Towards a Hierarchical Scheduling System for Distributed WWW Server Clusters

Daniel Andresen and Timothy McCune
Department of Computing and Information Sciences
234 Nichols Hall
Kansas State University
Manhattan, KS 66506
{dan, trm}@cis.ksu.edu

Abstract

In this paper we present a model for dynamically scheduling HTTP requests across clusters of servers, optimizing the use of client resources as well as the scattered server nodes. We also present a system, H-SWEB, implementing our techniques and showing experimental improvements of over 250%, which have been achieved through utilizing a global approach to scheduling requests. This is the first system to provide a hierarchical scheduling mechanism for distributed HTTP server clusters incorporating dynamic client-server task distribution and distributed data access. H-SWEB uses sophisticated scheduling techniques in monitoring and adapting to workload variation at the client and server clusters for supporting typical digital library tasks, such as fast WWW image browsing. We provide a discussion of our system architecture and implementation, and briefly summarize the experimental results that have been achieved.

1. Introduction

We have been studying scalability issues for distributed Hypertext Transfer Protocol (HTTP) servers on the World Wide Web (WWW), motivated particularly by applications within the digital library domain [3, 13, 16]. Digital library (DL) systems are increasingly moving from research to operational status, distributing their contents via the World-Wide Web. As their collections grow and diversify, their system architectures are also evolving from local clusters in geographical and logical proximity to widely diversified collections of databases and machines scattered across the globe. This diversity is at once an opportunity and an obstacle: an opportunity, since more resources and data are available, and an obstacle, due to the difficulties in effectively scheduling across widely separated heterogeneous domains.

Under the auspices of the Alexandria Digital Library Project (ADL) at the University of California, Santa Barbara (UCSB), we studied potential bottlenecks in digital library content delivery. The current collections of the Alexandria Digital Library project at UCSB [2] involve geographically-referenced materials, such as maps, satellite images, digitized aerial photographs, and associated metadata. The size of images can range in size from 10-100MB, with larger files expected in the near future as the scanning resolution increases. The metadata alone consists of millions of records and is multiple gigabytes in size. Typical requests include database searches, file fetches, and the progressive delivery of wavelet-encoded images, where the client possesses a low resolution image and requests the difference data be sent to construct a higher-resolution version. ADL has become a metadata repository for a number of physically and logically distinct collections of data, including data from various sites throughout the Western United States.

As digital libraries evolve, so must the depth and sophistication of our techniques. In our work on SWEB++ [5], we demonstrated that dynamically scheduling HTTP requests across a local server cluster and making use of client resources can significantly enhance the performance of WWW applications. We now investigate the potential performance improvements inherent in utilizing machines and resources beyond our local cluster and distributed across the Internet (Figure 1).

In this paper we present our model for partitioning and mapping client-server computation across multiple WWW server clusters. We also introduce a system, H-SWEB, implementing our model. We then present experimental results indicating that substantial performance benefits can be gained through using sophisticated multi-faceted scheduling techniques in a distributed environment.

The paper is organized as follows: Section 2 presents the computational model used by H-SWEB. Section 3 dis-

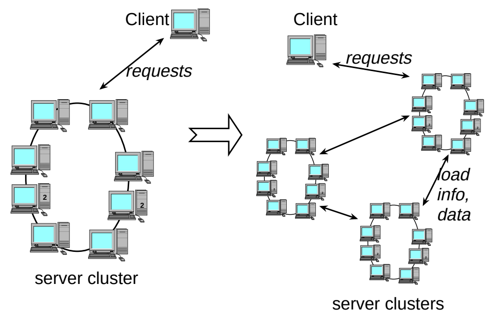


Figure 1. Extending SWEB++ to multiple clusters.

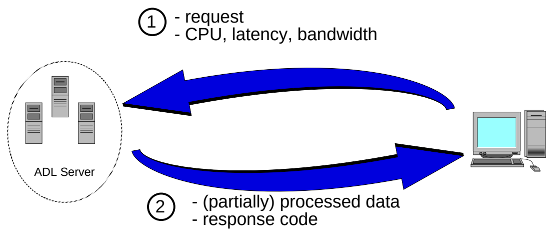


Figure 2. SWEB++ Client-server interactive model.

cusses the design and implementation of H-SWEB. Section 4 presents the experimental results, and Section 5 discusses related work and conclusions.

