Proceedings of USITS' 99: The 2nd USENIX Symposium on Internet Technologies & Systems

Boulder, Colorado, USA, October 11–14, 1999

EFFICIENT SUPPORT FOR CONTENT-BASED ROUTING IN WEB SERVER CLUSTERS

Chu-Sing Yang and Mon-Yen Luo

Efficient Support for Content-Based Routing in Web Server Clusters

Chu-Sing Yang and Mon-Yen Luo

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, R.O.C

Abstract

Clustered server architectures have been employed for many years on the Internet as a way to increase performance, reliability and scalability in the presence of the Internet's explosive growth. A routing mechanism for mapping requests to individual servers within cluster is at the heart of any server clustering techniques. In this paper, we first analyze the deficiencies of existing request-routing approaches. Based on these observations, we argue that the request routing mechanism in a cluster-based server should factor in the content of a request in making decisions. Thus, we designed and implemented a new mechanism to efficiently support content-aware routing in Web server clusters. With this mechanism, we also built in a number of sophisticated content-aware intelligence for making routing decision. Performance evaluation on a prototype implementation demonstrates substantial performance improvements over contemporary routing schemes. The proposed mechanism can also enable many new capabilities in cluster-based servers, such as sophisticated load balancing, differentiated service, special content deployment, session integrity, etc.

1. Introduction

The Internet, in particular the World Wide Web, has experienced explosive growth and continues to expand at an amazing paces [1]. This has resulted in heavy demands being placed on Internet servers and has raised great concerns in terms of performance, scalability and availability of Web services. A monolithic server hosting a service is usually not sufficient to handle these challenges. Cluster-based server architecture has proven [2,3,4] a successful and cost effective alternative to build a scalable, reliable, and high-performance Internet server system. In fact, popular Web sites are increasingly running Internet services on a cluster of servers (e.g., Alta Vista [5], Inktomi [6], Netscape [7]), and this trend is likely to accelerate.

An important issue that arises in such clustered architectures is the need to dispatch and route incoming requests to the server best suited to respond. Over the past few years, a considerable number of researchers and vendors have proposed methods for distributing the user requests in such clustered server. We classify these schemes based on where in the layer the request-distribution function is applied:

- **Client-side approach**
  First, request routing can be achieved through a Java applet that is downloaded and executed at client side (e.g., [8,9]). The applet provides service-specific customizations that forward each client request to an appropriate server.
- **DNS-based approach**
  When a client tries to request a document from a Web server, the client-side browser first has to construct a DNS query to resolve the mapping of server hostname (from URL) to IP address. The DNS-based approach [10,11,12] performs routing decisions at this name resolution level. The name server at the server side can be customized to resolve the same name to different IP addresses for the purpose of request distribution.
- **TCP connection routing**
  Once the IP address is resolved, the client will try to establish a TCP connection to the server with the designated IP address. A number of researchers [4,13,14,15] and vendors [16,17,18,19,20] have proposed a connection-routing frontend to route user requests at this stage. These front-end devices make parallel services on different servers appear as a virtual service on a single IP address.
- **HTTP redirection**
  After the TCP connection setup is completed, the client then issues the HTTP request. At this stage, the HTTP redirection approach (used in [21]) may use one special response code *URL Redirection* (defined in the HTTP protocol [22]) to perform request routing. This is achieved by returning the address of the selected server instead of returning the requested data, asking the client to create a second connection and resubmit its request to that

server.

We have previously designed and implemented a cluster-based framework [4,23] for building a scalable and highly available Internet server. From our experiences using this framework to build many large-scale Internet service sites, we found a number of important issues that cannot be effectively addressed by existing routing mechanisms. Examples include session (or transaction) integrity, sophisticated load balancing, quality of service, and content placement. We discuss these issues in section 2. Based on these observations, we argue that in many cases the routing mechanism in a cluster-based server should factor in the content (e.g., URL or service type) of a request in making decisions. We refer to this concept as "content-based routing." We propose a new mechanism to efficiently support content-based routing in Web server clusters. The design of the proposed mechanism is presented in section 3. We implemented the mechanism as a specialized software component called *content-aware distributor*, which is a software module for kernel-level extension. We present the key portions of the implementation in section 4. Section 5 presents the results of performance evaluation of the prototype system. The results demonstrate substantial performance improvements over contemporary routing schemes. We compare our system with related work in section 6, and then present the conclusion in section 7.

2. Issues Ignored by Existing Routing Schemes

Existing approaches for routing requests to individual servers within cluster have typically concentrated on \user transparency, load distribution, and scalability. From our experience of running Internet services on a cluster architecture, we argue that a number of other issues are very important as well:

Session Integrity: Currently, the HTTP protocol is stateless, i.e., a Web server fulfils each request independently without relating that request to previous or subsequent requests. However, there are many situations (e.g., electronic commerce) where it is essential to maintain state information from previous interactions between a client and a server. For example, such state might contain the contents of an electronic "shopping cart" (a purchase list in a shopping mall site) or a list of results from a search request. When the user checks out of the shop, or asks for the next 25 items from a search, the state information from the previous request is needed. A number of schemes are employed (e.g., cookies [24] or hidden variables within HTML FORM [25]) or proposed [26] for handling state on the Web. Unfortunately, these methods might not be processed properly in a cluster-based server. If the routing mechanism do not examine the content of each request in selecting a server, it is possible a request belonging to a session is dispatched to the wrong server. This could limit the usefulness of cluster architecture because the state concept is an increasingly critical part of Web behavior for e-commerce, Web-oriented database, and other dynamic transaction applications.

