A Content Placement and Management System for Distributed Web-Server Systems

Chu-Sing Yang and Mon-Yen Luo
Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, R.O.C

Abstract

Clusters of commodity computers are becoming an increasingly popular approach for building cost-effective high performance Internet servers. However, how to place and manage content in such a distributed and complex system becomes a challenging problem. In particular, such distributed servers tend to be more heterogeneous, and this heterogeneity will further increase the management burden. This paper describes the motivation, design, implementation and performance of a content placement and management system for the heterogeneous distributed Web server.

1. Introduction

The explosive growth of the Internet, in particular the World Wide Web, has resulted in heavy demands being placed on Internet servers and has raised great concerns in terms of performance, scalability and availability. A monolithic server hosting a service is usually not sufficient to handle these challenges. Distributed server architecture, consisting of multiple heterogeneous servers that appear as a single server, has proven [1,2] a successful and cost effective alternative for building a scalable, reliable, and high-performance Internet server system. Consequently, more and more Internet service providers run their service on a cluster of servers (e.g., Yahoo [3], Alta Vista [4], Netscape [5]), and this trend is likely to accelerate.

A very important but often ignored issue that arises in the clustered Web server is how to place and manage content (i.e., documents, Web pages, resources, etc., generally called content in the Internet parlance) in such a distributed system. In particular, such distributed servers tend to be more heterogeneous because they generally grow incrementally as need. This is an important advantage of distributed servers that they can scale gracefully with offered load, preserving previous investment in hardware and software. Unfortunately, this advantage will come at the cost of greatly increased management burden.

1.1 Existing Solutions

One possible solution to the content management problem is to place all content on a centralized network file system (e.g., NFS). Through the communication network, server nodes at different locations can access the content from the shared file system to serve user requests. The advantage of this approach is that we can easily manage and maintain the content with a centralized policy. However, such a design will suffer from the single-point-of-failure problem, which will make the entire system vulnerable. Furthermore, accessing data over the network file system will increase user perceived latency due to the overhead of remote-file-I/O and LAN congestion.

Some Web sites use an alternate approach that replicates all content on the local file system of each server node. The advantage of content replication is that it can avoid the significant overhead associated with the networked file system scheme. Furthermore, redundant data on each server node provides tolerance to system failures. However, besides the storage overhead, full replication of content will also pose a great administrative burden on content management. When a page is updated, such a change must be propagated to all participating servers of the cluster. If the content is highly volatile, the complexity of maintaining consistency will make this method undesirable.

Neither of the two schemes is a satisfactory solution for a heterogeneous clustered Web-server. One common problem of the two schemes is that they ignore the heterogeneity of content and server configuration. The content provided by modern Web servers can consist of static Web pages, dynamic content (e.g., generated by a CGI script), or multimedia data such as streaming audio or video. Different services have different requirements in terms of system resources. For example, requests for executing CGI scripts normally require much more computing resources than static file retrieval requests [6]. As a result, some nodes with slow processors are not suitable for providing CPU-intensive dynamic content. In

addition, placing all content together may also degrade the user perceived performance. We noticed that CPU-intensive dynamic web requests or long connection requests (i.e., requests for large files) will delay the delivery of static content, resulting in longer response time for these shorter requests. Others have made a similar observation [7]. Finally, not all content is equally important to the client and service provider. We also noticed that requests for popular pages have a tendency to overwhelm the requests for other critical pages (such as product lists or shopping-related pages). These critical documents should be separated or be allocated more system resources in order to achieve better quality of service.

1.2 Proposed System

Motivated by the above observations, we propose a new content placement scheme, which allows the administrator to partition (or partially replicate) the content by hosting it on different servers. The content partitioning may be governed by type (e.g., static HTML pages, CGI scripts, multimedia files, etc.) or by some other policy (e.g., priority). The two traditional schemes also can be incorporated with this scheme. The administrator can replicate some critical content to multiple nodes for achieving high availability or providing better performance. Otherwise, some specific content can be placed on a shared file server or application server (e.g., database server).

