such a way that the load is evenly distributed despite the dynamically changing web access patterns, then the problem of load balancing, one of the most important issues of creating a distributed web server, has been solved. Our solution will take this *graph-based* approach and will be based on the hypothesis that most web sites only have a few *well-known entry points* (*e.g.*, `www.washingtonpost.com`) from which users start navigating through the site's documents.

The proposed solution is to dynamically modify the web documents to change their hyperlink connectivity, and thereby distributing the document graph adaptively amongst several servers. The dynamic modifications will be performed automatically by the web servers and will require no user intervention. All the well-known entry points will be maintained at the *home servers* where the web documents originate, while less known internal documents may be migrated to alternate server computers which we call *co-op servers* for load balancing purposes. The home servers and co-op servers can serve collectively as a *distributed cooperative web server (DCWS)* for the need of web request processing with great flexibility and scalability.

There may be many possible situations where the distributed cooperative web server can be deployed to handle highly fluctuating web requests. Any stand-alone web server can be supported by several computers connected together by a local area network in the same organization. When the stand-alone web (home) server is overloaded, some of the computers can act as co-op servers by off-loading documents from the home server and delivering them on behalf of the home server. For another example, two or more departmental web server machines which work independently in the usual operational mode, can become a distributed cooperative web server; since the relative load may be different on each departmental web server depending on the time of year, project deadlines and so on, any of the lightly loaded servers can be a co-op server for any of the heavily loaded servers. The server machines can be geographically distributed. If an organization runs a number of independent web servers for branches in the east and west coasts of the United States and Asian and European countries, then the DCWS approach enables the web servers to adapt to the changes in geographic distribution of document requests and the changes due to different time zones. It also enables the web servers to take advantage of geographic caching of documents [5].

The distributed cooperative web server solution poses the following benefits over traditional systems based on packet-level manipulation, or domain name services (DNS) and distributed file systems:

- Network or packet level manipulation is not necessary. There is no entity (such as a router) that needs to touch every packet that is transferred between client and server. This eliminates a significant bottleneck present in traditional systems.

- Instead of implicit load balancing by using custom DNS servers, the cooperating servers make use of the connectivity of hyperlinks to directly control load balancing at the fine-grained level of documents.
- The cooperating servers do not need to be located within the same administrative domain or local area network. They may be geographically distributed and can distribute network traffic over multiple networks.
- Adding a new server is easy, flexible, and cost effective. Any available machine may be added as a cooperating server, without consideration as to the location of the machine relative to other existing servers.

In this paper, we present the design principles of the DCWS system and the detailed issues of its prototype implementation. We also demonstrate the scalable performance of the DCWS system by extensive experiments with real-life data sets. The rest of the paper is organized as follows. In Section 2, we briefly review related work for building scalable web servers. Section 3 presents the motivations behind the development of DCWS system, and describes the primary design issues. In Section 4, we describe the metrics for selecting documents to migrate, and present the detailed process of document migration and consistency considerations. Section 5 presents experimental results to demonstrate how the DCWS system works well with real-life data sets. Finally, in Section 6, we discuss the contributions of this paper and suggest future work.

2 Background and Related Work

Various load balancing techniques based on domain name service (DNS) have been proposed in the literature. The NCSA scalable web server is built on a cluster of identically configured servers, and uses round-robin DNS scheduling and Andrew file system (AFS) for load sharing among the servers [16, 18]. The IBM scalable web server is built on an SP-2 parallel system, which is essentially a cluster of identical RS6000 workstations. The IBM web server uses a TCP router instead of DNS scheduling for improved load balancing [11], but its use is limited to tightly coupled systems such as SP-2.

The presence of heterogeneous web servers not only increase the complexity of the DNS scheduling, but also makes a simple round-robin scheduling not directly applicable. Numerous variations of the round-robin DNS scheduling have been proposed for heterogenous web servers and non-uniform client distribution. Two-tier round-robin DNS scheduling divides clients into two classes normal and hot to handle non-uniform distribution of client requests [8]. Probabilistic and deterministic algorithms based on adaptive TTL (time-to-live) approach have been proposed [7]. Lower TTL values are assigned when the DNS chooses a less capable server or an address mapping request comes from a hot client.

Another solution proposed in [4] attempts to develop a distributed scheduling heuristic based on a multi-variate cost function (CPU, disk and network utilization), which helps make the decision on task migration. Two techniques are used for load balancing: DNS rotation and HTTP URL redirection. DNS rotation is used for initial load distribution, and HTTP URL redirection is used to dynamically adjust network load based on server utilization. A potential problem with DNS rotation is the development of "hot spots", which lead to serious load imbalances. A detailed study of the techniques for an online digital library was reported in [3].

Dynamic server selection [10] is proposed as a client-based solution, in which clients automatically determine the best server for a given file without a priori knowledge of server performance. The technique relies on replication of web documents by proxy servers. It is assumed that a list of proxy servers exists which contain a given document. A hybrid buffer management algorithm [23] has been proposed to balance intra-cluster network traffic and disk access by dynamically controlling the amount of data replication. A centralized round-robin router is used to route requests amongst multiple servers.

The Cisco LocalDirector Cisco systems [22] uses a virtual server to handle incoming requests at a virtual IP address. The LocalDirector functions as an intelligent router, routing requests from the virtual IP address to physical servers at real IP addresses. It is intended to be a general purpose solution, capable of handling other services in addition to the web. No discussion is given in the white paper as to what extent the LocalDirector is a bottleneck of the system.

Fast Packet Interposing is a user-level technique developed in [2] and is used by the MagicRouter to distribute load. The MagicRouter is used to make a cluster of servers appear to have a single IP address without modifications to any servers. Fast Packet Interposing is used to modify network addresses within the data packets that pass through the MagicRouter. Fault tolerance and Load Balancing are stressed by the paper. The MagicRouter is expected to be a bottleneck as all packets must arrive through it as a central resource.

A technique called dynamic packet rewriting (DPR) [6] is used to distribute load. DPR attempts to route requests at the IP level, by manipulating the methods in which an IP address is mapped to a host. It is a distributed algorithm, and attempts to eliminate the bottleneck of a centralized solution, such as a centralized round-robin router. Each host acts as both a web server and a packet-level router for incoming requests. DNS rotation is used as an initial partitioning method to achieve a rough distribution.

3 Design Principles

In this section, we present the motivations behind the development of distributed cooperative web server (DCWS), and describe the primary design issues to construct a flexible, fully symmetric and scalable web server that achieves dynamic load balancing.