Sophisticated Load Balancing: A server cluster requires some sort of load-balancing mechanism for directing requests in a way that utilizes the cluster resources evenly and efficiently. In current Web sites, the service type of incoming requests can be as varied as static Web pages, dynamic content generated by CGI scripts, or multimedia data such as streaming audio or video. The service time and the amount of resources consumed by each request vary widely and depend on several factors. For example, a request for executing a CGI script normally requires much more computing resources compared to static file retrieval requests [27]. This heterogeneity in request often causes skewed utilization of the server cluster. As a result, a more sophisticated load-balancing mechanism based on the service type of each request is essential. The load-balancing capability provided in many existing systems is still limited because they do not consider the service type of each request. Typical load-balancing strategies used in these systems include Round Robin [11, 14], Weighted Round Robin [17], Least Connections [18], Weighted Least Connections [17], and Monitored Server Load [17,18]. We previously implemented these mechanisms in our Web server cluster, and observed that these techniques still cannot deliver satisfactory performance when the workload characteristics of each request change significantly. Others have made similar observations [28, 29].

Differentiated Services: The Web continues to evolve from its initial role as a provider of read-only access to static documentation-based information, and is becoming a platform for supporting complex services. However, most current Web servers, both cluster-based and monolithic, provide service in a best-effort manner that does not differentiate between the requirements of different requests. This approach does not work well with advanced services and potential future needs. Different services may have different requirements for quality of service. If the routing schemes do not take the service type of each request into consideration, it is difficult to enforce priority policies and to provide the desired quality of service. Otherwise, not all content are equally important to the client and service provider. However, we notice that requests for popular pages have the tendency to overwhelm the requests for other critical pages (such as product list or shopping-related pages). As a result, enterprises and service providers (e.g., Web content hosting service providers) may want to exert explicit control over resource-consumption policies, to provide differentiated quality of service due to the variety of content.

Content Deployment: Given a cluster-based server, how to place and manage content in such a distributed system is an important and challenging issue. Because the incoming requests may be distributed to any server node in the cluster, each participant server must have the same capability for responding to requests for any portion of resource that the Web site provides. Typically, this requirement can be achieved by a shared network file system or full replication of all content on each server. However, neither of these two schemes is a satisfactory solution. The networked file system approach will suffer from the single-point-of-failure problem and increase user perceived latency by accessing data over the network file system. Full replication of content is expensive in terms of space utilization, and will not work for some Web services (e.g., a Web service using a commercial database). Thus, there are times when either to reduce space requirements or for performance reasons, a Web site administrator wants to place different functions or content on different sets of servers. In cases where this is done for performance, specific machines can be dedicated to a particular task (e.g., executing CGI scripts) for performance optimization. To support such flexible content deployment, the routing scheme must be content-aware so that it can know which server hosts the requested content.

These observations lead to the inevitable conclusion that in many cases the routing mechanism in a cluster-based server should factor in the content of requests in making decisions. However, this requirement cannot be effectively addressed by existing routing mechanism. 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. For example, servers that appear to be underutilized may quickly become overloaded because everyone sends their requests to those machines until new load information is propagated. Both DNS-based and connection-router 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-based routing.

3. The Content-Aware Distributor

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.

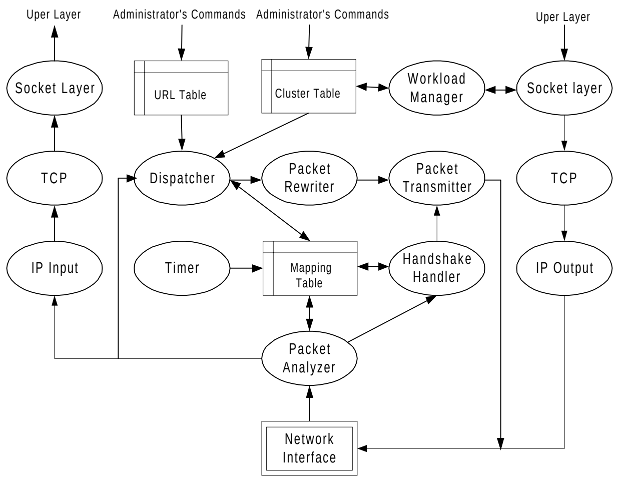

Figure 1. Functional Overview of Content-Aware Distributor

Figure 1 shows our design of the content-aware distributor. The operation of the content-aware distributor (distributor for short) is largely based on the following three data structures that hold vital information for routing a request: the cluster table, the mapping table and the URL table. Each entry in the cluster table represents one participant server in the cluster. The most important fields of the cluster-table entry include the IP address and MAC address of one participant server, a timer for failure detection, the load index (a measure of current load), and the available connection list (which will be described later).

The distributor also maintains and manipulates the mapping table for directing a request to the selected server. Each entry in this table is indexed by the source IP address and source port of the client, and also includes the IP address and port number of selected server, a state indication, TCP state information for the client and the selected server, and a timestamp. The URL table holds information (e.g., location of the document, document sizes, type, priority, etc.) that helps the dispatcher to make routing decisions.