2. Background and model of computation

Our system model is a cluster of local server nodes, presented as a single logical server to the Internet, and connected by a fast local network. Each cluster connects to the Internet via links of varying speeds. The individual servers monitor the resource usage of their neighbors, and a synopsis of this data is distributed to remote sites. Every node is running a scheduler and HTTP server, and there is one process per cluster responsible for linking to the hierarchical scheduling system.

We assume that clients are either SWEB-unaware, in which case scheduling only occurs among the servers and no client computation is considered [4], or SWEB-aware, in which case there is a dynamic interaction between the client and server (Figure 2). If the client is SWEB-aware, then it is responsible for providing its load and connection information to the scheduler, and contributing its resources to complete the request via a custom Java applets. We model this interaction as a task chain that is partially executed at the server node (possibly as a CGI program [21]) and partially executed at the client (as a Java applet [17]). A task consists of a segment of the request fulfillment, with its associated computation and communication. Task communication costs differ depending on whether task results must be sent over the Internet or can be transferred locally to the next task in the task chain.

For example, ADL has adopted a wavelet-based progressive multi-resolution and subregion browsing strategy to reduce Internet traffic in accessing map images. This approach is based on the idea that users often browse large images via a thumbnail (at coarse resolution), and desire to rapidly view higher-resolution versions and subregions of those images already being viewed. To support these features, the ADL system is using a hierarchical data representation for multiresolution images [10]. Figure 3 depicts a task chain for processing a user request in accessing an image or subimage with a higher resolution, based on an implementation in [22].

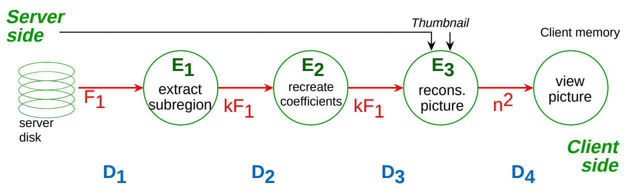


Figure 3. A task chain for image reconstruction.

Tasks in this chain are:

1. Fetch and extract the subregion indexing data when a user wants to review a subregion.
2. Use the indexing data to find coefficient data to be used in constructing a higher resolution subregion image.
3. Use the coefficient data and thumbnail image to perform a wavelet inverse transform in producing a higher resolution image.
4. View the image.

Thus there are four possible split points between a client and a server as shown in Figure 3.

D_1 Send the entire compressed image to the client and let it do everything.

D_2 Send over the relevant portions of the compressed image for the client to process.

D_3 Recreate the coefficients from their compressed representation and send them to the client for reconstruction.

D_4 Recreate the image requested by the client and send the pixels to the client.

For split point D_3, the thumbnail file is available at the client machine, thus the data access from the server disk for thumbnail data is eliminated.

We have shown analytically and empirically that a fixed partition point within the chain is unwise given the disparity in client resources expected [3, 6]. Our local algorithm, then, selects an appropriate node within the server cluster to process the request modeled by a task chain. If the expected response time is greater than a predetermined limit, the hierarchical scheduler is consulted. We also partition tasks in the chain into two sets, one for the client and another for the server, such that the overall request response time is minimized as illustrated in Figure 4. For example, if the client happens to be underpowered and one server node is lightly loaded, then keeping all of the tasks on that server node will be more reasonable in terms of request response time. In addition to client and server machine load and capabilities, network bandwidths between clients and a server also affect the partitioning point.

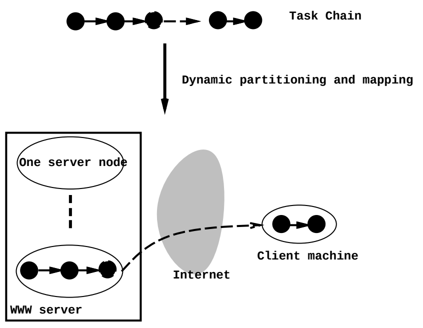


Figure 4. An illustration of dynamic task chain partitioning and mapping.