3.1 Entry-points Hypotheses

Most web sites only have a few well-known entry points for the site. A well-known entry point is a URL which is published to the rest of the world. For example, typical digital library applications such as newspaper sites, article archives and customer information services would publish the URL of their index pages to the world rather than publishing the URL of every single page in the sites. Specifically, the DCWS solution proposed in this paper is based on the following observations and generalizations about the organizations of and access patterns to the web documents:

- Web documents are either listed (well-known entry points) or not. Web pages that are not well-known are usually only accessed by a particular client by first accessing some well-known page and then following a path of hyperlinks from the well-known page to the destination page.

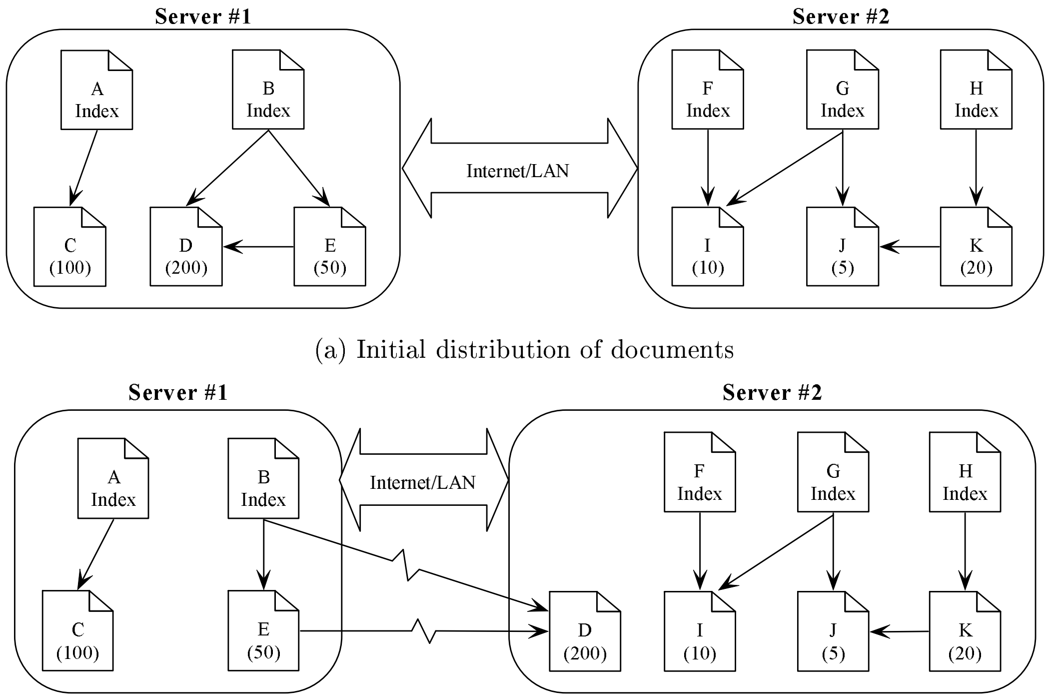
- The URL's of images embedded within a document are seldom published. Users need not know the URL's of the embedded images because the images are fetched automatically along with the corresponding web document without user intervention. Furthermore, images constitute a large portion of web bandwidth due to their comparatively large size.

- The use of frames in most modern web sites promotes a well-known *frame template* that is published along with several seldom-published internal frame pages. The frame template is usually small and easily hosted by the home server, while the internal frame pages may be large and can be migrated to other servers for load balancing purposes.

- In this setting, users do not care what the text of the URL is for internal pages within the site, as they primarily access the site through the well-known entry point.

There are a few exceptions to the above rules that must also be considered. Most notable are web search indexes and user bookmarks. Web search indexes are maintained by large search engines such as AltaVista and Infoseek. These search engines may make public any URL's contained within a certain web site. User bookmarks are URL's manually stored by end-users on their computers. Thus any user may potentially bookmark any page. However, it should be noted that there are many ways to force them to come in the front door only; it can be done either through cookies, or through adding tokens or sequence numbers to the URLs (generated by CGI scripts of JavaScript).

The proposed solution will assume that most requests follow the common pattern of arriving at a well-known entry point, while a minority of requests may be bookmarks or search site retrievals, and thus the solution will attempt to optimize the common case while providing only a minimal penalty in the less-common case.



(a) Initial distribution of documents

(b) After a document `D` is migrated from Server #1 to Server #2.

Figure 1: Illustration of document migrations for load balancing

3.2 Document Migration

All documents originally reside on a *home server*. The home server is where the administrator or authors have placed the documents during the process of creating them. A permanent copy of the original document will always be maintained on the home server for consistency and robustness purposes. All well-known entry points will be maintained on the home server so that the users may see a consistent view of the web site. Documents or images other than the well-known entry points may be migrated to *co-op servers*. A co-op server is another server computer which has been designated as a server which will share the load of the home server. [1]

Pertaining to the document migration, it is important to note that the document hyperlinks are modified in such a way that the load is balanced among the home and co-op servers by redirecting user requests from one to another. In Figure 1, for example, documents `A` - `E` and `F` - `K` were initially hosted by two different servers. The numbers in parentheses presents the load (*i.e.*, hits) associated with individual documents. Since the first server was overloaded, the document `D` was chosen to be migrated to the second server. In this example, the second server became the co-op server for the document `D`.

We adopt *lazy migration* policy in an attempt to minimize overhead incurred by physical data migration. Further details about the process of document migration will be discussed in Section 4.

3.3 Document Graphs for Load Balancing

The distributed cooperative web server (DCWS) is designed to be fully symmetric, in the respect that each server may be both a home server for its own documents as well as a potential co-op server for sharing the load of some other home server. There are two key data structures that must be managed by the DCWS

[1] In principle, a document can be replicated to more than one co-op servers, but the current prototype implementation allows each document to be migrated to only one co-op server.

Name	Location	Size	Hits	LinkTo	LinkFrom	Dirty
A	#1	-	-	C	nil	0
B	#1	-	-	{D,E}	nil	1
C	#1	-	100	nil	A	0
D	#2	-	200	nil	{B,E}	0
E	#1	-	50	D	B	1

Figure 2: LDG entries for the documents on the Server #1 in Figure 1(b)

servers in order that they may share and balance the load amongst participating servers. First, each server must have the information of the local documents and the link structure among the documents. All the information is stored in a *local document graph*, and each server is responsible for maintaining the graph for any documents stored within it. Second, the servers must globally communicate load information amongst one another so that intelligent load-balancing decisions may be made. This information is global in nature, but each node maintains its own local view of the global state. The best-effort global load information is stored in a *global load table* on each server machine.