The proposed scheme has several advantages over the traditional schemes and is better suited to heterogeneous distributed server. Firstly, compared to full replication, this scheme yields better resource utilization and scalability while it also can avoid the overhead associated with the shared file system approach. Many research studies have revealed that the access pattern of Web content is skewed [8], and the distribution of Web file size is heavy-tailed [9]. According to the statistical data reported by [10], large files (up to 64 KB) make up only 0.3% of the content but consume 53.9% of the required storage space. In addition, these large files receive only 0.1% of all client requests. Thus, full replication of these files is not cost-effective. We can just place these files on some nodes in the server cluster. Secondly, we can place different content on different servers optimized for addressing requirements imposed by various data types. For example, we can place multimedia content on servers optimized for stringent real-time requirements. Thirdly, content segregation can prevent interference between different requests (e.g., servicing CGI requests does not interfere with multimedia requests with real-time requirements), thereby optimizing the user perceived performance. Finally, such a scheme enables administrator to exert explicit control over resource allocation policies (e.g., place critical content on more powerful machines), which can provide differentiated QoS according to the variety of content.

However, for fulfilling such a scheme, we need an efficient management system to address the following problems:

- **Directing Mechanism**
  Given a distributed server, some directing mechanism is needed to route the incoming request to the server best suited to respond. However, under such separated content deployment, the directing mechanism should be "content-aware" so that it can direct a request to the server where the requested resources reside.
- **Content Management**
  When the server is composed of a group of loosely coupled machines, and then the document tree is partitioned into different nodes, the managing problem is exacerbated. Thus, developing a management system to automate administrative operations and provide a logical view of a monolithic system is extremely important.
- **Load Balancing**
  The dispersing content approach could lead to load imbalance derived from the access skew among the documents. In other words, the servers that store the hot documents will get overloaded, resulting in hot spots.

In section 2, we will describe how we designed and implemented a new directing mechanism called content-aware distributor to solve the first problem. To address the latter two issues, we implemented a content management system to mask the complexity and heterogeneity of the distributed environment. In this management system, we also implemented an automatic replication facility to guarantee load balancing among the server nodes. We describe the content management system in section 3. In section 4, we describe our experience of using the proposed system in our Web site. Section 5 presents the result of performance evaluation of the prototype system. The results will demonstrate the performance gain derived from the proposed scheme. We discuss other related work in section 6, and then present the conclusion in section 7.

2. Content-Aware Distributor

In this section, we first analyze the existing request routing approaches. We found that none of these approaches could efficiently support the proposed content-placement scheme. We then proposed a new request distributing mechanism termed content-aware distributor to address this problem.

2.1 Analysis

Over the past few years, a considerable number of researchers and vendors have proposed methods for routing the user requests in a distributed Web server.

These solutions can be broadly divided into the following categories: client-side approach (e.g., [11]), DNS-based approach [12,13], TCP connection routing (or termed layer-4 routing) [14,15,16,17,18,19,20], and HTTP redirection [21]. Unfortunately, none of these approaches could effectively support our content placement scheme.

A routing mechanism generally has to collect some information about server's state to perform routing decisions. When request routing is performed at the client side, the status information that is collected remotely may be stale, resulting in bad routing decisions. In addition, performing request-routing decision at the client side also poses security risks. Both DNS-based and layer-4 routing approaches are content-blind, because they determine the target server before the client sends out the HTTP request. HTTP redirection might be used for content-aware routing. However, we do not prefer HTTP redirection because this mechanism is quite heavy-weight. Not only does it necessitate the use of one additional connection, which introduces an extra round-trip latency, but also the routing decision is performed at the application level and uses the expensive TCP protocol as the transport layer. Motivated by these observations, we propose a new mechanism, called a content-aware distributor, to effectively support content-aware routing.

2.2 Proposed Mechanism

The major challenge in designing the content-aware mechanism is the connection-oriented semantics of TCP protocol that the Web service is based on. Whenever a client tries to issue a request to the Web server, it must first establish a TCP connection to the server. As a result, the dispatcher node has to establish a TCP connection with the client so that it can examine the subsequent HTTP request to perform content-based routing. When the routing decision has been made, relaying the HTTP request (or migrating the established connection) from the dispatcher to the selected server becomes a challenging problem. In particular, such a mechanism should be transparent to users and efficient enough to not render the dispatcher a bottleneck. Figure 1 shows our design to tackle this problem.