To explain the operation of the distributor, we look at the sequence of events when a client requests a document from a Web server, and then show how the distributor operations fit into the packet exchange (see Figure 2).

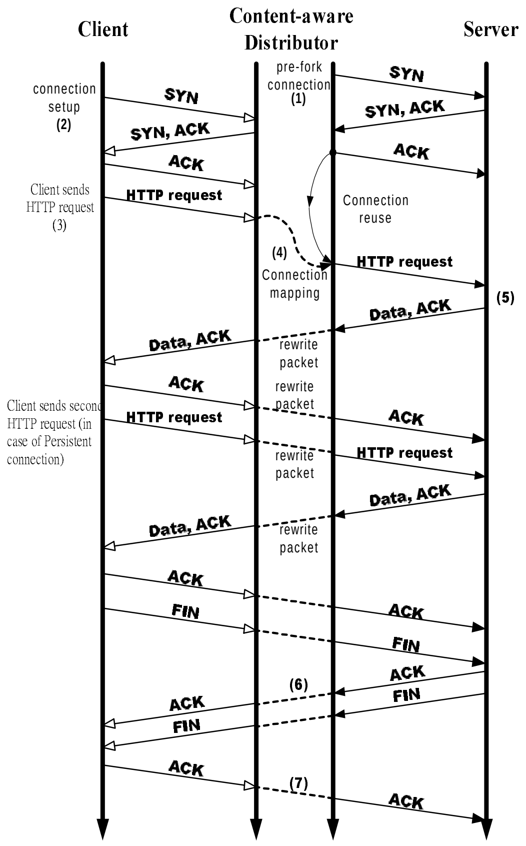

Figure 2. Operation of Content-Aware Distributor

3.1 Connection Setup

Whenever a client tries to initiate an HTTP request (position 2 in figure 2), 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 [30].

For each incoming packet, the packet analyzer inspects the information embedded in the IP and TCP header. If this packet is not destined for the Web service, the packet is passed up to the normal protocol stack. Otherwise, the mapping table is consulted to find the corresponding mapping information. If no information about this packet is found, the packet analyzer checks the SYN flag in the TCP header. If the SYN flag is set, representing the arrival of a new TCP connection, the handshake-handler module is invoked. The handshake-handler module first creates an entry in the mapping table for this connection and sets the status of this entry to SYN_RCVD. Then it handshakes with the client to complete the TCP connection setup. When connection establishment is completed, the status is changed to ESTABLISHED and the TCP state information (e.g., sequence number, ACK number, etc) is recorded in the mapping-table entry.

3.2 Transmitting the HTTP Request

After the TCP connection setup is completed, the client sends packets conveying the HTTP request for the specific content it is looking for. Once such a packet arrives, if the packet analyzer can find a corresponding entry in the mapping table and the status of this entry is ESTABLISHED, then the dispatcher module is invoked. The dispatcher parses the URL of this request, and then looks up the URL table and cluster table to select the server that possesses the requested content and is least loaded.

The cluster table is maintained by the workload manager which performs the following three functions. First, it estimates the current load on each server node to help the dispatcher make the optimal decision. Second, it monitors the health of the server nodes to bring them transparently in and out of service. Third, it pre-forks (i.e. creates a socket and establishes a TCP connection) a number of TCP connections to each server (position 1 in figure 2). These pre-forked connections are kept long-lived.

Once the dispatcher selects a target server, it also chooses an idle pre-forked connection from the available connection list. Then the dispatcher stores related information about the selected connection in the mapping table and changes the state to ACTIVE, which will bind the user connection to the pre-forked connection (position 4 in figure 2). The first objective behind such a design is to avoid the overhead of initiating a new connection to the selected server every time an HTTP request is made. Otherwise, it can increase user perceived performance by avoiding multiple slow-starts [31], because multiple successive requests conveyed by different user connections will be served by the same long-lived connection. We will demonstrate the advantages of this approach in section 5.

After the connection binding is determined, the packet-rewriter function is invoked to change the packet's IP and TCP headers for relaying the packet from the user connection to the pre-forked connection. After the packet's header is modified, the packet transmitter function is invoked for directing the packet to the selected node. The packet transmitter will find the network interface on which this packet should be forwarded. Then it constructs a network frame for transmission to the indicated interface. The forwarded frame has the MAC address of the network interface of the selected server.

3.3 Subsequent Data

When the designated server receives the request, it parses the URL to determine the content requested. It then fulfills the request and transmits the response to the client. The distributor also intercepts these response packets and performs the reverse packet modification so that the client can transparently receive and recognize these packets.

The distributor also handles the persistent connections suggested by HTTP 1.1 [22]. HTTP 1.1's persistent connections use one TCP connection to carry multiple HTTP requests, thereby reducing server load and client perceived latency [32]. Our mechanism inherently supports persistent connections. For HTTP 1.0 requests (non-persistent connections), the distributor can reuse the pre-forked connection to carry these requests, which will avoid extra TCP 3-way handshakes and multiple slow-starts. Otherwise, the distributor splits multiple HTTP 1.1 requests within a persistent connection into single requests. If these requests belong to the same session, the dispatcher routes them to the server assigned to the same session. Else, it schedules the individual HTTP 1.1 requests to different servers based on content of these requests.