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

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

$t_{redirection}$ is the cost to redirect the request to another processor, if required. t_{data} is the server time to transfer the required data from a server disk drive. If data files are local, the time required to fetch is approximated as the file size divided by the available bandwidth of the local storage system. If data files are located on another local node, or a remote cluster, then each file must be retrieved through the interconnection network. Then t_{data} is approximated as the file size divided by the available remote access bandwidth. If there are many concurrent requests, local data transmission performance degrades accordingly. At run-time, the system needs to monitor the disk channel load of each local disk and also available remote data accessing bandwidths. t_{server} is the time for required server computation, which can vary depending on whether the request response has been partially or completely cached. t_{client} is the time for any client computation required, which is the number of client operations required divided by client speed. Here we assume the speed reported by the client machine includes client load factors. In the implementation, we need to collect three types of dynamic load information: CPU, disk, and network. CPU and disk activity can be derived from the Unix *rstat* utility, as well as some network information. Latency to the client can be approximated by the time required for the client to set up the TCP/IP connection over which the request (with the latency estimate) is passed. Client bandwidth is determined by the client, which measures the number of bytes per second received from the server for messages over a minimum size. The client passes both the latency and bandwidth estimates to the server as arguments to its HTTP requests.

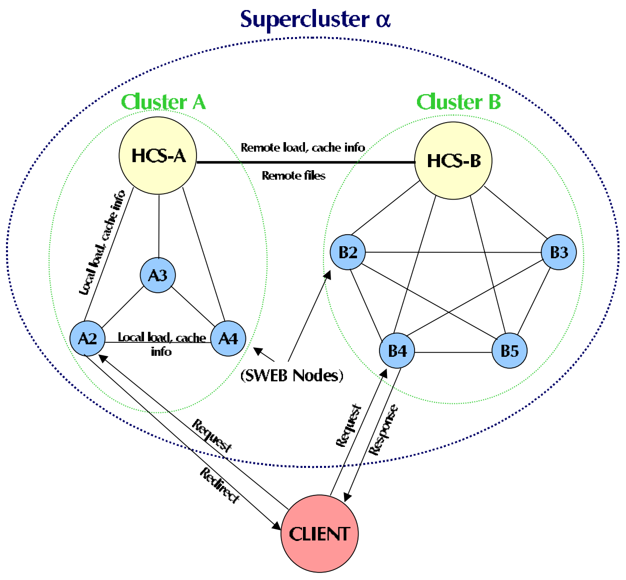


Figure 5. H-SWEB structural model.

3. System Design

Following the model given in the previous section, H-SWEB is implemented as a collection of collaborating WWW servers on a local network, with one process per cluster responsible for interaction with other clusters (See Figure 5). Interaction between clusters is achieved through the H-SWEB Cluster Server (HCS). The HCS runs as a separate process which may reside either on one of the server nodes, or a separate machine. It is responsible for three main functions. First, it gathers load and cache information about the entire cluster, as well as information about the bandwidth and latency between it and other clusters. It then distributes this information to other HCS nodes. Second, when the time for the local cluster to complete a request exceeds a predetermined threshold, the HCS analyzes the time to complete the request at each of the other remote clusters. It responds to the requesting server node with either a URL within a remote cluster that the client should be redirected to, or an instruction that the request should be fulfilled within the local cluster. Third, if a request requires data that resides within a remote cluster, the HCS can transfer this data to and from remote HCS hosts. If no communication has occurred with a remote cluster for a predetermined period, the cluster is presumed dead until communication is restored.

Within a cluster, the scheduler at each server node interacts with a daemon for handling HTTP requests. It has a server *broker* module which determines the best possible processor to handle a given request. The broker consults with two other modules, the $oracle$ and the $loadd$, to determine the appropriate split point. The $oracle$ predicts the CPU and disk demands for a particular request. The $loadd$ daemon is responsible for updating the local CPU, network and disk load information periodically (every 2-3 seconds) [4]. If the predicted response time is larger than a predetermined constant, the broker then consults with the HCS to determine the best possible site for processing the request. Our HTTP server code is based on a slightly-modified preliminary version of Swala, the scalable caching web server project at UCSB implementing the SWEB scheduling algorithms and task chain caching [19].