The distributor pre-forks a number of persistent connections (supported by HTTP 1.1 [22]) to the back-end nodes. When a client tries to initiate an HTTP request, the client-side browser first opens a TCP connection, resulting in an exchange of SYN packets as part of TCP's three-way handshake procedure [23]. After receiving the SYN packet, the distributor first creates an entry (indexed by the source IP address and port number) in an internal table (termed mapping table) for this connection then records the TCP state information (e.g., sequence number, ACK number, etc.) in the entry. The distributor then handshakes with the client to complete the TCP connection setup.

After the TCP connection setup is completed, the client sends packets conveying the HTTP request which contains the URL (specify the specific content it is asking for) and other HTTP client header information. Based on the content requested, the distributor consults an internal data structure called URL table to select the server that is best suited to this request. The URL table holds content-related information (e.g., location of the document, document sizes, priority, hits, etc.), which helps the distributor to make the routing decisions. We implemented several administration functions and new system calls for administrator to configure the URL table.

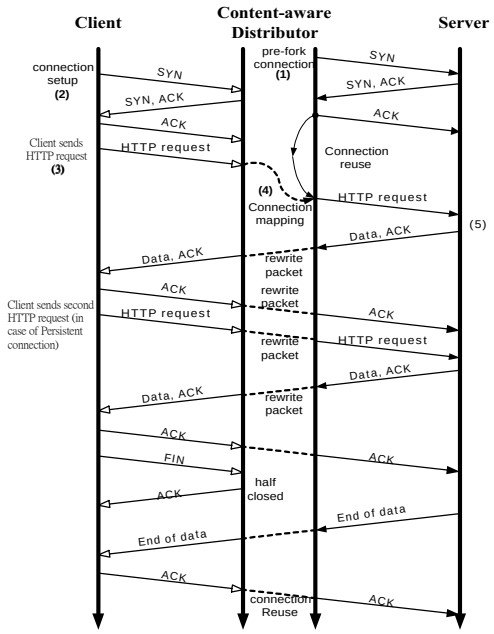

Figure 1. Operation of content-aware distributor

Once the distributor selects a target server, it also chooses an idle pre-forked connection from the available connection list. Then the distributor stores related information about the selected connection in the mapping table, which will bind the user connection to the pre-forked connection. After the connection binding is determined, the distributor handles the consequent packets by changing each packet's IP and TCP headers for seamlessly relaying the packet between the user connection and the pre-forked connection, so that the client and the server can transparently receive and recognize these packets.

If the distributor receives a FIN packet from the client, which indicates that the client intends to close this connection, the state of corresponding entry in the mapping table will be changed to FIN_RECEIVED. The distributor responds with an ACK packet to the FIN packet and then sets the state to HALF_CLOSED. When the last packet of this request is relayed and ACK of this packet from client arrives, the state is changed to

CLOSED. The distributor then deletes the entry associated with this connection and releases the pre-forked connection back to available connection list. If the client use HTTP 1.0 protocol, the distributor will set the FIN flag instead of server when it relay the last packet, and then also release the pre-forked connection.

2.3 Implementation

For both efficiency and elegance, we implemented [24] the proposed mechanism as a kernel loadable module, which is a software component that can be dynamically loaded into the kernel for the purpose of kernel extension. The distributor module inserts itself between the network interface (NIC) driver and the TCP/IP stack. To route the incoming Web request, all packets relating to Web service should go through the content-aware distributor. To achieve this, we can load the distributor module into the router that functions as the front end to the clustered Web servers. We have extended the Linux kernel with this module. Due to space limitations, we omit the description of implementation details. The reader is referred to [24] for a further information.

In addition, we noticed that the distributor represents a single-point-of-failure in our system, i.e., failure of the distributor will bring down the entire Web server. We implemented the primary/backup(s) mechanism (proposed in [2]) to achieve fault tolerance of the distributor. While the *primary* distributor is providing service normally, the *backup* distributor remains in a monitor state, continuing to monitor the primary and replicate the primary's state. If the primary distributor fails, the backup takes over the job of the primary and creates its own backup.

