Design and Implementation of an Administration System for Distributed Web Server

C. S. Yang and M. Y. Luo
National Sun Yat-Sen University, Taiwan, R.O.C.

Design and Implementation of an Administration System for Distributed Web Server

C. S. Yang and M. Y. Luo – National Sun Yat-Sen University, Taiwan, R.O.C.

ABSTRACT

The explosive growth of the World Wide Web has raised great concerns regarding many challenges – performance, scalability and availability of the Web system. Consequently, Web site builders are increasingly to construct their Web servers as distributed system for solving these problems, and this trend is likely to accelerate. In such systems, a group of loosely-coupled hosts will work together to serve as a single virtual server. Although the distributed server can provide compelling performance and accommodate the growth of web traffic, it inevitably increases the complexity of system administration. In this paper, we exploit the advantages of Java to design and implement an administration system for addressing this challenging problem.

Introduction

The explosive growth of the World Wide Web (WWW for short) [1] has raised great concerns regarding many challenges – performance, scalability and availability of the Web system. Due to the exponential growth of the World Wide Web, many popular Web sites are being overwhelmed by huge requests and suffer from server overload. In order to cope with the continued increasing demand on their Web servers, Web site managers must continually increase the server's capacity for providing the desired levels of quality of service. Upgrading the server to a more powerful machine may only solve the problem in the short term and might not be cost-effective in the long term. An increasingly popular solution for these problems is to deploy a set of computers, and enable them to work together as a single virtual server. A Web server based on such approach can provide the scalability necessary to keep up with growing client demand at popular Web sites. It could allow additional processing power to be dynamically added to increase the total capacity of the server as demand grows. The other key advantage of such approach is that one can more easily build a server that can tolerate hardware or software failures.

However, although such distributed server can provide compelling performance and accommodate the growth of web traffic, it inevitably increases the complexity of system administration. Failures, performance inefficiencies, content management, resource allocation, security compromises, and accounting are some of the problems associated with the operation of traditional server. When the server is composed of a group of loosely-coupled machines, the administrative burden will be larger. Reliable operational status of such distributed server, unless made in an automated way, requires significant human effort for propagating one administrative function to all nodes. In particular, such distributed servers tend to be more heterogeneous, and this heterogeneity will come at the cost of greatly increased management costs. As a result, effective management mechanisms and tools are required to mask the complexity and heterogeneity of internal composition of such distributed server, and manage the server composed of several nodes as easy as manage a single node. In this paper, we describe the work we are pursuing in exploiting the advantages of Java [2] to design and implement an administration system for addressing these problems. With the innovative administration system, the Web site manager can manage and maintain the distributed server as a single large system. In addition, the administration system running in the background can also make the clustered server a better place to work.

The rest of this paper is organized as follows. The next gives an overview of distributed Web server. Subsequently, we describe the management problems raised by such an environment, and then present an overview of the architecture of proposed system. The details of implementation are given in the next section. Then, we discuss some related issues and compare our system with other related works, before presenting the conclusion and future work.

Overview of Distributed Web Server

With the rapid growth of the Internet, and the Web in particular, it is difficult for organizations and people running web sites to predict future demands that clients will place on their servers. Consequently, the Web server should be designed to be capable of meeting the evolving demand constantly and easily. That is, if the offered load begins to exceed the server's capabilities, there should be an easy way to scale up the system when the hardware or software of the existing server does not need to be replaced. In addition, as more and more commercial applications appear on the WWW, it has become apparent that the function performed by these servers is critical. Consequently, making these servers highly available is another problem that is gradually getting attention.

For addressing these problems, we [3,4,5] designed and implemented a scalable and highly available Web server. Figure 1 illustrates the overview of our system. The entire server is composed of a group of interconnected machines. For providing the illusion of single virtual server across these machines, the entire severs cluster should be presented to the outside world by single addressing interface (e.g., single Domain name). Consequently, we designed and implemented a distributing mechanism to spread all incoming Web requests destined for this addressing interface. Such a mechanism contains the following three functions: load balancing, failure detection, and directing. That is, when a new HTTP request arrives, some load balancing mechanisms are invoked for selecting a suitable node to serve the request, and such a mechanism could ensure that the workload is evenly spread among these server nodes. If one server node goes down, or during periods of maintenance, the distributing mechanism could discover it and respond by directing new requests to the other available nodes. Finally, a directing mechanism is required for directing the request to the selected node, in a manner that is transparently to the outside user. We [3,4] analyzed the TCP/IP [6,7] and HTTP [8,9] protocols that the web is based on, and we found out a number of ways in which one could direct requests to a selected node. After evaluating the tradeoff among these methods, we [4,5] choose reroute TCP connection as the directing method in our system. We extend the Linux kernel to build in all mechanisms mentioned above for fulfilling the distributor, which performs the distributing mechanism in the distributed server.