3.4 Connection Termination

If the packet analyzer receives a FIN packet, which indicates that one host participating in this TCP connection intends to close this session, the state of corresponding entry in the mapping table will be changed to FIN_RECEIVED. Later, when the peer responds with an ACK packet to the FIN packet (position 6 in figure 2), the state is changed to HALF_CLOSED. When both FIN and ACK from two participating hosts in this TCP session arrive (position 7 in figure 2), the state is changed to CLOSED, and then the entry associated with this connection will be deleted after 2MSL (Maximum Segment lifetime, which is used to handle wandering packets in the Internet [30]).

If the client fails or loses connectivity with its server, there may be no FIN or ACK to arrive as anticipated above. This will result in accumulation of stale entries in the mapping table. To solve this problem, we maintain a timestamp for each TCP connection in the mapping table, which will timestamp the connection records each time a packet flows through them. The timer function periodically checks and deletes expired entries, defined as those that have been idle for a configurable amount of time. When one entry is deleted, the bound pre-forked connection is released to the available connection list, and it remains open for next client.

4. Implementation

For both efficiency and elegance, we implemented the proposed mechanism as a loadable module for Linux kernel. The kernel loadable module is a software component that can be dynamically loaded into kernel for the purpose of kernel extension. The distributor module inserts itself between the network interface (NIC) driver and the TCP/IP stack. By processing packets at this level, preventing them from traversing the protocol stacks, the distributor can analyze and route packets quickly. This effectively reduces the overhead of the distributor. To route incoming Web requests, all packets relating to Web service should go through the content-aware distributor. To achieve this, we can load the content-aware distributor module into the default router of the clustered Web servers.

Due to space limitations, we only present the key portions of the implementation. In particular, we focus on describing how the proposed mechanism can be implemented efficiently without imposing a significant amount of overhead. Because the ideas and mechanisms adopted in the content-aware distributor are generic, they should be applicable to other systems (e.g., BSD or Windows NT) as well. Some functions of the content-aware distributor are derived from our previous work [4,23]. Examples include failure detection and handling, workload evaluation and balancing, and a primary-backup mechanism for circumventing the single-point-of-failure problem. We do not describe these functions in this paper. For a detailed description, see [4, 23].

4.1 Hash Search

For all incoming packets, the mapping table must be consulted to find the corresponding mapping information. As a result, the process of locating the entry (binding information) relevant to each incoming packet can become a significant performance factor. To facilitate efficient access, insertions, and deletions to corresponding entry, we implemented the table as hash tables [33, 34] with doubly linked lists as collision resolution chains. Otherwise, we also implemented a mechanism to cache recently accessed entries, which is a proven [35, 36] technique for demultiplexing speedup.

The dispatcher also relies on information provided by URL table to make the routing decision. The URL table is implemented 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. Each entry includes information for making an intelligent decision. Examples include locations of the content, document sizes, type, priority, etc. The URL table is initialized and computed upon initialization of the distributor by scanning and parsing the content tree.

4.2 Packet Rewriting

To relay a packet from the user connection to the pre-forked connection correctly (vice versa), the packet rewriter must modify the packet's IP and TCP header before forwarding:

- Change source and destination IP address to that of the pre-forked connection.
- Update checksum in the IP header.
- Change source and destination port number (if necessary) to that of the pre-forked connection.
- Map the packet sequence number from the user connection to the pre-forked connection.
- Map the packet ACK number from the user connection to the pre-forked connection.
- Map TCP options (e.g., timestamp, window scale, and maximum sequence size) as needed.
- Update the TCP header checksum.

We have designed and implemented an algorithm to map the sequence number, ACK number, and TCP options from the user connection to the pre-forked connection. We find the idea behind the algorithm is similar to the TCP-splicing technique proposed by [37]. For the sake of brevity and space limitations, we omitted the description of this algorithm. The reader can see [37] for further information. In addition, changing these headers requires that the IP checksum and the TCP checksums be updated. Re-computing the entire checksum is expensive [38]. We implemented a method (described in [39]) to incrementally update the checksum.

4.3 Connection Management

The rationale of pre-forking long-lived connections (described in section 3.2) is to reduce overhead and latency of establishing a second connection. However, an idle open TCP connection incurs a system-resource cost (e.g., socket, buffer space, and PCB entry [40]). Thus, we designed and implemented a connection-management mechanism in the workload manager to strike a good balance between the benefits and costs of maintaining open connections. The connection-management module uses two parameters to control the number of pre-forked persistent connections: maximum number of connections (MAX for short) and minimum number of connections (MIN for short).

MAX defines the threshold between system trashing and maximum optimization of system resources. The workload manager periodically queries each server node for its current load to determine the value of MAX. The connection management mechanism keeps a fixed limit (MIN) on the number of available persistent connections. When the total connections (used and unused) reach MAX, it stops pre-forking connections. To date we use a trivial fixed length timeout policy for closing idle connections. An adaptive timeout mechanism may be more appropriate in some cases. This is an area of further research and beyond the scope of this paper.