3. Content Management System

When the server is composed of a group of loosely coupled machines, and the document tree is partitioned onto different nodes, the management problem is exacerbated. To address this, we leverage ideas in prior work [25] to design and implement a content management system. The content management system provides the administrator with a single system image, which makes the administrator oblivious of the presence of content segregation on multiple nodes. In addition, the system can also automate many management operations, which could ease the burden of system management. In this section, we first briefly describe the prior work, and then describe how we extend it to fulfill a content management system.

3.1 Prior Work

In previous work [25], we had exploited the advantages of Java to design and implement an extensible framework to address the administration problem of distributed Internet servers. This administration framework is composed of the following four key components: *controller*, *broker*, *agent*, and *remote console*.

The *broker* is a standalone Java application, which executes as a daemon process on each backend server in order to perform the administrative functions and monitor the status (e.g., load situation, failure) of the managed node. Each administrative function is implemented in the form of a Java class, which is termed an *agent*. The brokers distributed on each node may download the appropriate classes to perform the corresponding management tasks. One special daemon, called the *controller*, is responsible for receiving requests from the administrator and then invoking brokers to perform the delegated tasks by dispatching the corresponding agents. The controller resides on the distributor. The *remote console* is a Graphical User Interface (GUI) in the form of a Java applet; thus it can run under any Java-enabled browser. The administrator can download the remote console and interact with it to perform management operations. This GUI also supports tracking and visualization of the system's configuration and state. When an administrator issues a management command, the remote console will inform the controller, which then dispatches the respective agent to each server node for performing the job.

3.2 Extension

We first extended the remote console to produce a single, coherent view of the Web document tree, comprised of portions that actually reside on several different server nodes. The remote console provides a file manager interface containing methods for inserting, deleting, and renaming files or directories. With the GUI, the administrator can easily assign different content to different servers for meeting the performance requirements of the heterogeneous data types and applications. The administrator also can assign some specific content to multiple server nodes for fault tolerance or high availability. Whenever the administrator changes the document tree, the remote console will inform the controller of these changes. The controller will change the URL table to adapt to these changes, and then send the agent that performs the content management function to propagate these changes to the whole system. We implemented several new agents to perform the content management functions. For example, one agent is responsible for deleting a file from the local file system of the node that it executes. If the administrator tries to offload some pages from a server, the controller will send this agent to that node for performing this job.

There are two main advantages of this content management system. First, implementing the daemon in Java can relieve the concerns related to heterogeneity of the target platforms. As a result, these daemons can be capable of executing on a variety of hardware architectures and operating system. The second advantage

is derived from the notion of downloaded executable content (data that contain programs that are executed upon receipt), which is a powerful feature of Java. Using the mobile code technology, we can deploy just a simple local daemon (broker) at each node, but supports a variety of management functions via downloading agent. As a result, this management system can be tailored or extended to the different requirements of different system type and installation, without requiring significant redesign and coding.

3.3 Load Balancing

Since the access patterns of WWW exhibit high skew [8] with regard to the access frequencies of content, the segregation content placement may lead to load imbalance. As a result, we implement an auto-replication facility to further ensure an even load distribution. We use the following load metrics to perform load balancing. We define l_i is the load caused by a request to a given content *i*. The l_i is caculated using the following formula:

$$l_i = (\text{load}_{CPU} + \text{load}_{Disk}) \times \text{processing time}$$

The processing time is the time that the request took from start to finish, which is caculated by distributor. For a request to the static content, load_{CPU} is set to one and load_{Disk} to nine, since disk activity is the dominant factor of a server's load. For the request to a dynamic content, load_{CPU} is set to ten and load_{Disk} to five. These constants are chosen heuristically based on the performance data provide by [2,6]. A different weighted-parameter can be chosen, this is an area of further research and beyond the scope of this paper. In this paper, we only show that a somewhat heuristic constant that makes intuitive sense works well. We define L_j is the load caused by all contents placed on server j. The L_j is caculated using the following formula:

$$L_j = (\sum (l_i \times \text{access frequency})) / \textit{Weight}$$

The *Weight is* a static weighting value which is based on the capacity of each server. Periodically, the load metrics L is calculated by distributor and reflects the accumulated load in the time interval. The distributor also calculates the average load of all server nodes. If the load of one node exceeds the average load by a threshold, the node is determined to be overloaded. Under such condition, the distributor will inform the controller, and then the controller will decrease the content copies of that server. Conversely, if the load of one node is below to the average load by a threshold, the node is determined to be underutilized. The controller then sends several agents to automatically replicate some popular content to this underutilized server. Whenever the replication or offloading occurs, the controller also updates the URL table to adapt to these changes.