One node in the system will run the modified kernel to serve as distributor for distributing incoming web requests; the other nodes will execute Web server software responding the incoming HTTP request. Each host participating in the clustered server has a unique IP address, but only the distributor's IP address is associated with the domain name representing this

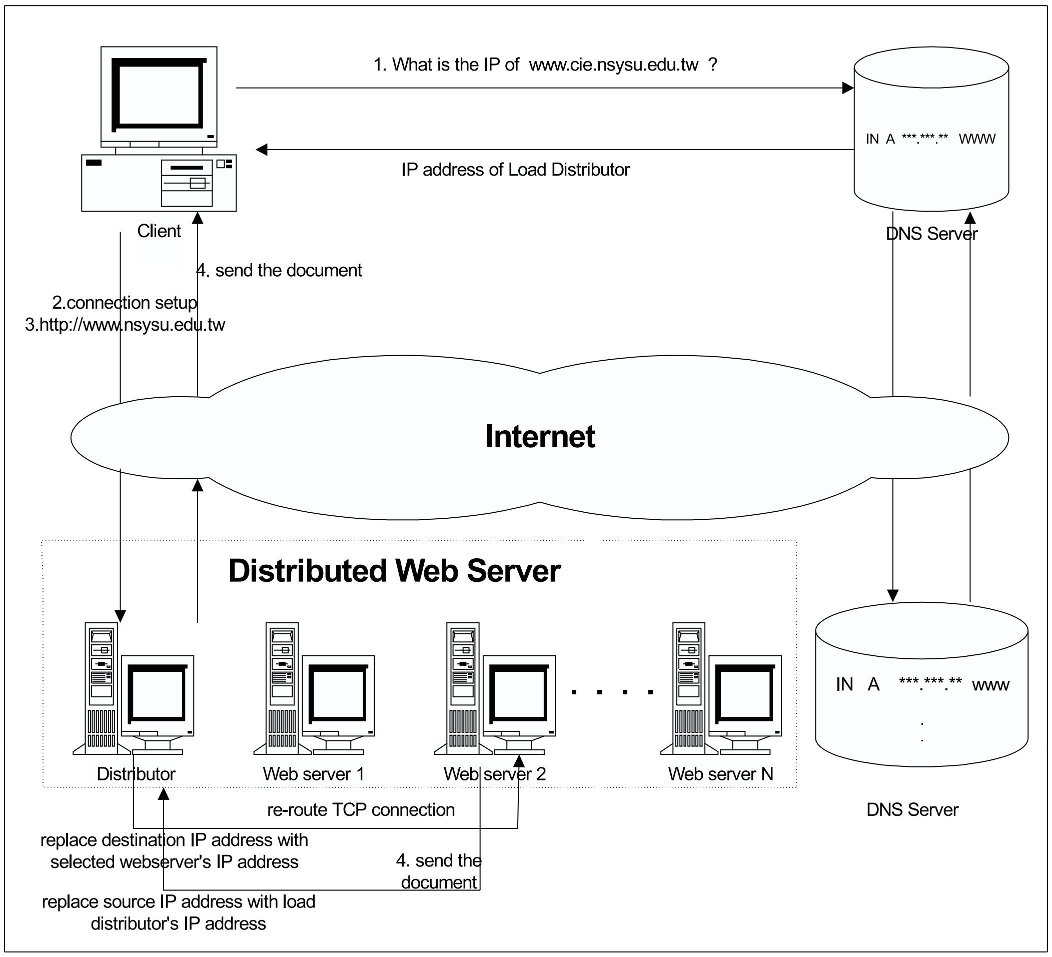
1. What is the IP of www.cie.nsysu.edu.tw ?
IN A ***.***.** WWW
IP address of Load Distributor
Client
DNS Server
4. send the document
2.connection setup
3.http://www.nsysu.edu.tw
Internet
Distributed Web Server
IN A ***.***.** www
Distributor
Web server 1
Web server 2
Web server N
DNS Server
re-route TCP connection
replace destination IP address with selected webserver's IP address
4. send the document
replace source IP address with load distributor's IP address

Figure 1: Overview of Distributed Web Server.

web site. As a result, only the distributor's IP address is advertised to the outside world, so that all HTTP requests destined for this web site will be delivered to distributor. The distributor will forward these incoming requests to the selected nodes via distributing mechanism.