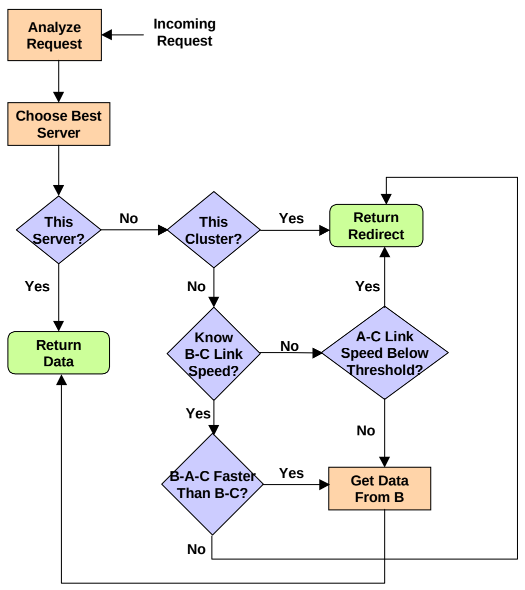


Figure 6. The HCS decision path.

In some cases, a request, such as a wavelet transform, may require one or more input files that are not located within the local cluster. These input files may reside in any number of different remote clusters. When the HCS analyzes this request, it first considers the time required to transfer all of the input files to a particular cluster. Then it considers the time it will take that cluster to fulfill the request based on the cluster's load information. These times are compared, and the fastest cluster is then chosen. A illustration of H-SWEB's decision process is given in Figure 6.

Our bandwidth and latency information between clusters is provided by the Network Weather Service (NWS). The NWS is a generalizable and extensible facility designed to provide dynamic resource performance forecasts in metacomputing environments. While the forecasting methods are general, they focus on the ability to predict the TCP/IP end-to-end throughput and latency that is attainable by an application using systems located at different sites [23].

H-SWEB is designed to use true hierarchical scheduling. In such a system, an HCS can provide service to a supercluster (a group of clusters or a group of other superclusters) in the same fashion that it provides service to a cluster (a group of individual nodes.) The current implementation uses a hierarchy that is only one layer deep. We plan to use an H-SWEB hierarchy with multiple levels, but this functionality has not yet been implemented. In its current form, H-SWEB would be best suited for deployment within a close-knit community such as the Digital Library projects, due to the need to install trusted software and a modified HTTP server. Clients, however, require only the use of a Java-compatible browser.

Minimizing the overhead involved with HCS scheduling has had a significant impact on design issues. In some cases, abstractions were useful in reducing this overhead. For instance, instead of broadcasting load information on every node within its cluster, the HCS broadcasts only the average load, the lowest load, and the number of nodes. In other cases, we dynamically adapt the imposed load of the HCS based on available system resources. As an example, the rate at which an HCS retrieves load and cache information from other clusters changes dynamically based on the amount of information being transferred as well as the bandwidth and latency between the two clusters.

4. Experimental Results

In our experiments we demonstrate the effectiveness and logic of our scheduling scheme in a distributed, heterogeneous environment. We also briefly discuss the overhead incurred by our system. Two types of requests are examined: fetching a file and fetching a wavelet-encoded image (as described in Section 2). We fetch two file sizes, *small* (10KB), and *large*(1.5MB), representing typical HTML and significant binary data, respectively. The wavelet operation is to extract a 512×512 subregion at full resolution from a $2K \times 2K$ map image, representing the user zooming in on

	Client location	
Data loc.	KSU	UCSB
KSU	small and large files remain local	small files are mixed, large are redirected
UCSB	small files are mixed, large are redirected	small and large files remain local

Table 1. Scheduling results for fetching files

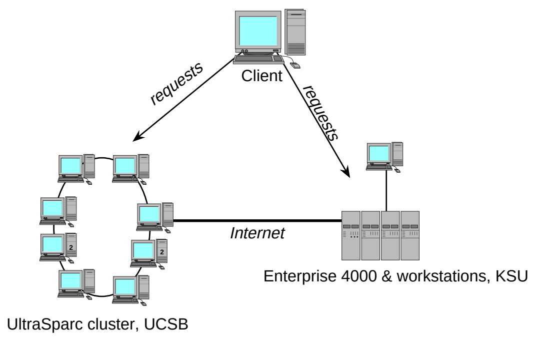


Figure 7. The KSU-UCSB network model.