4. Experience

The system is being successfully used on our distributed Web server, which is providing many large-scale Internet services. As would be expected in any distributed server, our system is highly heterogeneous and is made up of several different hardware and software environments. As a result, some nodes may not be powerful enough to support the entire service, but can probably support some components of the service. The proposed system enables us to place different content on different nodes according to their capabilities, which helps maximize server investment returns. For example, we can place dynamic content on some servers with powerful CPUs, and plain html content on a cluster of machines with fast disks.

In our Web site, some documents are mutable, which presents an interesting challenge for content placement and management in the distributed server. If these documents are replicated, some form of consistency control is required. When these documents are highly volatile, the complexity of maintaining consistency will place an undue burden on the server. Our proposed system provides a solution for this problem. We can separate such mutable content onto a dedicated server node. The content-aware distributor can separate requests (for these documents) from other Web requests and send them to the dedicated server. Because of this, consistency of object modifications by the content provider can be maintained by a centralized policy.

In addition, our Web server also provides content hosting services, which involves Web access information belonging to third-party content providers. The proposed system enables us to allocate different resources to providing differentiated levels of service according to the variety of content. We think this will be an attractive and helpful feature for Web hosting service providers. In such configurations, the variety of content from different customers means that the expectations and requirements on the quality of the hosting service differs, and the amount of money each customer is willing to and can afford to pay differs. With our proposed system, they can exert explicit control over resource allocation according to the variety of content.

5. System Evaluation

This section presents performance data of our system. Some data were measured from our Web site running the proposed system. We also constructed a test-bed in laboratory and observed how it performs under a variety of synthetic workload.

5.1 Test-bed and Workload

For the detailed performance evaluation, we constructed the following test environment in laboratory. We used a 350 MHz machine (with 128 MB memory) running Linux (with modified kernel) to serve as

distributor. The servers cluster consists of the following machines: three 150MHz machines with 64MB of memory and 4GB IDE disks, two 200 MHz machines with 128MB of memory and 4GB SCSI disks, and four 350 MHz machines with 128MB of memory and 8GB SCSI disks. Some of the back-end servers run Windows NT with IIS, and the others run Linux with Apache. The reason for such a configuration is that we want to show that the servers clustered by our mechanism can be heterogeneous. We used fastethernet network interfaces (100Mbps) on each node in order to allow enough throughput to show the clustered server's capabilities.

We used 24 Pentium 300MHZ machines (with 64M RAM) to generate a synthetic workload to evaluate the test platform. Each machine runs four WebBench [26] client programs that emit a stream of Web requests, and measure the system response. The stream of requests is called the workload. We created two workloads that model the Web server workload characterization (e.g., file size, request distribution, file popularity, etc.) published in papers [9,10,27]. The first workload (workload A) consists of static content, and the second workload (Workload B) includes a significant amount of dynamic content (e.g. CGI and ASP).

5.2 Overhead of Content-Aware Routing

The major concern of our system is that the additional overhead introduced by the distributor might become a performance impediment. We had measured the overhead in [24], which shows that the overhead is insignificant.

For each incoming HTTP request, the URL table must be consulted to make the routing decision. As a result, the process of locating the entry relevant to each incoming request can become a significant performance factor. To facilitate efficient access, insertions, and deletions to corresponding entry, we implemented the URL table as a multi-level hash table, in which each level corresponds to a level in the content tree. Each item of content in the Web site has a record corresponding to it in the URL table. Otherwise, we also implemented a mechanism to cache recently accessed entries, which is a proven [28] technique for demultiplexing speedup. We are aware that the overhead data (presented in [24]) do not reflect the real overhead introduced by lookup operations on the URL table, because such overhead is dependent on the size of the table and therefore on the amount of data stored in the web-site. As a result, we measured this overhead from our Web site running the proposed system. Our Web site contains about 8700 Web objects. In such scale, the memory consumed by the URL table is about 260k bytes. During the peak load, the average lookup time is about 4.32 μsecs, which is insignificant.