4.4 Dispatcher

The dispatcher is responsible for making routing decisions based on content-aware intelligence. Up to now, we have implemented the following content-aware intelligence:

- **Affinity-Based Routing**

  A request routing function can reduce retrieval latency not just by balancing loads and maintaining low overhead but also through a third mechanism: affinity-based routing. An important factor to consider is that the latency for the server to serve a request from the disk is far longer than the latency to serve the request from the memory cache. With the content-aware mechanism, it is possible to direct requests for a given item of content to the same server so that the server can respond more quickly, due to an improved hit rate in the memory cache. In other words, requests are always dispatched to servers that already have data cached in main memory. Due to significant locality presented in user's references [41], the overall performance can benefit quite strongly from such a design. However, naively affinity-based routing achieves higher cache hit rates at the possible expense of load balancing. A number of algorithms have been proposed to overcome this problem. Examples include [42, 43]; our implementation is based on [42].

- **Dispersing Content Placement**

The content-aware mechanism enables a new content placement scheme, which allows the administrator to partition the content by hosting portions on different servers. The content partition may be governed by type (e.g., static HTML pages, CGI scripts, multimedia files, etc.) or by some other policy (e.g., priority). We implement some administration functions for administrator to configure the URL table to deploy the content tree. The new content-placement scheme offers several advantages over the traditional schemes. First, compared to full replication, this scheme yields better resource utilization and scalability while also avoiding the overheads associated with NFS approach. According to the traffic characterization of a modern Web site [44], 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 cluster. Second, we can place different content on different server optimized to address the requirements imposed by various data types. For example, we can place multimedia content on a server optimized for stringent real-time requirements. Finally, it provides the Web site manager with greater flexibility in selecting the optimum cost/performance configuration for their Web server. This is an important feature for cluster-based servers. Cluster-based servers tend to be heterogeneous because they generally grow incrementally as needed. In such a heterogeneous environment, some nodes may not be powerful enough to support an entire service, but can probably support some components of the service. The content-aware mechanism enables the administrator to place different content on different nodes according to their capability, which can maximize server investment returns.

- **Content Segregation**

From our previous experiences, we noticed that resource-intensive dynamic content processing (e.g. a complex database query) has the tendency to slow down the requests for static content, resulting in longer response times for these short requests. As a result, we suggest that the dynamic content should be segregated from the static content on different nodes. Another reason supporting the idea of content segregation is that the dynamic content is generally non-cacheable, so the affinity-based routing strategy is not directly applicable to it. We implemented a Weighted Least Connections algorithm to dispatch the requests for dynamic content. For requests to static content, the dispatcher performs routing decision with the affinity-based routing mechanism.

4.5 Future Directions

Because the function of the distributor is well defined and modularized, it could be easily extended or customized to meet unique system requirements. In the future, we will enhance the capability of the content–aware distributor by adding more content-aware intelligence. We will further explore more sophisticated load-balancing techniques based on the desired content of each request. Further, we will design and implement mechanisms for supporting session integrity so that the state shared by multiple requests in a session not be disrupted or delivered to wrong node. Finally, we also plan to design and implement some mechanisms to enable differentiated service in a clustered server. For example, it is increasingly common for a web site offering service both to paying subscribers and the public. Requests by paying customers should be given preferential treatment over those of nonpaying ones. The content-aware mechanism can include admission-control functions to reject some low-priority requests in highly overloaded circumstances, preventing the server from thrashing and guaranteeing the responsiveness of high-priority requests (e.g., requests by paying users, requests to critical pages such as home page or order form).

5. Performance Evaluation

In this section, we present our performance evaluation of the proposed mechanism. We used WebBench 3.0 [45] as the benchmark to evaluate the performance. The server cluster consists of the following machines connecting through 100Mbps Fast Ethernet: one Pentium 150MHZ machine (with 128M RAM) running the Linux operating system (with a modified kernel) serves as the distributor, and six Pentium 150MHZ machines (with 64M RAM) serve as back-end servers. 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 16 Pentium 150MHZ machines (with 64M RAM), which serve as WebBench clients.

5.1 Overhead of Content-Aware Routing

To quantify the overhead of the content-aware distributor, we measure and compare the response time of a Web request (for static content) traveling either across or not across a distributor for a variety of sizes. The overhead is defined to be the difference in response time between the two situations. We utilize WebBench to load the two configurations with many concurrent requests for the same file and then measure the response time. The results are given in Table 1. The values in the "baseline" are the response time for sending the request directly to the Web server. The additional overhead introduced by the content-aware distributor is acceptable. The latency via our content-aware distributor is about 4-10% slower than direct access. Notice that this experiment is performed over a local area network, where high-speed connections are the norm. The overhead would be insignificant when compared with the latency over wide-area networks with lower bandwidth. [46].

File size (Kb)	2K	8K	32K
Overhead (ms)	0.224	0.325	0.468
Latency(baseline)	5.124 ms	5.484 ms	7.42 ms
Percentage	4.8%	5.9%	6.3%
File size (Kb)	64K	256K	1024K
Overhead (ms)	0.767	2.325	9.524
Latency(baseline)	10.386 ms	26.973 ms	93.454 ms
Percentage	7.4%	8.6%	10.2%

Table 1 Overhead of Content-Aware Routing

We conduct another experiment to confirm that our design can effectively reduce the overhead. We first disable the pre-forking mechanism. That is, once the routing decision has been made, then the distributor creates a second TCP connection to the selected server. When the second connection has been setup, the distributor then starts to relay packets to the selected node. Under such a situation, the overhead increased by 89 μsecs. More importantly, such a change obliges the distributor to allocate memory buffer to queue the pending packets, increasing the burden of the distributor.