a point of interest at a higher resolution after examining at an image thumbnail. We also use the full $2K \times 2K$ extraction as a more significant task, taking approximately 20s. of processor time on a Sun Ultra1 workstation (in contrast with the 2-3s. for the 512×512 subimage). For both types of requests, if the data required to fulfill the request is located at a remote cluster, the scheduler has the option of transmitting the data to a local node for processing or redirecting the request for processing at the cluster where the data is located.

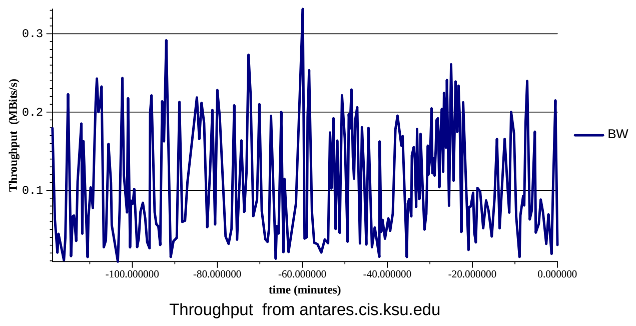


Figure 8. KSU-UCSB network bandwidth variations.

For our experiments, we used clusters located at UCSB and Kansas State University (KSU). At UCSB the cluster consisted of Sun Ultra1 and Ultra2 workstations connected locally via 100Base-T Ethernet, and connected to the Internet by a vBNS link. At KSU, the cluster consisted of a Sun Ultra1 and 10-processor Sun Enterprise 4000 linked by 10Base-T Ethernet, and connected to the Internet by dual T1 lines (which average over 95% utilization). Clients were located at both KSU and UCSB. As is usually the case when working with a distributed computing solution over the Internet, individual test results varied widely depending on the usage of the clusters and the available network capacities. For example, Figure 8 shows the typical bandwidth variations in the early afternoon between KSU and UCSB (data from NWS). All tests were for a period of 30 s., and the results are averaged over multiple experiments. The HCS process is implemented in Java, running under JDK 1.1.4 without a JIT compiler. As such, HCS is portable across various platforms without requiring recoding or recompilation. Swala is written in C++ [19].

File Fetch The results from our experiments where a file was fetched repeatedly from the various sites were as expected, as shown in Table 1. The configuration was an Ultra1 at KSU, and multiple Ultras at UCSB. Requests for small files, whose total response time would not be materially affected by any normal scheduling, were evenly distributed between redirecting to remote hosts or fetching the data for retransmission. The individual decisions were swayed by the balance between the available bandwidth (affecting t_{data}) and cost to establish a new connection ($t_{redirection}$). Fetching large files, however, causes the t_{data} and t_{net} terms to dominate. The scheduling algorithm then follows file locality. When H-SWEB scheduling is enabled, allowing the servers to follow data locality, response times become 64% to 197% faster (averaging 112%) (Figure 10(a)). In Figure 10(b), improvements for large files ranged from 44-175%, averaging 122%.