In addition, as the incoming requests may be distributed to any node in the system, we also must provide a mechanism to ensure that each server node would look and feel identical to the requesting clients. In other words, we must make each node with the same capability of responding to requests for any portion of resource that the web site provides. For solving this problem, all content provided by the web site can be served from a centralized network file system. However, accessing data over the network file system can be very slow due to the overhead of LAN congestion. Furthermore, such a design will suffer from the single-point-failure problem, which will make the entire system more vulnerable. As a result, we choose an alternative approach as the content sharing method in our system: replicating all content on the local file system of each server nodes.

With these mechanisms, although all machines participating in the servers cluster are autonomous, they will seamlessly work together to serve the requests and provide the illusion of a single server. The new machine can be dynamically added to increase the total capacity of the server as demand grows. We expect that this approach should be attractive, because it can preserve the previous financial investment and reduce the costs of scaling the server. The other key advantage of this architecture is that we can more easily build a server that can tolerate hardware or software failures.

Although the foregoing system can provide a degree of high availability, the failure of distributor will still bring down the entire web server. We designed a primary-backup mechanism for addressing such a problem. One backup distributor can be setup to prevent the problem of single point failure. The primary distributor will broadcast a message periodically in addition to perform the distributing mechanism. The backup distributor threats the message as "heartbeat" of the primary distributor. It performs the same mechanism as just mentioned for detecting the failure of primary distributor, and takes over the responsibility of distributing request when the primary distributor fails.

Proposed System

Design Consideration

Although this system can accommodate growth of web traffic, it inevitably increases the complexity of system administration. Unlike traditional single-server configuration in which the web site manager can has full control on the whole system easily. When the entire Web server is composed by a group of loosely-coupled machines, the administrative burden of managing and maintaining the entire system will be considerable. Consequently, we intend to provide an administrative mechanism for web site manager to mask the complexity and heterogeneity of internal composition of the distributed server, and manage the server composed of several nodes as easy as managing a single node. Such a mechanism should address the following problems to which an administrator of distributed Web server would like to have answers:

- **Content management.** To guarantee that any request could access any resource regardless of the server to which that it is directed, we replicate all content on the local file system of each server nodes. However, the content of a Web site may change over time. When a change occurs, the system must propagate that change throughout the entire Web site. Usually, such changes need to be updated by manager manually. To address the problem, the proposed administrative mechanism should enable the web site manager to be capable of performing functions on all nodes at once.
- **Configuration.** We should provide a mechanism for administrator to know which node is operating as a part of the system. In addition, adding or removing a node should be an easy way and not require the extensive reconfiguration of all other nodes.
- **Self-diagnose.** If any failure or specific condition occurs in the system, the system should automatically inform the administrator by E-mail or other ways. Otherwise, the troubleshooting will become administrator's nightmare when they face such a complex system.
- **Performance and health monitoring.** We should provide a mechanism for administrator to monitor the status (e.g., resource utilization) of each node, ensure the resources provided by the web site are operational, and specified content can be delivered.
- **Security Concerns.** We should provide a mechanism (i.e., watch the log files created by each server node) to identify security problems or other situations.

System Architecture

To fulfill the administration system, we intended to construct a group of daemons running on each node and enable them to cooperate for performing the administrative functions. However, many problems arose when we tried to implement such an idea. The first major problem is platform heterogeneity, which arises from the fact that we hope the proposed system is flexible enough that each server node can use any kind of hardware, operating system, and Web server software. This means that these daemons that make up the administration system must be capable of running on different platforms. The second considerable problem is extensibility (or versioning and distribution problem). That is, the functionality of this administration system cannot be extended or modified without rewriting, recompilation, reinstallation, and re-instantiation of all existing daemons.