Local Document Graph The local document graph (LDG) consists of a set of tuples

`(Name, Location, Size, Hits, LinkTo, LinkFrom, Dirty).`

The `Name` field is simply the name of the document, as it is requested by the user. It is also directly related to the name of the file on the server's local disk, so that the server knows where the contents of the document are located. The `Location` indicates which server presently is hosting the document, whether it be the home server where the document originated, or a co-op server that the document has been migrated to. The `Size` and `Hits` are used for load balancing computations, in order to determine which documents should be migrated. The `LinkTo` is a list of documents that the current document has hyperlinks pointing to; the `LinkFrom` is a lists of documents that have hyperlinks pointing to the current document. The `Dirty` bit is used to indicate whether some of the tuple's `LinkTo` documents have been migrated, thus requiring the system to regenerate the document with some altered hyperlinks. Figure 2 shows the local document graph entries stored in the first server after the document `D` is migrated to the second server. Note that some information such as document sizes is left out for visual clarity.

The local document graph is computed upon initialization of the web server by scanning its disk and parsing the documents. It is intended to be a dynamic structure and can be modified over time if the content of pages is changed by an administrator. Section 4 describes how the local document graph is updated when a document is migrated. A hash table is utilized to quickly find a tuple given the document name. It is important to optimize with a hash table because retrieving the tuple is necessary for each request that the server processes. The local document graph is assumed to be small enough that it can be stored entirely in memory. If this is not the case, then it should be a straightforward process to store the structure on disk and use memory as a cache for frequently accessed tuples.

Global Load Table The global load table (GLT) stores the overall state of the server group. Although the load information is global in nature, each server maintains a local copy of the load information and attempts to maintain data using a best-effort consistency mechanism. The global load table consists of a set of tuples

`(Server, LoadMetric).`

The `Server` is the name (or IP address) of the server computer. The `LoadMetric` is some measurement of the load that the server is experiencing and is used for load balancing purposes. For example, the total number of requests per minute could be used as a satisfactory load metric. Each server may trivially compute its own load information tuple by recording its `LoadMetric` as user requests bombard the server. The interesting point is how load information is communicated from one server to another.

Since the network is already presumably filled with many user requests and responses, it is desired not to initiate any additional data transfers simply for the purpose of communicating load information. Such a solution would be wasteful in a system where network bandwidth is an important resource. Instead, the solution that was chosen was to *piggyback* the load information onto existing HTTP transfers. The idea of piggybacking information has been used for cache coherency by server invalidation [17].

The HTTP protocol allows for inserting *extension headers* into the existing protocol semantics.[2] Extension headers may be included in both the HTTP request (client to server) and the response (server to client). Thus, it is possible to insert an arbitrary amount of bi-directional information into an existing HTTP transaction. Amongst the DCWS servers, transfers are already occurring frequently to migrate documents between the servers, and these transfers provide an excellent opportunity to also communicate the load information by piggybacking. Thus, no additional communication channels are required for the means of communicating load information.

In the unlikely case that load information is not being communicated frequently enough, then it is possible to insert an artificial transfer to communicate load information. This would incur additional overhead since the transfer would not have occurred normally. A special *pinger* thread is present to watch for out-of-date information and automatically generate artificial transfers to bring the information up to date. The pinger thread is discussed fully in Section 4.5.

Figure 3 depicts the functional modules and data structures of the DCWS system, and illustrates access requests from web clients and interaction between home and co-op servers to process the requests. Further details of the multithreaded implementation of the DCWS prototype will be described in Section 5.

3.4 Document Naming Conventions

For a document located on its home server, the name of the document can be arbitrary. However, once a document has been migrated to a co-op server, a naming convention must be adopted so that the co-op server is able to determine which home server the document was migrated from.

We propose a naming convention for migrated documents. Assume that a document `foo.html`, which has the following

$$\texttt{http://h\_name:h\_port/dir}_1\texttt{/dir}_2\texttt{/.../dir}_n\texttt{/foo.html}$$

as its generic URL, has been migrated from its home server to a co-op server. Then, the migrated document will have a URL of the following form:

$$\texttt{http://c\_name:c\_port/\~{}migrate/h\_name/h\_port/dir}_1\texttt{/dir}_2\texttt{/.../dir}_n\texttt{/foo.html}$$

When a URL with "˜migrate" as the first path component arrives to a co-op server, it can recover the original URL of the document by stripping off everything up to the "˜migrate" component, and by changing a few punctuation separators, reconstruct the original URL.

4 Process of Document Migration

In this section, we describe the metrics for selecting documents to migrate, and the detailed process of migrating documents including the lazy migration policy, hyperlink modification, load information update, and consistency considerations.

4.1 Metrics for Selecting Documents to Migrate

There may be several conditions and factors that should be considered in determining which documents be migrated. Among others, we have chosen a few criteria for selecting documents to migrate such that load balancing can be achieved with a small number of document migrations and low overhead. The algorithmic procedure for the document selection is presented in Algorithm 1 in Figure 4.

[2]These extension headers are ignored by any server which does not understand them, but may be interpreted by a server that does understand the extension header [13].