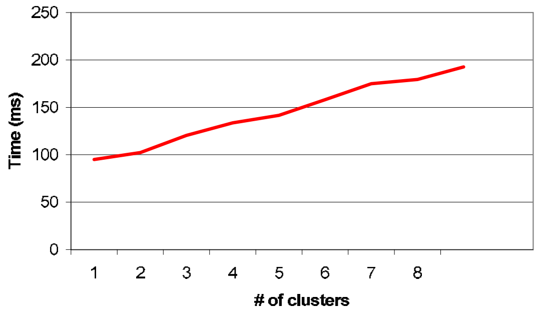


Figure 9. HCS scheduling time

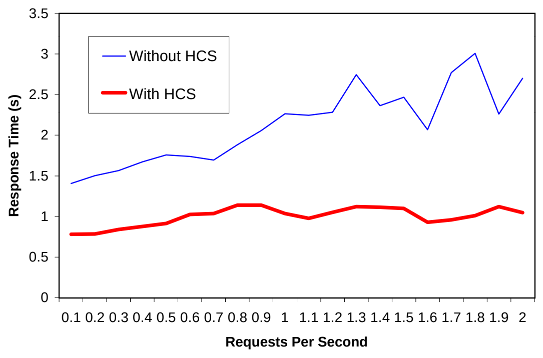
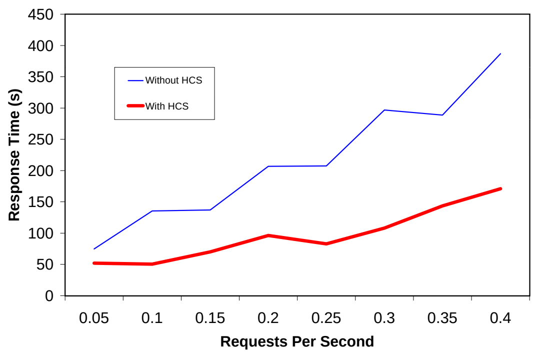
(a)  (b) 

Figure 10. File-fetch request times for (a) 10KB and (b) 1.5MB files.

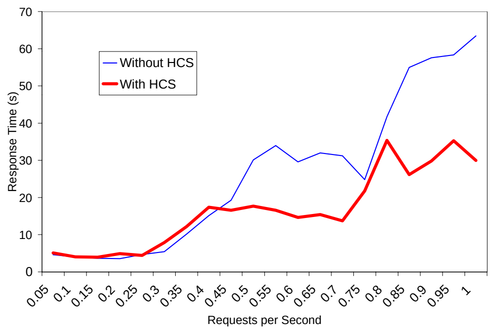
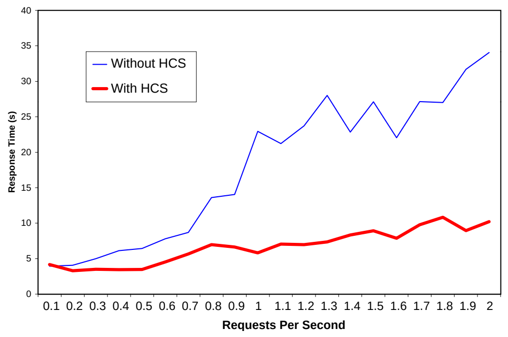
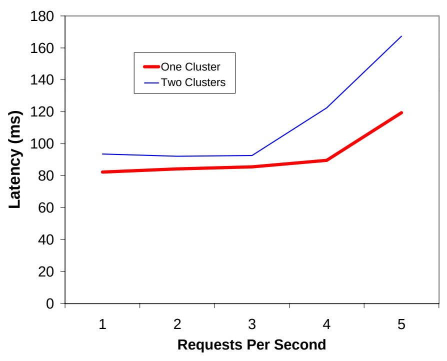
(a)  (b) 

Figure 11. H-SWEB wavelet performance. (a) uniform requests (b) heterogenous requests

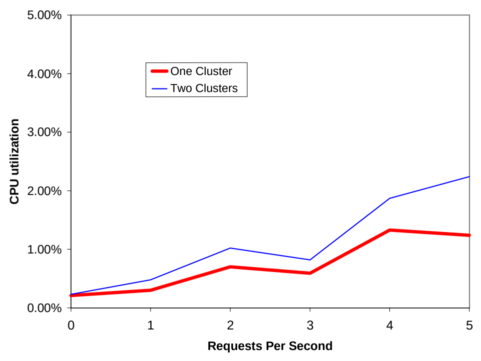
Wavelets A request for an image stored in the wavelet file format is considerably more CPU intensive than a typical file fetch. As such, the processing resources available at any particular moment (affecting t_{server}) tend to significantly affect the scheduler. Because of this high demand for system resources, if one particular cluster is significantly more powerful than the others, the vast majority of the wavelet requests will end up at this cluster. The real advantage of HCS scheduling becomes evident when all of the clusters involved in the system possess a similar amount of resources. When this is the case, initial requests tend to remain within the local cluster to save the overhead of redirection. Once the local cluster reaches a certain saturation, succeeding requests are redirected to remote clusters. As requests continue to come into the system, the load on these remote clusters reaches levels similar to those of the local cluster. At this point, subsequent wavelet requests will typically be evenly distributed across all clusters. For the experiments shown in Figure 11(a), we used the same configuration as in the previous section, launching homogenous 512x512 wavelet transform requests in bursts of 30 sec., and achieved improvements in the response times of averaging 18%, peaking at 53%. The experiments in Figure 11(b) consisted of a uniform mixture of 512x512 and 2048x2048 wavelet transform requests distributed evenly to the Ultra cluster at UCSB and a two Ultra, one Enterprise-4000 cluster at KSU, with the client located at KSU. The requests arrived over a 30 second period. The improvements ranged from -6% under low loads to 250+% at higher loads, averaging over 147%.