For tackling these problems, we decided to construct the administration system by Java [2,10]. Java is developed to support applications in a heterogeneous environment, which requires that applications be capable of executing on a variety of hardware architectures and operating system. This is achieved by generating an intermediate code called bytecode, which is an architecturally neutral format designed to be transported easily to multiple hardware and software platform. Such a design can relieve both the developer and user of concerns related to heterogeneity of the target platforms. Thus it is the primary reason that we choose Java to implement the administration system. In addition, the another attractive feature of Java is the notion of downloaded executable content (data that contain programs that are executed upon receipt). Base on this, we can implement each administration function in the form of Java class instead of implementing it in the daemon. All daemons distributed on each node will download the appropriate classes from a central location to perform the management task. Such a design will avoid the software distribution problem. If a new capability needs to be added, all we have to do is to implement a new Java class. We expect that using the capability of downloadable code from Java should provide unlimited possibility to enhance the function of the administration system.

The Java-based administration system is composed of the following four key components: controller, broker, agent, and remote console. Figure 2 illustrates the overall architecture of the proposed system. The broker will run on each Web server node to perform the administrative function, and monitor the status of the managed node. The administrative functions will be implement in agent, which is in the form of Java class. One special daemon called controller will be responsible for receiving request from administrator, and then invokes brokers to perform the delegated tasks by dispatching the corresponding agent. The remote console is a Java applet, which can be run on any Java-enabled Web browser. The administrator can download the remote console and interact with it to perform management operations.

Implementation

In this section we describe the implementation details and the current status of the proposed system.

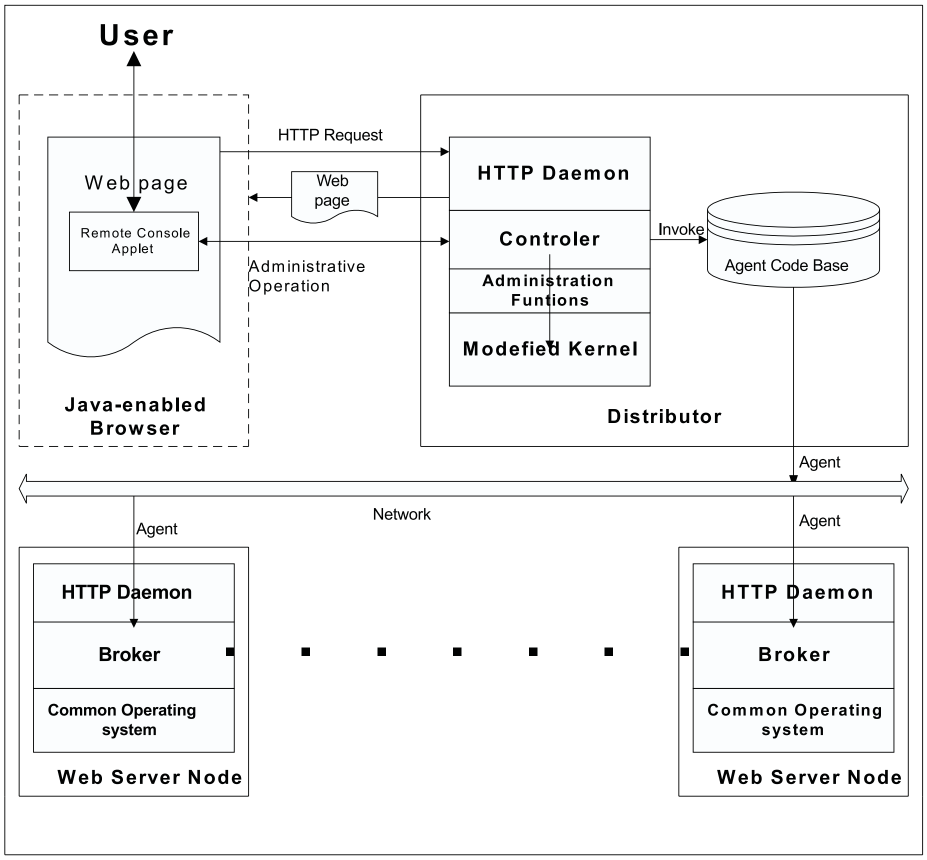

Figure 2: Overall Architecture of the Proposed System.

Control Interface

Because we made the distributor the control center of system administration, as well as distributing requests, we provided several administrative functions for the system administrator to control the system such as the following:

- Turn on/off the distributing mechanism.
- Join/remove a server node into/from the system.
- Read the related statistical data, such as outstanding request of each server nodes.

These functions are user-level programs written in C language. Because all related data are located in the kernel, we also implemented several new system calls that provide an interface for these user-level routines to access kernel-level data.