5.2 Benefits of the Proposed System

To demonstrate the benefits of the affinity-based routing mechanism, we first conducted experiments in the following environments: (1) 6 server nodes clustered by the content-aware distributor; (2) 6 server nodes clustered by the NAT (Network Address Translator [11,12]) router. The NAT router is the implementation in our previous work [4]. In the NAT router, we implemented a "Weighted Round-Robin" mechanism for load distribution, which is content-blind.

We used the WebBench benchmark with its standard static test suites to generate the client workload in these experiments. This standard static test suites define the workload WebBench uses to simulate traffic at actual Web sites. The workload is representative because it was created by examining log files supplied by many real Web sites, such as ZDNet, the Internet Movie Database, Microsoft, and USA Today [45]. Figure 3 shows the results in terms of throughput. It clearly shows that the server cluster with the content-aware distributor consistently achieved a greater throughput than the NAT cluster. The result also indicates that the additional overhead introduced by content-aware distributor can be compensated by sophisticated content intelligence.

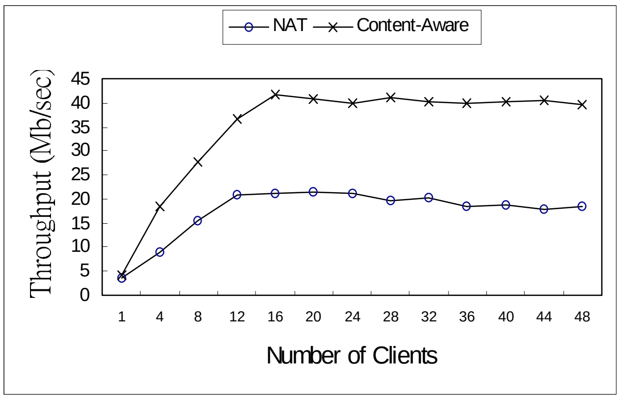

Figure 3. Benefit of Proposed System (Throughput)

We conducted another experiment to quantify the performance benefits of content-aware routing incorporated with content segregation. We used WebBench with a dynamic test suite (a mix of 20 percent CGI requests and 80 percent static requests) for this evaluation. We measured the performance under the same two environments described above. In the NAT cluster, the content sets are also replicated in each participating node. For the content-aware cluster, we separated dynamic content and static content on different servers.

Figure 4 shows the results. The results show that the throughput achieved with content-aware routing outperforms that of Weighted Round-Robin. In the content-aware router with content segregation, the average CGI request and average static request increased by 27 percent and 36 percent respectively. The reason for this higher performance is because the content segregation prevents short Web requests (e.g., request for static content) from being delayed by long running request.

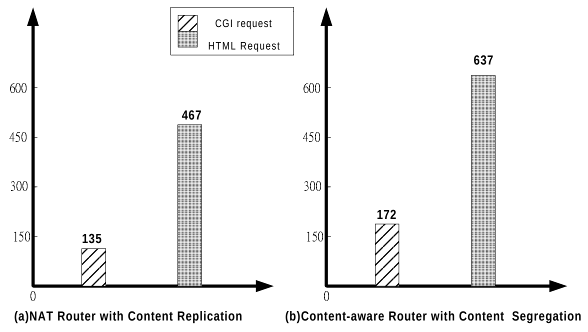

Figure 4. Benefit of Content Segregation (Average request/second)

5.3 Lessons Learned

Examining the results of our performance evaluation, it is noteworthy that the overhead increases with document size. This is because our mechanism performs some data-touching operations such as forwarding the packet and computing checksums. In our current implementation, packets from the server to the client must pass back through the distributor, and header modification is performed on each packet. If packets can go directly from servers to the client without having to pass through the distributor, the overhead can be further decreased. This is particularly beneficial while the amount of data sent from the server to the client is significantly larger than the amount of data sent from the client to the server.

Thus, we implemented a module that can be loaded into the kernel of the back-end server. The module can change the outgoing packets so that they can go directly to client. However, many Web site builders may not prefer this approach because it requires modification of the back-end server's kernel. In contrast, our previous implementation can be applied to any existing Web site without modification of backend server, but it pays a performance penalty. We suggest that the administrator can use the first approach for those nodes executing kernels that cannot be modified, and apply the second approach to those nodes that we can modify for performance gain.

6. Comparison

In this section, we compare our mechanism with the existing methods and related work. We discuss the advantages and disadvantages of various mechanisms.

NCSA [10,11] first proposed a clustering technique that uses the DNS-based approach, which has the advantages of low overhead and ease of implementation. The main weakness of this approach is that the name-to-IP-address mapping can be cached by multiple levels within the hierarchy of DNS service [46], bypassing address resolution. Thus, all requests behind a particular name server will be directly sent to the same server node until the cached address expires. This will lead to significant load imbalance, which has been quantified in [48,49], and several researches have attempted to overcome the problem [50,51,52]. However, another problem is that internal changes of the clustered server will propagate slowly through the Internet, due to address caching. That is, if one server node fails or is removed temporally for maintenance, a number of clients may continue trying to access the failed server using the cached address. In contrast, our mechanism is much faster than the domain name service in detecting failures and responding to it.

The connection-routing approach [13-20] can achieve fine-grained control in incoming requests compared to the DNS-based approach. However, since all connections to a server cluster have to pass through the front-end, it is a single point of failure and may become a bottleneck at high loads. Additionally, because they do not look into the content of requests, they are incapable of using more sophisticated load balancing policies. The request distribution strategies used in them generally are variations of weighted round-robin (or weighted least connections).