Overhead The system and per-request overhead imposed by HCS is small. The overhead of the additional layer of scheduling is only incurred where the request is estimated to take a long time to completion. We expect only a small fraction of requests will require this additional computation; however, these also stand to gain the most from a possible transfer to another site. The per-request scheduling performed by Swala/SWEB++ takes < 10 ms., and, HCS adds approximately 10 ms. + round-trip communication time (80ms. between two Java VM's over an Ethernet 10Base-T network). Figure 9 shows the time required for scheduling as the number of clusters increases. We also see from Figure 12(a) the scheduling times increases slowly as multiple simultaneous requests arrive. We feel these times will drop substantially with the use of a Java JIT compiler for HCS.

The overhead of load monitoring via the NWS and the SWEB/Swala systems is also small, taking less than 1% of CPU and network resources, and HCS itself averages less than 1% under normal loads (Figure 12(b)) [3, 23]. The ability of H-SWEB sites to dynamically increase or decrease



(a)



(b)

Figure 12. (a) HCS request overhead. (b) HCS system overhead

the intervals between updating remote information further contributes to keeping the imposition small on local and remote clusters.

5. Related work and Conclusions

Other systems have explored runtime load balancing within a distributed system, although none with the context of the WWW and digital libraries. The Flock of Condors work provides a hierarchical metacomputing environment with services beyond those of H-SWEB, including checkpointing and system call translation, at the cost of requiring application relinking to the Condor libraries [12]. The Legion project provides a distributed load-balancing environment, at the cost of rewriting applications to fit the project libraries and languages [18]. The Globus project also aims at providing a global metacomputing object-oriented environment, but is oriented towards a different set of problems than H-SWEB [14]. Projects in [8, 11, 9] are working on global computing software infrastructures, but are generally still in the design stage. Scheduling issues in heterogeneous computing for large-scale scientific applications using network bandwidth and load information are addressed in [7]. The above work deals with an integration of different machines as one server and does not have the division of client and server. Recent work on mobile computing (e.g. [1]) also uses bandwidth and data locality information to dynamically migrate computation. Our project focuses on the optimization between a server and clients, and uses tightly coupled server nodes for a WWW server, generalizing to loosely coupled server clusters. Addressing client configuration variations is discussed in [15] for filtering multi-media data but it does not consider the use of client resources for integrated computing. Several distributed HTTP servers and techniques are listed on the "Distributed Internet Servers" page, but are designed for implementation on local area networks [20].

In this paper we have presented a model for dynamically scheduling HTTP requests across clusters of servers, optimizing the use of client resources as well as the scattered server nodes. We also presented a system, H-SWEB, implementing our techniques and showing experimentally gains of over 250% can be achieved through utilizing a global approach to scheduling requests. H-SWEB is the first system to provide a hierarchical scheduling mechanism for distributed HTTP server clusters incorporating dynamic client-server task distribution, caching of partial results, and distributed data access. H-SWEB uses sophisticated scheduling techniques in monitoring and adapting to workload variation at the client and server clusters for supporting typical digital library tasks, such as fast WWW image browsing. In the future we plan to extend our system to more generalized scheduling problems, and explore several object-oriented scenarios.