Controller

The controller is a standalone [11] Java application, which runs as a background process on distributor as the control center of system administration. The main function of controller is to respond the request from system administrator. When the controller starts as a process under the kernel, it forks a main thread to register a socket on a pre-assigned TCP port and block itself on listening mode for any incoming request. Upon a request arriving, the main thread will create a new working thread to carry out the requests so that it can handle other requests waiting in queue or wait for new requests. We expect such multithreaded implementation will allow controller to be capable of serving multiple requests simultaneously with improved response time and throughput.

The request issued by an administrator may be mainly classified into two categories: (1) configuring the system (e.g., add/remove a node into/from the system), or (2) performing a management function (e.g., the web site manager intend to delete or add a file) on all nodes. For condition (1), the working thread will invoke the administrative functions described in the control interface. These administrative functions perform in the form of native method. The native method is a mechanism in Java, which is used to call function that is written in languages other than Java. The administrative function will look up the kernel and setup the related data via the corresponding system call, which is defined in the control interface. In the condition (2), the controller will dispatch the corresponding agent to each server node for performing delegated task. The working thread will keep on listening for the execution result of the agent and report it to the administrator.

Broker

The broker is also a standalone Java application program, which performs as a daemon process on each server node in the system. The broker is composed of two operating threads: agent thread and monitor thread.

Agent Thread

The agent thread is responsible for accepting the dispatched agent from controller, and executing it for performing the delegated task. As a result, it should be capable of loading code from a variety of resources and provide an environment for executing it. For addressing this, we implemented the following components:

- **Class Loader.** To load code from other sources, the Java runtime system calls a subclass of the ClassLoader (an abstract class in Java), which defines an interface for the runtime system to ask a Java program to provide a class [12]. As a result, we implemented a specialized version of the ClassLoader as the skeleton of the agent thread, which can load Java class files from a variety of resources and convert the raw data of a class into an internal data structure representing that class.
- **Agent Context.** We defined and implemented an interface called agent Context, which is responsible for providing an environment in which the downloaded codes executes and interacts with broker.
- **Security Manager.** The capability of downloadable code is powerful, but it is also a potential security threat to a system and raises many concerns. The essence of the problem is that running programs on a computer typically needs to access certain resources on the host. However, if downloaded code is not careful to restrict the access of some critical system resources, it can also provide a malicious code with the same ability to do mischief on the host. In Java, the SecurityManager class is meant to define an interface for access control [13,14]. The SecurityManager class itself is not intended to be used directly, instead it is intended to be subclassed and installed as the system security manager. The Java platform is designed in such a way that all system calls made by a Java program must be routed through a security manager, which can decide whether or not certain sensitive operations should be allowed. Consequently, we defined a security policy and implemented a specific security manager to support runtime security on host environment. For example, the downloaded code can only access the given directory and file on the local file system.

Monitor Thread

The primary role of the monitor thread is to monitor the status of the managed node. Periodically it will wake up and initiate a request to web server running on the managed node. For minimizing the additional workload added on the managed node, such a request is designed for retrieving a small file (e.g., Home page in our implementation)

If the server responds normally, then the broker will send a message to distributor. Such a message will be treated as "heartbeat" of this managed node. The distributor keeps a counter for each server node, and such a counter will be incremented periodically. When the distributor receives a heartbeat from one server node, it resets the counter associated with that server node. On selecting a server for a new arriving request, distributor checks the counter associated with the candidate server, which is selected by load balancing module. If the counter exceeds a "warning value," which means that the server node may be either unreachable or overburdened. Such a node will be skipped, and the request will automatically be allocated to the next most available server. If the counter exceeds a "dead value" (which is a higher threshold than warning value), the node will be declared dead and be removed from the server cluster. The administration system will automatically record such an event in a log file and inform administrator by E-mail. As a result, any failure in the clustered server can be masked by such a mechanism, and the user from the outside world will be unaware of it.

In addition, the monitor thread also measures the response time that the request took from start to finish and then performs the following algorithm:

```
If (RPT_NOW > RPT_LAST) then
  Interval_time = RPT_NOW * Multiplier;
 Else
    Interval_time = RPT_LAST * Multiplier;
RPT_LAST = RPT_NOW;
Sleep(Interval_time);
```

RTP_NOW denote the response time of the request issued by this "wake-up." We also keep the response time of request issued by the last "wake-up" in RTP_Last. Multiplier is a pre-assigned value for calculating the Interval_time. The Interval_time is the interval from this "wake-up" to next "wake-up." There are two main purposes in such a design. First, it will prevent the monitor thread from burdening the load of web server. If the server is overloaded, the monitor thread will decrease the frequency of probe by discovering the longer response time. Second, fine-grained load balancing can be achieved by such a design. That is, if the managed server node is overloaded, the time of interval will be lengthened. This means that the broker will increase the time of interval between this heartbeat and the next. As a result, the distributor will stop to dispatch new request to this node, because it does not receive the heartbeat for a long time.