5.3 Benefit of Proposed System

We used WebBench with workload A to perform the first experiment on the following three configurations: (1) the entire set of files was replicated on each of the servers, (2) the entire set of files shared using NFS, (3) the document sets was dispersed with the content aware routing. In the case 1 and case 2, the experiments were conducted in the server cluster front-ended by a TCP connection router (performs Layer-4 routing), which is the implementation in our previous work [2]. In the TCP connection router, we implemented "Weight Least Connection" mechanism for load distribution. In the case 3, the experiment was performed in the server cluster front-ended by our content-aware distributor. We roughly partitioned the document tree by content type in the configuration 3. We also place large video file in the nodes with large volume and fast disk. Figure 2 shows the results in terms of throughput.

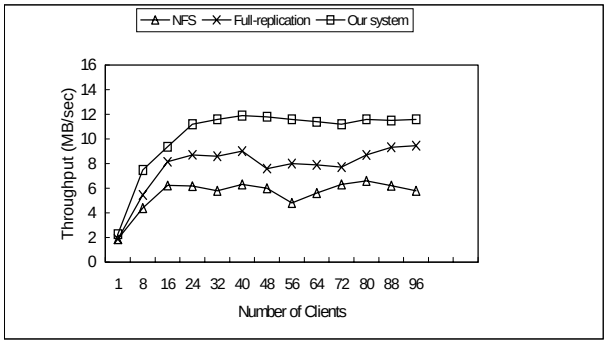

Figure 2. Benefit of content partition(Workload A)

It clearly shows that the server cluster with an NFS server performed very poorly compared to the other two content placement schemes. This is because the majority of the requested content could not be found locally on the Web servers, thereby the web server has to obtain the content from NFS server and then serve it to the client. This introduces significant additional latency and makes the NFS server to become the performance bottleneck. The content partition with content-aware routing consistently achieved a greater throughput than the other schemes. The reason for this higher performance is because in the content partition scheme each server only poses part of the content, so that each server sees a smaller set of distinct requests and the working set size is reduced. This greatly increases performance due to the improved hit rates in the memory cache, thereby a server could directly serve the request from its memory.

We used WebBench with Workload B to conduct the second experiment on the configuration 2 and 3. In our content-smart cluster (configuration 3), we separated dynamic content and static content on different servers. We placed dynamic content (CGI scripts and ASP) on the servers with powerful CPU, plain html content on the nodes with slow processor and disk. We also separated large file (e.g., video file) on the server nodes with fast

disk. Figure 3 shows the results.

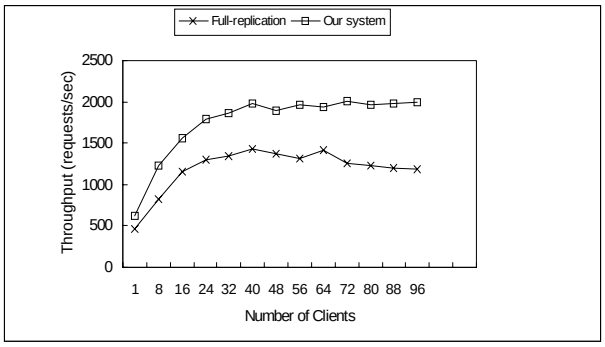

Figure 3. Benefit of content partition(Workload B)

The measurement metric is number of requests served per second. The results show that the throughput achieved with our proposed system outperforms that of content full-replication with Weighted-Least-Connection load distribution mechanism. The reason for the poor performance of later is because the content placement scheme (full-replication) does not take the heterogeneity on the capability of each node into consideration. For example, when a complex database query or a heavy request for a long-running CGI script is dispatched to the node with a slow processor, it will take orders of magnitude more time than the request is dispatched to the node with powerful processor.