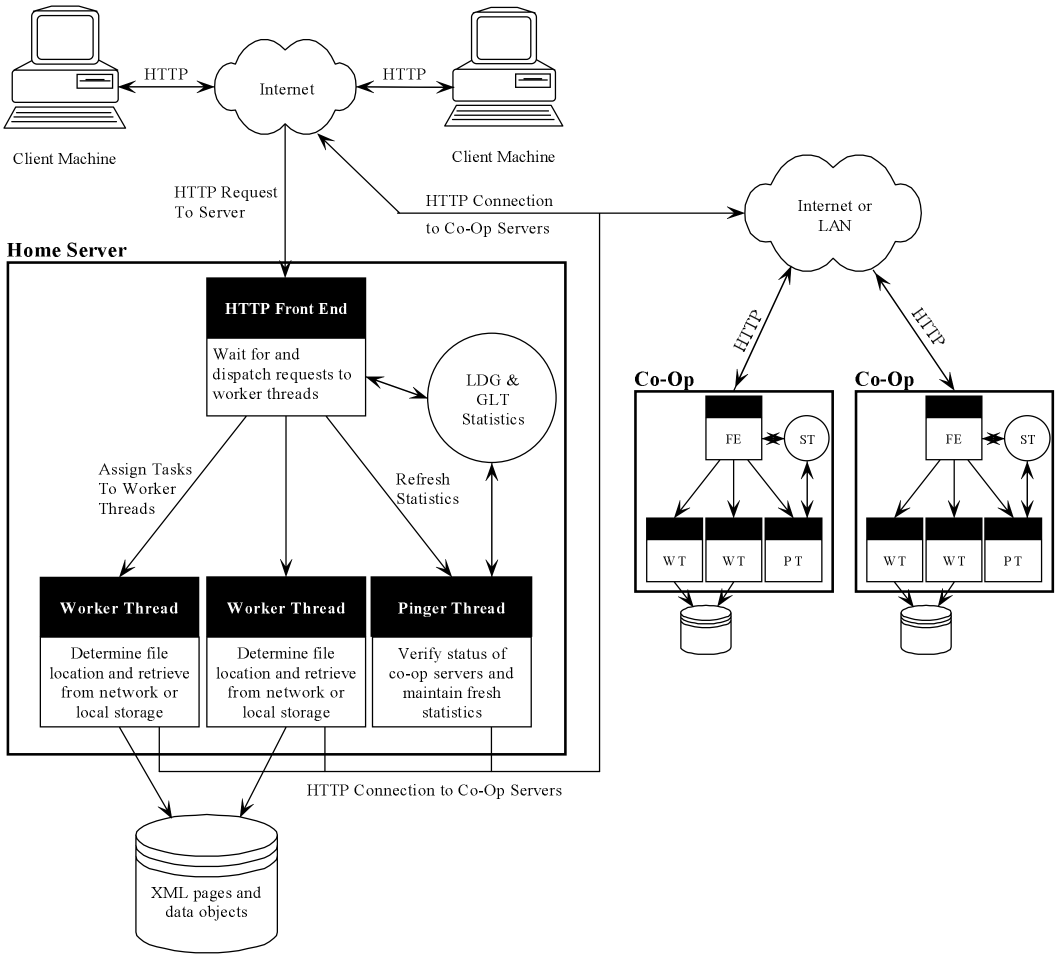


Figure 3: Functional diagram of DCWS

It is typical that well-known entry points will provide users with an external view of the server. Users will use the well-known entry points to gain access to the server. If any of the well-known entry points is migrated to a co-op server, the home server should redirect every user request to the co-op server. Thus, step 2 is necessary to maintain a consistent view of the home server and to avoid burdensome request redirections.

Step 3 is important because we want to balance the work load by migrating as few documents as we can. In other words, the access frequency of the document should be high enough to justify the migration process because migrating a document that receives only a few hits does not do much good for load balancing.

If a document is migrated from its home server to a co-op server, all the documents pointing to the migrated document (*i.e.*, the documents in the `LinkFrom` list) must be modified to update the hyperlinks and connectivity information. Step 4 seeks to minimize additional network traffic that would be required to update the hyperlinks in the `LinkFrom` documents on remote servers. Step 5 allows a document that points to a minimal number of `LinkTo` documents. This promotes future compliance with the goal of step 4.

4.2 Lazy Migration

If it is determined that one or more document migrations should occur, then the following process is used:

- The server with the lowest `LoadMetric` value is selected from the global load table. This can be done

Algorithm 1 *Document Selection for Migration*

Input: Given a local document graph of a home server, and a threshold $\mathcal{T}$of load.

Output: This algorithm selects a document to be migrated to a co-op server.

1. Let the candidate document set $\mathcal{C}$ be a set of all the documents in the graph.
2. Remove all the well-known entry points from $\mathcal{C}$. if $\mathcal{C}$ is empty, return `nil`.
3. Remove documents from $\mathcal{C}$ if their load (*i.e.*, `Hits` value in the tuple) is less than the threshold value $\mathcal{T}$. if $\mathcal{C}$ is empty, reset it to the previous set and repeat this step with reduced value of $\mathcal{T}$ until $\mathcal{C}$ becomes non-empty.
4. Select a document (or more) pointed to by a minimal number of `LinkFrom` documents that do not reside on the home server.
5. If two or more documents are selected in step 4, pick one that points to a minimal number of `LinkTo` documents.

end

Figure 4: Document Selection for Migration

with a simple scan of the table. This server will become a co-op server for the home server and will host the migrated documents.

- The local document graph is updated accordingly. Specifically, the `Location` field of the tuple for the document is modified to reflect the new location. For each document referenced by the `LinkFrom` field of the tuple, the `Dirty` bit is set for that tuple. This will cause the documents referenced by the `LinkFrom` field to be regenerated next time a request arrives for one of them.

Through the above steps, the selected documents are migrated only logically. The physical document migration is deferred until it is actually required. We call this a *lazy migration* of documents.

To understand the physical migration process, it is necessary to discuss the behavior of the DCWS servers in detail. When a request arrives at a server, it may be one of two cases. Either it is a request for a document that is local to the server (*i.e.*, the server is a home server for the document), or the request is for a document that has been migrated to the server (*i.e.*, the server is a co-op server for the document).

If the request is for a migrated document, then there are two sub-conditions that exist:

1. The co-op server does not have a copy of the document on it's local disk. In this case, the co-op server must initiate a HTTP session with the document's home server to retrieve the document. Once the document is retrieved from the home server, a copy is stored on the co-op server's local disk for future purposes, and the contents are also forwarded back to the user that initiated the original request.
2. If the co-op server does have a copy of the document on it's local disk, then the document must have already been physically migrated, and the copy on the co-op server's local disk can be sent to the end user.

4.3 Document Parsing and Reconstruction

To modify the hyperlinks embedded in a document, a HTML parser builds a simple parse tree from an HTML source file of the document. Any modified links are then replaced in the parse tree, the parse tree is turned back into a stream of HTML tokens, and then written back to its HTML source file. Since parsing and regeneration of documents is expected to be a time intensive process, it is desired to postpone the process until the latest time possible in order to eliminate any redundant or unnecessary work. The `Dirty` bit of a document's tuple in the local document graph is used as an indication as to whether the document needs to be parsed and regenerated with some modified links.

When a request arrives to retrieve a document, if the `Dirty` bit is not set, then the document is assumed to be up-to-date and a copy from disk is retrieved. If the `Dirty` bit is set, then the document is considered to be outdated and will be parsed, regenerated, a new copy will be written to disk, and the `Dirty` bit will be reset.

4.4 Requests for Migrated Documents

Some provision needs to be made for requests that arrive at a home server for a document after it has been migrated. This condition may occur under a variety of circumstances:

- The user bookmarked or a search engine indexed the page when it was on the home server, before it had been migrated. Thus the user has the pre-migration address.
- The currently displayed web page on the user's computer was obtained before the migration occurred, and thus has hyperlinks embedded in it which refer to the pre-migration address.
- The user's local cache has a copy of an old document in it, which contains the pre-migration address.

The HTTP protocol has a provision to redirect a client browser by issuing a `301` response with the new address. The redirection message is small and consumes little network bandwidth. It does, however, necessitate the use of one additional connection to request the document, which will result in some extra latency to the end user.

Sending redirection responses should cause a fairly low amount of load on the server since the server does not have to fetch any information from disk. All information required to generate a redirection is contained within the server's local document graph.