The HTTP redirection approach has the potential to support content-aware routing. The main disadvantage of this approach is that a request may require two or more connections for getting the desired service, which will increase the response time and network traffic. Furthermore, every request is initially addressed to the "scheduler" server, which also creates a single point of failure and the potential for a bottleneck due to servicing redirects.

The major advantage of our approach is that it can effectively support content-aware routing. In contrast, none of the schemes described above actually support content-aware routing, which will limit the usefulness of their server clusters. We have shown that content-aware routing is essential in many cases. Our implementation can be easily extended or customized to add more content-aware intelligence.

Recently, some commercial start-up companies have also proposed similar idea of content-based routing. Examples include Arrowpoint [53], Resonate [54], and HydraWeb [55]. Several features of our works differentiate it from these commercial products. First, we give a detailed description of implementation and performance evaluation. We believe that some of our experience may be helpful in this area. Second, our approach reuses the pre-forked connection and seamlessly relays packets from the client-side connection to a pre-forked connection. We have demonstrated that such a design can decrease latency. In contrast, the product by Resonate defers opening the back-end TCP packet until it sees the content of this request, and then decides which server to redirect (and open/SYN) the packet to. It then encapsulates the entire IP packet, as it was sent from the client, and sends it to the selected server. Third, we have discussed the content-aware issue more widespread (e.g., session integrity, affinity-based scheduling, differentiated service) and have given some viable solutions to the problem.

Authors in [43] proposed a routing strategy called LARD for content-based request distribution. With LARD, the front-end may distribute incoming requests in a manner that achieves high locality in the back-ends' main memory caches as well as load balancing. Their work is focused on algorithm design and simulation for validation. Although they proposed a protocol to migrate established TCP connections, they did not describe how to implement their approach. In this aspect, our work is complementary to theirs. We provide a general and implementable solution for supporting content-aware routing and propose optimization techniques to make the mechanism efficient. In addition, their protocol requires modifications to the TCP implementation of backend server's kernel.

The major drawback of our approach is the extra processing overhead. However, this overhead can be minimized by efficient implementation, which is discussed in section 4. The performance-evaluation results show that the overhead is not substantial. In addition, we also demonstrate that the additional overhead can be compensated for by sophisticated content intelligence (e.g., direct requests to the best-fit node).

A second possible drawback is limited scalability. Due to the additional processing overhead of examining the content of each request, the distributor may become the impediment to scaling the server. To eliminate this problem, we can combine the proposed mechanism with a DNS-based scheme to further improve scalability. That is, a number of distributors can be used when the single distributor becomes a performance bottleneck, and the DNS-based approach can be used to map different clients to different distributors.

Finally, the distributor represents a single point of failure, i.e., failure of the distributor will bring down the entire Web server. Questions of fault tolerance are beyond the scope of this paper, but they will be discussed in detail in a future work [56]

7. Conclusion

Web server clustering is an important technique for constructing a high-performance, reliable and scalable Web server. However, the deficiencies of existing request-routing mechanisms limit the usefulness of the cluster-based architecture. In this paper, we analyze these problems and then argue that the request-routing mechanism should factor in the content of a request in making decisions. We designed and implemented an efficient mechanism to support content-aware routing. The performance evaluation results show that the additional overhead introduced by the mechanism is insignificant. With this mechanism, we also built in a number of sophisticated content-aware intelligence for making routing decisions. Performance evaluation on a prototype implementation demonstrates substantial performance improvements over state-of-the-art routing schemes that use only load information to distribute requests.

Our content-aware mechanism can enable many new capabilities in cluster-based server, such as sophisticated load balancing, differentiated service, special content deployment, session integrity, etc. While the usefulness of most existing Web server clustering schemes were constrained by lack of content-aware intelligence, our mechanism can enable many new services such as electronic commerce, database searching, and Web content hosting on cluster-based servers. This will dramatically increase the usefulness of the Web-server clustering technique.

Acknowledgements:

We would like to thank our anonymous reviewers for their valuable comments. We also wish to express our gratitude to Dr. Geoff Kuenning for his proof-reading and making a number of helpful suggestions.
This work was supported by the National Science Council, R.O.C., under contract no. NSC 88-2213-E-110-022.

References:

[1] Internet Weather Report (IWR). Available at http://www.mids.org.

[2] T. E. Anderson, D. E. Culler, and D. A. Patterson, "A case for NOW (Networks of Workstations)," IEEE Micro, 15(1):54-64, February 1995.

[3] A. Fox, S. Gribble, Y. Chawathe and E. A. Brewer, "Cluster-based scalable network services," Proceedings of SOSP '97, pp. 78-91. St. Malo, France, October 1997.

[4] 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, pp. 124-131 April 29- May 1, 1998.

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

[6] Inktomi Corporation. The Inktomi Technology Behind HotBot, a White Paper. http://www.inktomi.com

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

[8] 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, pp. 105-117, January 6-10, 1997.

[9] Y. M. Wang, P. Y. Chung, C. M. Lin, and Y. Huang, "HAWA: a client-side approach to high-availability web access," Proceedings of the Sixth International World Wide Web Conference, April 1997.