Agent

The primary role of agent is to perform one delegated task on all nodes according to the request of administrator. An agent is dispatched by controller and executes within the environment created by broker. The agent is in the form of Java classes and is transported across the network as byte streams. It will be reconstituted into Class objects by class loader and runs in its own thread of execution after arriving at a host.

We first defined an Agent class (a subclass of Object) to be the abstract base class, and then we implemented (inherit the Agent class) all management functions in the form of agent. For instance, we implement an agent to visit all server nodes for updating (or deleting) a file.

In addition, we built in a priority mechanism in the agent system. Based on this, we can assign different priority to different agent. For example, we implemented an agent to analyze the log file of each server node for security concern. Such an agent needs to be executed continually in the system, but it does not need to be executed immediately. Consequently, we can assign a lower priority to this agent. When controller receives a request for executing such agent, it will not dispatch the agent to a server node until that node is idle. Such a design will prevent those background jobs (i.e., housekeeping job) from burdening the load of server node during the high-traffic periods.

Remote Console

For simplifying the system administration as much as possible, we implement a management tool called remote console from which an administrator can issue management operations. The remote console is a Graphical User Interface (GUI) in the form of Java applet, thus it can run under any Java-enabled browser environment. The remote console applet is built completely on top of Java abstract window toolkit (AWT). It will interact with the user and communicate with the controller over the network. At any given time, the Web site manager can download the remote console applet anywhere after proper authentication and then control or monitor the whole system. The applet has a main thread for listening request from the user. Upon receiving a request, it will create a new working thread, and then the main thread is released and ready for other user request. The working thread will talk to the controller for carrying out the requests. We implemented a prototype remote console, which provides the following functions:

- **Configuration function** (e.g., join/remove a server node, or schedules a down time for maintenance): When one select such function, the virtual console will inform the controller, which will invoke the administration function described above to accomplish the configuration setup.
- **Content management** (e.g., add/delete or modify a file): When one selects such a function and fills in the related variables, the remote console will inform the controller, which will dispatch the respective agent to each server node for performing the job.
- **Monitor the activities of the system** : One can select the monitor function to monitor the activities of each node. The controller will invoke

the "read related data" function defined in the control interface to report the related data. The administrator also can assign some special event auditing, and then the controller will dispatch an agent to each node for analyzing its log file.

These user selectable functions are available in the form of buttons at the bottom of the applet's main window. One can see the remote console and its demo operation from our web site [15].

Discussion

In this section, we discuss some related issues raised in our system and compare our system with other related works.

Security

The security of our system depends fundamentally on the following three layers: the Java language itself, class loader, and the security manager. First, the Java language has the following important features from a security standpoint: access control for variables and methods within classes, lack of pointers as a language data type, and automatic garbage collection. Second, the class loader performs further security check to verify that the downloaded code does not violate the security requirements of the Java language. Finally, the security manager supports runtime security on host environment.

Recently, the security manager has been augmented with fine-grained access control mechanisms that allow it to make decisions based on who signed the downloaded code and where it was loaded from [16]. In the future, we will use cryptographic techniques (e.g., digital signature) to guarantee the integrity of code transferred and to provide an identification of the agent provider.

Load balancing

Given a clustered server, poor performance may still exist, which is often due to uneven load distribution throughout the system. As a result, a good load balancing mechanism is required for allocating incoming requests in a way that utilizes the cluster resource evenly and efficiently. At first, we [4,5] used a round robin method as the load-balancing policy for the reason of minimizing the cost associated with load balancing mechanism. This simple scheduling mechanism suffers from the fact that the processing time of individual request is not constant For instance, the load imbalance still happens while one server node receives five requests for 3KB HTML files, another same server node receives five requests for 3MB MPEG files. Consequently, we implemented an alternative scheduling mechanism to solve this problem. The new mechanism keeps track of the number of outstanding requests in each server node, and distributes the request to the node with the least number of outstanding requests. However, the both two mechanisms do not always reflect the actual load because they do not track the actual load condition on the server nodes. A potentially better approach is to combine the two mechanisms with some additional information about the actual load. With the administration system, such a multi-level load-balancing can be achieved by monitor thread of broker. The distributor can dispatch the incoming requests to one of the server nodes, using the previous scheduling mechanism, unless the periodic heartbeat of some of these nodes stops. The periodic heartbeat is stopped when the monitor thread detects the fact that the load of the managed server exceeds a critical threshold. In such a case, the distributor temporarily excludes the overloaded servers from scheduling consideration until it receives the heartbeat again (when the load returns under the given threshold).