In addition, this experiment also serves as a proof of the performance benefits of content-aware routing incorporated with content segregation. Figure 4 shows the throughput when the server was saturated by 120 concurrent WebBench clients. In the content-aware router with content segregation, the average CGI request, average ASP request, and average static request (request for HTML file, image, etc.) increased by 45 percent, 42 percent, and 58 percent respectively. The reason for this higher performance is because the content segregation prevents short Web requests from being delayed by long running request.

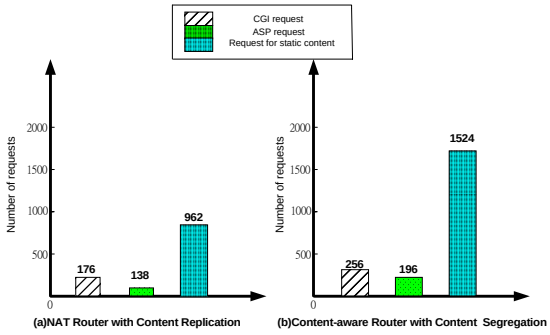

Figure 4. Benefit of content segregation

6. Related Work

Most of the prior work on distributed web servers has concentrated on request distribution among a fixed set of server nodes. We describe and discuss these approaches in section 2.1. Little attention has been given to the management problem of distributed Web servers, in particular the content management problem. To the best of our knowledge, the most closely related work is [5]. The authors describe several techniques learned from the management of Netscape's Web site. They perform request distribution in the Netscape Navigator browser itself with built-in knowledge. Obviously, this method can not take into consideration the current load or availability of the servers. Hence, this is not sufficient solution to the cluster-based servers. They also combined several existing tools (e.g., CVS and rdist) to perform content management. In contrast, our content-aware management system provides more benefits (as we discussed above) than their approach.

In addition, a number of monitoring approaches and systems for administrating distributed systems have been proposed (e.g., [29,30]). Although some of these systems have various design goals and objectives, they are insufficient to support a scalable and large-scale Internet service built on distributed server.

Recently, some commercial start-up companies (Arrowpoint [31] and Resonate [32]) have also proposed a similar idea of routing requests based on the requested content. Several features of our work differentiate it from these commercial products. First, our content-aware routing mechanism is different from theirs. Our mechanism reuses the pre-forked connection and seamlessly relays packets from the client-side to the pre-forked connection. Second, we provide an agent-based content management system that seamlessly integrates with a content-aware distributor. With this management system, we provide a complete solution to the content placement and management problems that arise in the clustered Web server. In contrast, these products are just front-end devices that perform content-based request distribution across a group of servers. The administrator would have to manually perform many management functions.

7. Conclusion

Server clustering is an important technique for building a high performance, reliable and scalable Web server. One of the important factors in efficient utilization of a distributed server is to be able to deploy content on each node according to its capability, and then direct clients to the best suited server. In this paper, we propose a content placement and management system to address this issue. The proposed system provides the web-site manager with great flexibility in selecting the optimum cost/performance configuration for their Web site. The Web site manager can partition the documents set by

hosting it on different servers. We demonstrate that segregation of dynamic content and static content can achieve better performance, in particular in the distributed server system. We also provide an agent-based management system to mask the complexity of such a distributed environment. With the innovative content management system, Web site managers can easily manage and maintain the distributed server as a single large system. The load balancing and auto-replication mechanism could further ensure an even load distribution and self-configure with respect to the change of content access pattern. In the future, we will further investigate more sophisticated load-balancing algorithm and its performance.

Acknowledgements:

This work was supported by the National Science Council, R.O.C., under contract no. NSC 89-2213-E-110-007.

References:

[1] A. Fox, S. Gribble, Y. Chawathe and E. A. Brewer "Cluster-based scalable network services," Proceedings of SOSP '97, St. Malo, France, October 1997.

[2] C. S. Yang, M. Y. Luo "Design and implementation of a environment for building scalable and highly available web server," Proceedings of 1998 International Symposium on Internet Technology, April 29- May 1, 1998.

[3] "Portals Provide Layer 4 Acid Test", http://www.techweb.com/wire/story/TWB19990315S0009.

[4] Digital Equipment Corporation. About Alta Vista. http://www.altavista.com/av/content/about.htm, 1995.

[5] D. Mosedale, W. Foss, and R. McCool. "Lessons learned: administering Netscape's Internet site," IEEE Internet Computing, 1(2): 28-35, 1997.