[10] E.D. Katz, M. Butler, and R. McGrath, "A scalable HTTP server: the NCSA prototype," Computer Networks and ISDN Systems, 27:155-164, 1994.

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

[12] T. Brisco, "DNS support for load balancing," RFC 1794, http://www.internic.net/ds/

[13] H. Y. Yeom, J. Ha, and I. Kim, "IP multiplexing by transparent port-address translator" Proceedings of the 10th USENIX System Administration Conference (LISA X) Sep. 29 – Oct. 4, 1996 Chicago, IL, USA.

[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 the COMPCON'96, pp.85-92, Santa Clara, CA, February 1996

[17] IBM Corporation. The IBM Interactive Network Dispatcher 1998. http://www.ics.raleigh.ibm.com/netdispatch

[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." 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] 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), pp. 131-140, Boston, Massachusetts, December 6-11, 1998.

[24] Netscape Communications Corporation. Client Side State - HTTP Cookies. http://www.netscape.com/newsref/std/cookie spec.html.

[25] D. Raggett, A. Le Hors, and I. Jacobs, HTML 4.0 Specification, W3C Working Draft, (1997).

[26] D. Kristol and L. Montulli, "HTTP state management mechanism," RFC 2109, Feb. 1997.

[27] A. Iyengar, E. MacNair and T. Nguyen, "An analysis of web server performance" Proceedings of the IEEE 1997 Global Telecommunications Conference (GLOBECOM '97), Phoenix, AZ, November 1997.

[28] H. Zhu, T. Yang, Q. Zheng, D. Watson, O.H. Ibarra and T. Smith, "Adaptive load sharing for clustered digital library servers," Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC-7), Chicago, July 1998.

[29] H. Zhu, B. Smith and T. Yang, "A scheduling framework for Web server clusters with intensive dynamic content processing" Technical report, university of California, Available at http://www.cs.ucsb.edu/~hczhu/publications/rcgi.ps

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

[31] J. Heidemann, "Performance interactions between P-HTTP and TCP implementations." ACM Computer Communication Review, 27 2, 65-73, April, 1997.

[32] J. C. Mogul, "The case for persistent-connection HTTP," Proceedings of the SIGCOMM'95, pages 299-313. ACM, August 1995.

[33] R. Jain, "A comparison of hashing schemes for address lookup in computer networks," IEEE Transactions on Communications, October 1992.

[34] P. E. McKenney and Ken F. Dove, "Efficient demultiplexing of incoming TCP packets," ACM SIGCOMM 92, August 1992.

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

[36] C. Partridge and S. Pink, "A faster UDP," IEEE/ACM Transactions on Networking, July 1993.

[37] D. Maltz and P. Bhagwat. "TCP splicing for application layer proxy performance," IBM Technical Report RC-21139, March 1998.

[38] B. Braden, D. Borman, and C. Partridge, RFC 1071: Computing the Internet checksum, Sept. 1988.

[39] A. Rijsinghani, RFC 1624: Computation of the Internet checksum via incremental update, May 1994.

[40] J. C. Mogul, "Operating system support for busy Internet servers," Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V),Orcas Island, WA, May 1995

[41] S. D. Gribble and E. A. Brewer, "System design issues for Internet middleware services: deductions from a large client trace," Proceedings of the 1st USENIX Symposium on Internet Technologies and Systems, December 8-11, 1997, Monterey, California, USA

[42] D. Thaler and C. Ravishankar, "Using name-based mappings to increase hit rates," IEEE/ACM Transactions on Networking, Vol. 6, No. 1, February 1998.

[43] V. Pai, M. Aron, M. Svendsen, G. Banga, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-aware request distribution in cluster-based network servers," Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, October 1998.

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

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

[46] P. Barford and M. E. Crovella, "Measuring Web performance in the wide area," Performance Evaluation Review, August 1999. Also available at http://www.cs.bu.edu/faculty/crovella/paper-archive/wawm-per.ps

[47] P. Mockapetris, Domain Names - Concepts and Facilities, Information Sciences Institute, University of Southern California, November 1987. RFC 1034.

[48] J. C. Mogul, "Network behavior of a busy Web server and its clients." Technical report, Digital Western Research Lab, October 1995. WRL Research Report 95/5.

[49] M. Colajanmi, P. S. Yu, and D. M. Dias, "Scheduling algorithms for distributed web servers," Proceedings of the 17th International Conference on Distributed Computing Systems, pages 169-176, Baltimore, MD, May 1997.

[50] M. Colajanmi and P. S. Yu, "Adaptive TTL schemes for load balancing of distributed web servers," ACM SIGMETRICS Performance Evaluation Review, 25(2):36-42, September 1997.

[51] M. Colajanni, P. S. Yu, D. M. Dias, "Analysis of task assignment policies in scalable distributed Web-server system," IEEE Transaction on Parallel and Distributed Systems, vol. 9, no. 6, June 1998.

[52] V. Cardellini, M. Colajanni, P. S. Yu, "Efficient state estimators for load control policies in scalable Web-server cluster," Proceedings of IEEE 22nd Int. Computer Software and Application Conference (COMPSAC'98), Vienna, Austria, Aug. 1998.

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

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

[55] HydraWeb. http://www.hydraweb.com/

[56] C. S. Yang, M. Y. Luo and C. W. Tseng, "Fault-tolerance Web Server", Submit for publication.