5.1 Acknowledgments

This work was supported in part by startup funds from Kansas State University, NSF IRI94-11330, CCR-9702640, CDA-9529418, and grants from Navy NRaD and UC MICRO/SUN. We would like to thank Vegard Holmedahl, Ben Smith, and Tao Yang for the use of Swala and their advice and suggestions. We also thank Rich Wolski and Jim Hayes of UCSD for their gracious assistance with the Network Weather Service.

References

[1] A. Acharya, M. Ranganathan, and J. Saltz. Sumatra: A language for resource-aware mobile programs. *Lecture Notes in Computer Science*, 1222:111–??, 1997.

[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, and Q. Zheng. The WWW prototype of the Alexandria digital library. In *Proceedings of ISDL'95: International Symposium on Digital Libraries*, Japan, Aug. 1995.

[3] D. Andresen and T. Yang. Multiprocessor scheduling with client resources to improve the response time of WWW applications. In *Proceedings of the 11th ACM/SIGARCH Con-*

ference on Supercomputing (ICS'97), Vienna, Austria, July 1997.

[4] D. Andresen, T. Yang, V. Holmedahl, and O. Ibarra. SWEB: Towards a scalable world wide web server on multicomputers. In *Proceedings of the 10th IEEE International Symp. on Parallel Processing (IPPS'96)*, pages 850–856, Honolulu, Hawaii, April 1996.

[5] D. Andresen, T. Yang, and O. Ibarra. Towards a scalable distributed WWW server on networked workstations. *Journal of Parallel and Distributed Computing (JPDC)*, 42:91–100, September 1997.

[6] D. Andresen, T. Yang, D. Watson, and A. Poulakidas. Dynamic processor scheduling with client resources for fast multi-resolution WWW image browsing. In *Proceedings of the 11th IEEE International Symp. on Parallel Processing (IPPS'97)*, pages 167–173, Geneva, Switzerland, April 1997.

[7] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing'96*, Pittsburgh, PA, November 1996.

[8] H. Casanova and J. Dongarra. NetSolve: A network-enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, Fall 1997.

[9] B. Christiansen, P. Cappello, M. F. Ionescu, M. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-Based Parallel Computing Using Java. *Concurrency: Practice and Experience*, 1997. to appear.

[10] C. K. Chui. *An Introduction to Wavelets*. Academic Press, San Diego, CA, 1992.

[11] K. Dincer and G. Fox. Building a world-wide virtual machine based on Web and HPCC technologies. In *Proceedings of Supercomputing'96*, Pittsburgh, PA, November 1996.

[12] X. Evers, J. F. C. M. de Jongh, R. Boontje, D. H. J. Epema, and R. van Dantzig. Condor flocking: load sharing between pools of workstations. Technical Report DUT-TWI-93-104, Delft University of Technology, Department of Technical Mathematics and Informatics, Delft, The Netherlands, 1993.

[13] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, et al. RFC 2068: Hypertext transfer protocol — HTTP/1.1, Jan. 1997.

[14] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[15] A. Fox and E. Brewer. Reducing WWW latency and bandwidth requirements by real-time distillation. *Computer Networks and ISDN Systems*, 28(711):1445, May 1996.

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

[17] J. Gosling and H. McGilton. *The Java Language Environment: a white paper*. Sun Microsystems, Oct. 1995.

[18] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds, Jr. Legion: The next logical step toward a nationwide virtual computer. Technical Report CS-94-21, Department of Computer Science, University of Virginia, June 08 1994.

[19] V. Holmedahl, B. Smith, and T. Yang. Cooperative caching of dynamic content on a distributed web server. In *Proc. of the Seventh IEEE International Symposium on High Performance Distributed Computing(HPDC7)(to appear)*, Chicago, Illinois, July 1998.

[20] I. Kallen. Distributed internet servers, Feb. 1998. http://www.arachna.com/webservers/distributed_servers.html.

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

[22] A. Poulakidas, A. Srinivasan, O. Egecioglu, O. Ibarra, and T. Yang. Experimental studies on a compact storage scheme for wavelet-based multiresolution subregion retrieval. In *Proceedings of NASA 1996 Combined Industry, Space and Earth Science Data Compression Workshop*, Utah, April 1996.

[23] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, April 1998.