4.5 Consistency Concerns and Revoking Migrated Documents

A system needs to be in place to revoke a document by returning control of that document from the co-op server to the home server. This is necessary in four cases:

- The content of the document has been changed by the document's author, and changes need to be propagated to any co-op server that may be hosting the document.
- Workload characteristics have changed, and an imbalance may have developed between home and co-op servers.
- A co-op server has crashed and is no longer hosting documents.
- The home server has crashed

The first consistency constraint can be satisfied by a *co-op server validation* approach based on timeout. The co-op servers are configured to automatically re-request documents at a specific time interval so that any change can only cause an inconsistency for the duration of the timeout. After the timeout has expired, the system is guaranteed that all copies on all co-op servers are accurate. Another timeout is present on the home server to take care of the second case. This timeout specifies the time interval at which the home server is able to abandon a migration and re-migrate the file to a different co-op server.

The third case is handled by a special "pinger" thread present on all servers. The pinger is responsible for periodically waking up and initiating an artificial HTTP request with any co-op servers that have not communicated within a certain interval. If a co-op server does not respond to several pinger requests, then it can be considered down and it's documents are recalled. The fourth case is a catastrophic event and will result in an inability to handle user requests. The system can attempt to make its best effort at handling requests since the co-op servers will have some subset of the home servers pages available. Thus, a co-op server should not throw away any data until absolutely necessary (i.e. lack of disk space) in order to make that data available in case of a home server crash.

5 Experiments

5.1 Prototype Development

The DCWS server is constructed in a modular fashion using *multithreaded* paradigm. Components include a multithreaded HTTP front-end, which is responsible for accepting and parsing requests, a worker module, which utilizes multiple threads to process and respond to requests from the front-end, and a statistics module, which is responsible for maintaining Global Load Table. The multithread support was implemented with portability as a chief concern. The server can run on Linux using the Posix pthreads library as well as Microsoft Windows products using the Win32 Thread API. The current Linux implementation has been tested using Linux kernel version 2.0.30 and pthreads version 0.5.

The multithreaded paradigm was chosen as opposed to the traditional *pool-of-processes* paradigm since a shared memory space and efficient communication were necessary to implement cooperation between the DCWS worker modules and the statistics module. The more traditional pool-of-processes approach would have made sharing statistical data such as the Local Document Graph and Global Load Table difficult and inefficient. Some popular web servers, including Apache [14], are either multithreaded or have experimental multithreaded efforts. [15, 12].

A general purpose HTML parser has been used to build simple parse trees for HTML source files. Although the parser is not optimized for the DCWS prototype, it is expected that an optimized parser would only improve performance by a constant amount and would not affect the speed up or scale up performance of the server.

5.2 Experimental Settings

Testing and benchmarks were performed on a cluster of 64 Intel Pentium workstations with 200 MHz clock rate. Each workstation has 128 MB of memory and 2 or 4 GB of disk storage. The workstations are connected by a 100 Mbps switched Ethernet network. The switch can handle an aggregate bandwidth of 2.4 Gbps in an all-to-all type communication. A server process (either home or co-op) ran on each of the Pentium workstations. Each server process contained 12 worker threads as well as a front-end thread and a pinger thread. Each home server was configured to migrate files at a maximum of one file per 10 seconds. No single co-op server was allowed to accept more than one migrated file every 60 seconds. This time interval is necessary to avoid overloading a co-op server by migrating documents too quickly, before it has a chance to adjust and recalculate its load statistics. The co-op servers were configured to validate migrated documents for consistency every 120 seconds. The pinger thread was assigned a sleep value of 20 seconds, in order to guarantee that all statistical data was accurate within 20 seconds. The server configuration parameters are summarized in Table 1.

Description	Parameter value
Number of front-end threads ($\mathcal{N}_{fe}$)	1
Number of pinger threads ($\mathcal{N}_{pi}$)	1
Number of worker threads ($\mathcal{N}_{wk}$)	12
Socket queue length for backlogged requests ($\mathcal{L}_{sq}$)	100
Statistics re-calculation interval ($\mathcal{T}_{st}$)	10 seconds
Pinger thread activation interval ($\mathcal{T}_{pi}$)	20 seconds
Co-op server document validation interval ($\mathcal{T}_{val}$)	120 seconds
Home server document re-migration interval ($\mathcal{T}_{home}$)	300 seconds
Minimum time for migrations to the same co-op server ($\mathcal{T}_{coop}$)	60 seconds

Table 1: Setting of server parameters

Client benchmark configuration A custom benchmark was constructed due to the unique property of the distributed cooperative web servers that the hyperlinks of documents may be modified dynamically.

Algorithm 2 *Custom Benchmark*

```
do forever  begin
    reset cache
    set current_url ← a randomly selected well-known entry point
    set no_steps ← random(1..25)
    for i=1 to no_steps do begin
        request a document current_url from its server if it is not in the cache.
        request all embedded images in parallel (using helper threads).
        wait until all the requested documents arrive.
        parse the document and select a new link from the document
        set current_url ← new link
    end
end
end
```

Figure 5: Outline of the simulated customer threads

Conventional benchmarks such as SPECweb96 [9] are not suitable as they are designed to request documents without regard to the hyperlinks contained within the documents.

The benchmark is intended to correspond to the real-world behavior that most clients exhibit while they are accessing the web. Web clients running browsers typically maintain a client-side cache. This client side cache effects access patterns significantly and reduces temporal locality [1]. With the custom client benchmark, we sought to simulate this caching behavior by building a client-side cache into the benchmark program. The cache is maintained for the duration of each simulated access sequence (1-25 document requests) of the benchmark and reset after each sequence.

The effects of the client cache on the DCWS system performance are expected to be two-fold: (1) Hot-spot behavior of images linked to multiple pages is reduced, and (2) redirections are increased due to increased stale link data being stored client-side. Both of these effects are real-world phenomena which increase the realism of the DCWS custom client benchmark.

The client benchmark program is multithreaded and includes one main thread to load a document and four additional threads to load images in parallel. The parallelism is intended to simulate the actions of existing web browsers which make use of multiple threads as well. Approximately eight instances of the client benchmark were configured to run on each client benchmark workstation. The number of client benchmark processes was selected to consume all available CPU and network resources of the client machine. The throughput produced per client workstation on the median dataset, LOD, was approximately 700 CPS and 1.7 Mega BPS. The LOD data set will be described later in this section. Twenty-five workstations were configured as benchmark machines. The detailed procedures that performed by the custom benchmark is outline in Algorithm 2 in Figure 5

Request drop behavior It is likely that the request arrival rate might be often higher than the request service rate. The backlogged requests are queued in the socket queue at the server, and the socket queue may grow beyond the preset maximum queue length (see Table 1). Then, the connection is dropped gracefully with a `503` error response. This is the most graceful way of dropping requests, but also the most load intensive method for servers. When a `503` is received, a client has an exponential back-off and retry to minimize server load. That is, a client thread sleeps for a second at the first drop, sleeps for two seconds at the second drop, sleeps for four seconds at the third drop, and so forth.