Comparison

The distributed server concept has been used by many research projects to address the scalability and availability problems of Internet server. In this section, we compare these works with our system (i.e., the distributed server coupled with Java-based administration system).

The NCSA scalable HTTP server is described in [17,18]. Their architecture consists of a number of server machines cooperating to provide the Web service, that use the Andrew file system for sharing content provided by the Web site, and using the round robin DNS server for distributing accesses. In contrast to our scheme, this architecture has the following problems. First, the round robin DNS approach will inevitably suffer from the problems of DNS caching effect [4,5]. In comparison, the load balancing achieved using our routing TCP connection approach is significantly better than that achieved using the round robin DNS techniques. Second, it simply distributes incoming requests in a round robin fashion, which does not consider the heterogeneity of each request and existing load of respective machine. Furthermore, if the configurations (e.g., CPU type, memory size, etc) of machines in the clustered server are different, it is not considered in making a mapping. Third, this architecture did not consider the problem of failure detection and handling.

The Magic Router [19] provides transparent access by placing a modified router before a set of machines, which cooperate to provide a service. Their approach is very similar to our distributor, which redirects incoming Web requests via rerouting TCP connection. In the modified router, they use a user level process to intercept all IP packets and possibly modify packets destined for WWW service for directing it to a selected host. It allocates requests using round robin, random, or incremental system load methods. The main problem of this scheme is that using user space approach will pay the performance cost of context switching delay, and that multiple competing processes will increase the scheduling delay by over an order of magnitude. Furthermore, it does not consider the content sharing and management problem.

IBM proposed a prototype scalable and highly web server [20] built on an IBM SP-2 [21] system. The system architecture consists of a set of logical front-end nodes and a set of back-end nodes. The front-end nodes run the web daemons and are connected to the external network. The back-end node function as the server node for the sharing file system, used by the frond-end to access the data [22]. They use a TCP router approach to dispatch incoming requests. This scheme requires dedicated hardware (i.e., SP-2 system) and software. Consequently, it may not be feasible for all Web sites. In contrast, our approach can be built from commodity hardware and software components. It also can be applied to any existing web site in a manner that any kind of software/hardware of the original server does not need to be replaced.

In addition, several products [23,24,25] have been announced for use as front-end nodes that perform distributing functions across a group of servers. Due to space limitations, we do not describe the details of all these products.

All these works do not address the system administration problem. Many administration or content management operations must be done manually on each node. It should be the nightmare of web site manager, in particular, a number of web site cannot afford specialized computer operations staff. In contrast, we think our system has advantages over other architectures in terms of scalability, high availability, fine-grained load balancing, and powerful and sophisticated system administration functions.

Conclusion

In this paper, we demonstrate the design and implementation of an administration system for distributed server. We exploit the advantages of Java to construct this administration system, which provides the solution for the automation of many tasks in system administration and content management that usually must be done manually. We also offer an easy-to-use GUI for web site manager to maintain and manage the system. With the proposed system, the administrator can perform functions on all nodes at once, monitor the activities of Web site, and manage the entire system as easy as facing a single host. Furthermore, with the feature of downloaded agent, the proposed system provides unlimited possibility to extend its function.

In addition, the proposed system enables multi-level control of incoming requests. The combination of load-balancing mechanism provided by distributor and fine-grained load balancing archived by the administration system gives more precise control for directing incoming requests.

The system also provides excellent high availability features, including automatically detection of failures, and alerts in the form of log file and e-mail. The end user will be unaware that any failure has occurred on the server, although the aggregate capacity of the server will temporarily be reduced. In addition, our approach can enable a high performance server to be built from commodity hardware and software components. Otherwise, it also can be applied to any existing web site in a manner that any kind of software/hardware of the original server does not need to be replaced.

In the future, we will further investigate the security issues raised by Java language and our system in detail.

Acknowledgements

This work was supported by the National Science Council, R.O.C., under contract no. NSC 86-2213-E-110-026 and NSC 86-2213-E-110-032.