[6] A. Iyengar, E. MacNair and T. Nguyen "An analysis of web server performance," Proceedings of the IEEE 1997 Global Telecommunications Conference, Phoenix, November 1997.

[7] M. E. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection scheduling in Web servers," Boston University Computer Science Technical Report BUCS-TR-99-003, April, 1999.

[8] V. Almeida, A. Bestavros, M. Crovella, and A. Oliveira, "Characterizing reference locality in the WWW," Proceedings of 1996 International Conference on Parallel and Distributed Information Systems, December 1996.

[9] M. Arlitt and C. Williamson. "Web server workload characterization: The search for invariants," Proceeding of the 1996 ACM SIGMETRICS Conference, Philadelphia, PA, April 1996.

[10] M. Arlitt, T. Jin, "Workload Characterization of the 1998 World Cup Web site," Hewlett-Packard Technical Report, February 1999.

[11] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler, "Using smart clients to build scalable services," Proceedings of the 1997 USENIX Annual Technical Conference, January 6-10, 1997.

[12] R. McGrath T. Kwan and D.Reed. "NCSA's World Wide Web server: design and performance," IEEE Computer, November 1995.

[13] M. Garland, S. Grassia, R. Monroe, and S. Puri, "Implementing distributed server groups for the World Wide Web," Technical Report CMU-CS-95-114, School of Computer Science, Carnegie Mellon University, January 1995.

[14] E. Anderson, D. Patterson, and E. Brewer. "The magicrouter, an application of fast packet interposing," http://HTTP.CS.Berkeley.EDU/~eanders/projects/magicrouter/osdi96-mr-submission.ps

[15] O. Damani, P. Chung, Y. Huang, C. Kintala, and Y. Wang. "ONE-IP: techniques for hosting a service on a cluster of machines," Computer Networks and ISDN Systems, 29, 1997.

[16] D. Dias, W. Kish, R. Mukherjee, and R. Tewari, "A scalable and highly available web server," Proceedings of COMPCON'96, Santa Clara, CA, February, 1996

[17] IBM Network Dispatcher. http://www.ibm.com/software/enetwork/dispatcher

[18] CISCO. Local Director. http://www.cisco.com/

[19] F5Labs. BigIP. http://www.f5.com/

[20] Foundry Networks. ServerIron Server Load Balancing Switch. http://www.foundrynet.com, 1998.

[21] D. Andresen, T. Yang, V. Holmedahl, and O.H. Ibarra. "Sweb: towards a scalable world wide web server on multicomputers," In Proceedings of the 10th International Parallel Processing Symposium.

[22] T. Berners-Lee, R. Fielding, H. Frystyk, J. Gettys, J. C. Mogul. Hypertext Transfer Protocol – HTTP/1.1, http://www.w3.org/Protocols/

[23] G. Wright and W. R. Stevens " TCP/IP Illustrated, Volume1" Addison-Wesley, May 1994.

[24] C. S. Yang and M. Y. Luo, "Efficient support for content-based routing in web server clusters," Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, Colorado, USA, October 11-14, 1999.

[25] C. S. Yang and M. Y. Luo, "Design and implementation of an administration system for distributed Web server," Proceedings of the 12th USENIX Systems Administration Conference (LISA'98), Boston, MA, December 6-11,1998.

[26] WebBench, http://www.zdbop.com

[27] P. Barford and M. E. Crovella, "Generating representative web workloads for network and server performance evaluation," Proceedings of ACM SIGMETRICS '98, Madison WI, 1998.

[28] J. C. Mogul, "Network locality at the scale of processes," ACM Transactions on Computer Systems, May 1992.

[29] E. Anderson and D. Patterson, "Extensible, scalable monitoring for clusters of computers," Proceedings of 11th USENIX Systems Administration Conference, San Diego, California, USA, October 26-31, 1997.

[30] E. Al-Shaer, H. Abdel-Wahab and K. Maly, "HiFi: A new monitoring architecture for distributed systems management," Proceedings of 19th International Conference on Distributed Computing Systems, Austin, 1999.

[31] Arrowpoint. http://www.arrowpoint.com/

[32] Resonate, http://www.resonate.com.