Data sets We have chosen four real-world data sets with different characteristics. The first three in the following are the authors' own creations, and they are available in `http://www.cs.arizona.edu/dcws`. The last data set is the Sequoia 2000 storage benchmark data [21] publicly available in `http://epoch.cs.-berkeley.edu:8000/sequoia/benchmark/`.

- *MAPUG Mailing List Archive*: The mailing list archive contains 1,534 documents, 28,998 links and 5,918 Kbytes aggregate size of all files. The mailing lists are topical discussions conducted via electronic mails. Communication is achieved by posting a message to a mail list server which forwards the message to all participants in the mailing list. Mailing lists are occasionally archived for public browsing for those who wish to read, but not participate in the discussion. The data set is mostly text, each with 4-6 bit-mapped images, which are buttons for links to the next, previous, next_thread, previous_thread, and several index pages. The bit-mapped buttons have a high request rate and are among the first pages migrated by the server.
- *SBLog Web Statistics*: The web statistics contains 402 documents, 57,531 links and 8,468 Kbytes aggregate size for all files. The statistics report contains overview index files that describe activity by date, IP address, and directory, as well as a large number of files which describe in-depth details for individual files on the web site. The data set is entirely text, except for one JPEG image, which is used to display bar graphs. This JPEG image file is extremely popular.
- *LOD Role-Playing Adventure Guide*: This data set is a graphical database for a popular computer role playing game, with 349 documents and 1433 links (240 of the 349 documents are images). About a half dozen pages consist of large tables of characters or data items with about 50 thumbnail images in each page depicting these data items. Images follow a bimodal distribution with approximately half of the images averaging 1.5 Kbytes and the remainder averaging 3.5 Kbytes. The aggregate size of the documents is 750 Kbytes.
- *Sequoia benchmark data*: The raster data for Sequoia 2000 storage benchmark contains 130 AVHRR (Advanced Very High Resolution Radiometer) image files from NOAA satellite. The images are compressed and in the 1-2.8 Mbytes range. We created an HTML front-end page to the Sequoia raster data set that includes a hyperlink to each image file..

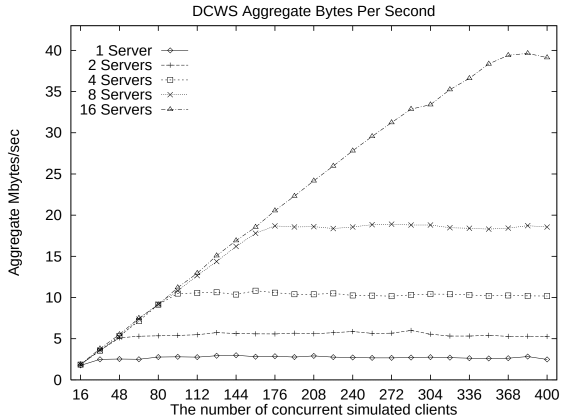
5.3 Experimental Results

Among many ways to evaluate the performance of web servers, there seems to have emerged a consensus that three most important measures are *connections per second (CPS)*, *bytes transferred per second (BPS)*, and *round-trip time (RTT)* [18, 20]. However, the third measure, round-trip time is difficult to measure for an operational web server and depends on various performance factors such as network overhead and bottleneck, which are not directly related to the web server itself. Thus, in our experiments, we have decided to use the first two measures to evaluate the performance and scalability of the DCWS prototype system.

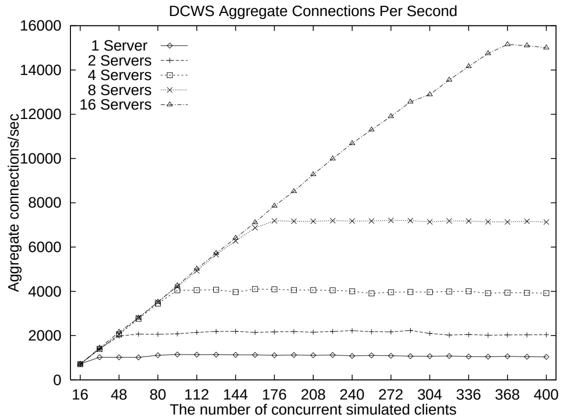
Peak load One of the most critical challenges for web servers is to deal with the peak load. To accurately measure the peak performance of the DCWS, we averaged the bytes per second (*i.e.*, throughput) and connections per second measures for a fixed number of concurrent simulated clients (*i.e.*, threads running on clients workstations). Increasing the number of concurrent clients from 16 up to 400, we repeated the same experiment and obtained averages of the measures. The third data set (LOD Role-Playing Adventure Guide) was used for this set of experiments, because this data set does not develop any hot spots and is suitable to demonstrate the close-to-linear scalability of the DCWS system. We will show the performance effects of hot spots in another set of experiments done with other data sets.

Figure 6(a) and Figure 6(b) show BPS and CPS measures, respectively, as compared to number of concurrent clients, with different numbers of servers being used. In both figures, the performance measures increased almost linearly until the peak was reached. After the peak was reached, the measures remained stable, presumably due to dropping excessive requests beyond the server capability. On the other hand, it is quite obvious that the DCWS prototype performs in a scalable way. Whenever the number of servers was doubled up, the peak performance was improved proportionally. For example, with 8 servers, the peak performance of about 18.6 Mega BPS and 7150 CPS was reached at 176 clients. With 16 servers, the peak performance of about 39.4 Mega BPS and 15150 CPS was reached at 368 clients.