The authors would like to thank David D. H. Lin of IBM for his valuable suggestions and help on this work. The authors also greatly appreciate Gretchen Phillips for her proof reading and comments on this paper.

Author Information

C. S. Yang received the B.S. degree in engineering science and the M.S. and Ph.D. degrees in electrical engineering from National Cheng Kung University, Tainan, Taiwan, Republic of China, in 1976, 1984, and 1987, respectively. During 1988-1992, he was with the Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, Republic of China. In 1992, he joined the faculty of the Institute of Computer and Information Engineering at National Sun Yat-Sen University, where he is a professor and Chairman of the Institute now. His current research interests include mobile computing, parallel/distributed programming support environment, and scalable architecture. Reach him at csyang@cie.nsysu.edu.tw .

M. Y. Luo received the B.S. degree in Physics from the National Sun Yat-Sen University, Kaohsiung, Taiwan, Republic of China, in 1995, and the M.S. degree in Computer Science from the Institute of Computer and Information Engineering at National Sun Yat-Sen University in 1997. He has been working toward his Ph.D. degree in the Institute of Computer and Information Engineering at National Sun Yat-Sen University. His research interests are in the areas of computer network, Internet technology, and parallel and distributed system. Reach him at myluo@cie.nsysu.edu.tw .

References

[1] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Nielsen, A. Secret. "The World-Wide Web" *Communications of the ACM*, Aug. 1994.

[2] *Java-Programming for the Internet*. http://java.sun.com .

[3] C. S. Yang, M. Y. Luo, "Design an Environment for Scalable Web Server," *Proceedings of 1996 Multimedia Technology and Applications Workshop*, pp. 107-114. Dec. 1996.

[4] M. Y. Luo, *Design and Implementation of a Scalable and Highly Available Web Server.* M.Sc. Thesis, Institute of Computer and Information Engineering, National Sun Yat-Sen University, June 1997.

[5] 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.

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

[7] G. Wright and W. R. Stevens *TCP/IP Illustrated, Volume2*, Addison-Wesley, Reading, May 1995.

[8] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0*, http://www.w3.org/hypertext/WWW/Protocols/ .

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

[10] J. Rodely, *Writing Java Applets*, The Coriolis Group, Inc.

[11] J. Gosling and H. McGilton. *The Java Language Environment, A White Paper*, May 1996. Available via ftp://ftp.javasoft.com/docs/papers/langenviron-ps.zip .

[12] T. Lindholm and F. Yellin, "The Java Virtual Machine Specification." Addison-Wesley, 1996.

[13] Joseph A. Bank, "Java Security," Available via http://swiss-ftp.ai.mit.edu/~jbank/javapaper.ps

[14] F. Yellin. "Low level security in Java." In *Fourth International World Wide Web Conference*, Boston, MA, Dec. 1995. Available via http://www.w3.org/pub/Conferences/WWW4/Papers/197/40.html .

[15] http://pds1.cie.nsysu.edu.tw/WebScale/Demo/console.html .

[16] Li Gong and Roland Schemers. "Implementing protection domains in the Java Development Kit 1.2." In *The Internet Society Symposium on Network and Distributed System Security*, San Diego, California, March 1998.

[17] E. D. Katz, M. Butler, and R. Mcgrath. "A Scalable HTTP Server: The NCSA Prototype," *Proceedings of First International WWW Conference*. May 1994.

[18] R. McGrath T. Kwan and D.Reed. "NCSA's World Wide Web Server: Design and Performance." *IEEE Computer*, November 1995.

[19] 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 .

[20] D. Dias, W. Kish, R. Mukherjee, and R. Tewari, " A Scalable and Highly Available Web Server," *COMPCON* 1996, pp.85-92, 1996.

[21] T. Agerwala, J. Martin, J. Mirza, D. Sadler and M. Snir, "Sp2 system Architecture," *IBM System Journal*, Vol. 34, no. 2, pp. 152-184, 1995.

[22] C. R. Attanasio, M. Butrico, C. A. Polyzois, S. E. Smith, and J. L. Peterson. "Design and Implementation of a Recoverable Virtual Shared Disk." *IBM Research report RC 19843*, T.J Watson Research Center, Yorktown Heights, New York, 1994.

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

[24] "Cisco LocalDirector," http://www.cisco.com/warp/public/751/lodir/index.html .

[25] Cisco *System. Scaling the Internet Web Servers: A white paper*. http://www.cisco.com/warp/public/751/lodir/scale_wp.htm, 1997.