Scalability and hot spots As was mentioned above, if there are any hot spots (*i.e.*, extremely popular documents or images) in the data set, it may have negative effects to the performance of the DCWS system and limit its scalability. Figure 7(a) and Figure 7(b) show the peak performance of the DCWS prototype

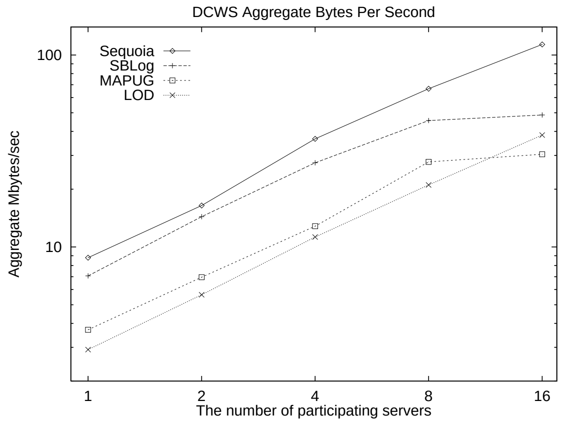


(a) Bytes per second (BPS)

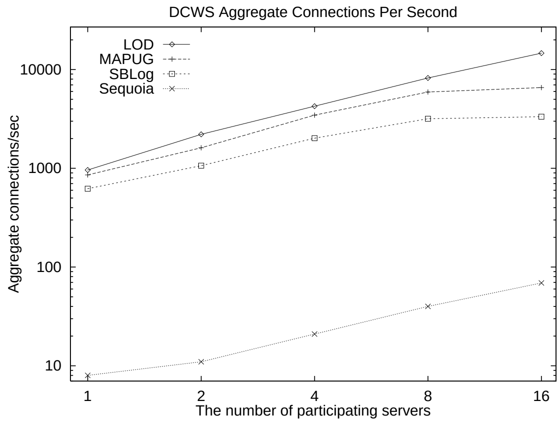


(b) Connections per second (CPS)

Figure 6: DCWS Performance from LOD data set with increasing numbers of concurrent clients



(a) Bytes per second (BPS)



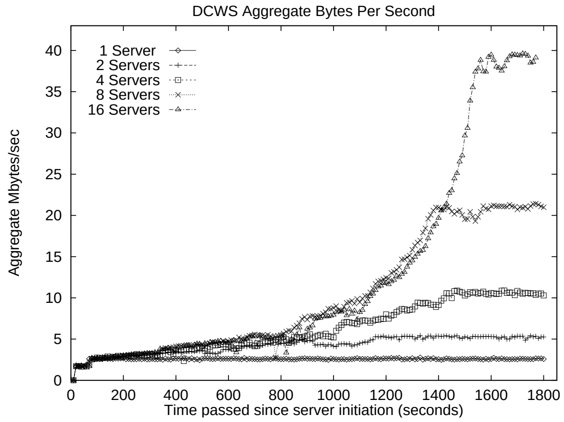
(b) Connections per second (CPS)

Figure 7: DCWS performance from different data sets with varying numbers of cooperating servers

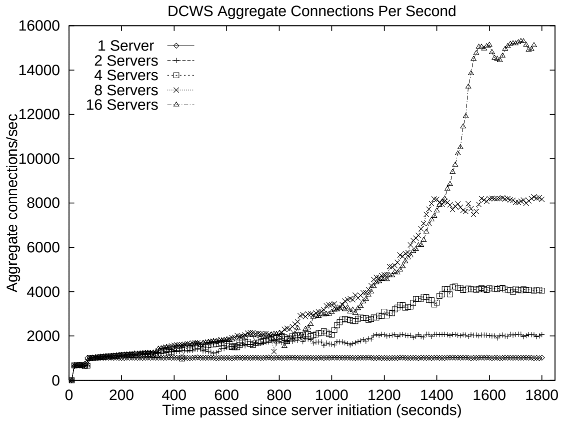
measured in BPS and CPS respectively, as the number of servers is increased, with four different data sets being used. With LOD and Sequoia data sets, both measures were found to be very close to linear up to 16 servers, which was the maximum number of available servers for the experiment.

However, we have observed substantially sub-linear scalability from the other two data sets SBLog and MAPUG. With SBLog data set, for example, 8 servers were able to handle approximately 3180 CPS and 45.5 Mega BPS. After the number of servers was increased from 8 to 16, the measures were approximately 3340 CPS and 48.7 Mega BPS, which were only about 5 and 7 percent improvement in performance. The reason is that there is intrinsic skew in access patterns in the data sets SBLog and MAPUG. Both of the data sets have very few images, but the images are linked from most of the documents in each data set. This produces excessive hits on whichever co-op servers get the migrated images, and eventually those servers become saturated on load and begin dropping requests.

From this set of experiments, we acknowledge that data distribution and data access characteristics have significant impact on the performance, and hot spots can limit the potential parallelism of the DCWS system. We conjecture that the only way to get around this problem is to adopt replication of hot spots, which is



(a) Bytes per second (BPS)



(b) Connections per second (CPS)

Figure 8: Time exponential performance growth from initiation of the DCWS system

not currently implemented in the DCWS prototype.

Exponential performance growth Figure 8 shows the performance of the DCWS system plotted against time. The server was configured with one home server started in initially a cold state with all files located on the home server and all the co-op servers empty. The DCWS system was run for a period of 30 minutes for each test, and the results were sampled at 10-second intervals. The results were interesting and somewhat unexpected in that the system performance (both CPS and BPS) showed an exponential increase over time. The performance improved slowly as the system was initiated from a cold state, but once it became warmed up and a significant number of documents had been migrated, the performance improved rapidly at a seemingly exponential rate.

We believe this exponential behavior is due to the parallel nature of the environment. System performance is a function of all machines in the DCWS server group. As each file is migrated from its home server to a destination co-op server, it produces the following multiple effects: (1) Utilization of the destination co-op server is increased due to having an additional document available to the server. (2) Each document remaining on the home server receives a higher per-document hit rate, since more bandwidth is available to be divided amongst the remaining documents. (3) A higher hit rate of the documents on the home server in turn causes enhanced utilization of other co-op servers, because the documents on the co-op servers are linked from the documents on the home server.

CPS vs. BPS The measurements of CPS and BPS are related by the size of the documents involved and connection overhead. Executing a web transaction via TCP connections requires exchange of several connection setup and tear-down packets in addition to the packets used to transfer the actual data. Thus, while a small file size increases CPS by reducing the number of bytes per transfer, it also decreases the overall BPS by wasting more bandwidth on additional connection overhead packets. This is corroborated by the results presented in Figure 7(a) and Figure 7(b). The highest BPS was observed from the Sequoia data set and followed by SBLog, MAPUG and LOD data sets, which is the decreasing order of the data sets in terms of average size of documents in the data sets. As we expected, in contrast, the CPS measures were observed in the reverse order.

Since real-world web transactions are fairly small [5]. we chose to use CPS as a balancing metric rather than BPS in Section 4. On the other hand, it is possible that in a system which uses significantly larger file sizes (such as the Sequoia storage benchmark data set), BPS may be a better load balancing metric. Since the connection setup and tear-down overhead is amortized by the large file size, BPS can represent load more accurately in a more fine-grained measurement than CPS.

Performance tuning Throughout the experiments, we have used the same parameter values summarized in Table 1. We believe that non-trivial changes in performance measures will be observed with different parameter values, and the choice of the parameter values will affect the performance of the DCWS system substantially. Although we have not yet investigated to tune the DCWS performance through extensive experiments, we expect there will be some clear tendencies in performance trade-offs. They are summarized in Table 2.

Parameters	Higher values	Lower values
$\mathcal{T}_{st}$	longer delay to balance load	overhead due to more often migration and load recalculation
$\mathcal{T}_{pi}$	less accurate statistics	more overhead due to forced pinger requests
$\mathcal{T}_{val}$	less piggybacked statistics and lower consistency	more retransmission of unchanged documents
$\mathcal{T}_{home}$	higher consistency but slower adjustment to changing conditions	more overhead for migration and redirection
$\mathcal{T}_{coop}$	less often migration and higher chance of over-migration	shorter delay to balance load

Table 2: Tuning server parameters

Overhead for parsing and reconstruction For all the HTML documents used in the experiments, the average size was approximately 6.5 Kbytes, the average time to parse hyperlinks without reconstruction was 3 milliseconds per document, and the average time to reconstruct documents was 20 milliseconds per document. In practice, the DCWS system requires less reconstructions. For the LOD data set, for example, the observed document reconstruction rate was 1.3 and 17.2 documents per second on average and at peak, respectively. Thus, parsing and reconstructing documents did not impose a significant performance penalty on the DCWS servers.

6 Conclusion and Future Work

We have designed and implemented a prototype system of the Distributed Cooperative Web Server (DCWS). The DCWS system is based on dynamic manipulation of the hyperlinks embedded in web documents in order to distribute access requests among multiple cooperating web servers. We have analyzed the performance of the DCWS prototype system with real-life data sets including the Sequoia storage benchmark data.

The experiments show that the DCWS system has a high potential to achieve linear scalability by effectively avoiding the bottleneck of centralized resources such as disk and network bandwidths. The overhead involved in hyperlink modifications was negligible compared with the improved performance gained by utilizing distributed servers. It has been shown that without the presence of the hot spots the peak performance was scaled almost linearly up to 16 cooperating servers. The load was evenly distributed among the servers and load balancing was achieved faster than linearly from a cold start. We conclude that the DCWS system proposed in this paper is a viable solution to building a scalable web server in both locally and geographically distributed environments. As an example, the DCWS system can be used to integrate a group of independent servers to build a federated web server in order to archive large-scale images and scientific data being produced and stored in geographically dispersed locations.

We recognize that there still remain several issues for further study of the DCWS system development. We have not taken into account the effects of user think time in the custom client benchmark and we have not used actual access logs for the experiments. These considerations may enhance the understanding of the DCWS system performance in more realistic situations. We plan to carry out further experiments in order to evaluate policies for document migration and consistency, and tune the performance parameters in geographically distributed and heterogenous environments. We also plan to extend the current implementation of the

DCWS system so that it can handle hot spots by replicating popular documents in a controlled manner, and investigate the effects of initial data distribution on the potential parallelism and scalability.

References

[1] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the WWW. In *the 4th International Conference on Parallel and Distributed Information Systems*, pages 92–103, Miami Beach, FL, December 1996.

[2] Eric Anderson, Dave Patterson, and Eric Brewer. The MagicRouter: An application of fast packet interposing. Submitted for publication.

[3] D. Andresen, T. Yang, O. Egecioglu, O. Ibarra, and T. Smith. Scalability issues for high performance digital libraries on the world wide web. In *Proceedings of ADL'96 Forum on Research and Technology Advances in Digital Libraries*, pages 91–100, Washington D.C., May 1996.

[4] Daniel Andresen, Tao Yang, and Oscar H. Ibarra. Toward a scalable distributed WWW server on workstation clusters. *Journal of Parallel and Distributed Computing*, 42:91–100, 1997.

[5] Martin F. Arlitt and Carey L. Williamson. Internet web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, October 1997.

[6] Azer Bestavros, Mark Crovella, Jun Liu, and David Martin. Distributed packet rewriting and its application to scalable server architectures. Technical Report TR-98-003, Boston University, Boston, MA, February 1998.

[7] Michele Colajanmi and Philip S. Yu. Adaptive TTL schemes for load balancing of distributed web servers. *ACM Sigmetrics Performance Evaluation Review*, 25(2):36–42, September 1997.

[8] Michele Colajanmi, Philip S. Yu, and Daniel M. Dias. Scheduling algorithms for distributed web servers. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, pages 169–176, Baltimore, MD, May 1997.

[9] Standard Performance Evaluation Corporation. SPECweb96 benchmark. http://www.specbench.org/-osg/web96/, April 1996.

[10] Mark E. Crovella and Robert L. Carter. Dynamic server selection in the internet. In *the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95)*, August 1995. Also available as TR-95-014, Boston University.

[11] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available web server. In *Proceedings of IEEE COMPCON Conference on Technologies for the Information Superhighway*, pages 85–92, February 1996.

[12] Peter Eriksson. Phttpd - a multithreaded web server. http://www.signum.se/phttpd.

[13] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. http://www.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-http-v11-spec-rev-03.txt, March 1997. Internet Draft.

[14] Roy T. Fielding and Gail E. Kaiser. The Apache HTTP server project. *IEEE Internet Computing*, 1(4):88–90, July/August 1997.

[15] Dean Gaudet. Apache performance notes. http://www.apache.org/docs/misc/perf-tuning.html.

[16] Eric Dean Katz, Michelle Butler, and Robert McGrath. A scalable HTTP server: The NCSA prototype. *Computer Networks and ISDN Systems*, 27:155–164, 1994.

[17] B. Krishnamurthy and C. E. Wills. Piggyback server invalidation for proxy cache coherency. *Computer Networks and ISDN Systems*, 30:185–193, April 1998.

[18] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed. Ncsa's world wide web server: Design and performance. *IEEE Computer*, 28(11):68–74, November 1995.

[19] Tam Nguyen and V. Srinivasan. Accessing relational databases from the world wide web. In *Proceedings of the 1996 ACM-SIGMOD Conference*, pages 529–540, Montreal, Canada, June 1996.

[20] Venkata N. Padmanabhan and Jeffrey C. Mogul. Improving HTTP latency. *Computer Networks and ISDN Systems*, 28:25–35, 1995.

[21] Michael Stonebraker, Jim Frew, Kenn Gardels, and Jeff Meredith. The SEQUOIA 2000 storage benchmark. In *Proceedings of the 1993 ACM-SIGMOD Conference*, pages 2–11, Washington, DC, May 1993.

[22] Cisco System. Scaling the internet web servers. http://www.cisco.com/warp/public/751/lodir/-scale_wp.htm, November 1997. White Paper.

[23] Shivakumar Venkataraman, Miron Livny, and Jeffrey F. Naughton. Memory management for scalable web data servers. In *the 13th Inter. Conference on Data Engineering*, pages 510–519, Birmingham, UK, April 1997.