Dispatcher Based Dynamic Load Balancing on Web Server System

Harikesh Singh, Dr. Shishir Kumar

Department of Computer Science & Engineering, Jaypee University of Engineering & Technology, Guna (MP), India
harikeshsingh@yahoo.co.in, shishir_ks@yahoo.com

Abstract

Use of Internet and frequent accesses of large amount of multimedia data are fast increasing the traffic of network. Performance evaluation and high availability of server are important factor to resolve this problem using various cluster based systems. There are several low-cost servers using the load sharing cluster system which are connected to high speed network, and apply load balancing technique between servers. It offers high computing power and high availability. The overall increase in traffic on the World Wide Web is augmenting user-perceived response times from popular Web sites, especially in conjunction with special events. A distributed website server can provide scalability and flexibility to manage with growing client demands. To improve the response time of the web server, the evident approach is to have multiple servers. Efficiency of a replicated web server system will depend on the way of distributed incoming requests among these replicas. A distributed Web-server architectures schedule client requests among the multiple server nodes in a user-transparent way that affects the scalability and availability. The aim of this paper is the development of a load balancing techniques on distributed Web-server systems.

Keywords: *distributed system, cluster system, load balancing, distributed Web server system, dispatcher.*

1. Introduction

In a distributed system, many clients act as clients to computers known as servers. To implement a system in a distributed approach, firstly ensure that processing power is as close to the users as possible and the second is to ensure a high degree of robustness, for example via the use of data replication; and the third is to enable hardware to be easily added as the resource demands of the applications running on the distributed system start increasing. Many problems have been facing the designer of a distributed system including predicting the performance of a particular design, keeping all the clocks in the system synchronized, and ensuring that if a hardware element of the system malfunctions, users are at best, only affected in a minimal way.

Internet usage has meant that many distributed systems are open to the world that has been given rise to a major problem: ensuring that such systems are secure. The best-known distributed system is the moderately simple World Wide Web which consists of a very large number of clients running browsers and a large number of Web Servers. An important core of the title lies in Web Server. A Web Server is the computer program that is responsible for accepting HTTP requests from the Clients and serving them HTTP responses along with optional data contents which usually are web pages such as HTML docs and linked objects etc. A Web Server is the one which hosts or contains all the important resources in the internet and deals with the requests from others (clients) for those resources and provides

them those resources with various conditions. A Web Server has both hardware and software requirements.

A Hardware requirements includes a high configured hardware are needed with the high processor speed and high connectivity. In software requirements, there is a need of special software known as Web Server software. There is much Web Server software available today for e.g. Apache Tomcat Server, Microsoft Internet Information Service (Microsoft IIS), Sun java Web Server etc. Any single computer system having all the Internet services for a small group of company would include the HTTP server (Web pages and files), FTP server (file downloads), NNTP server (newsgroups) and SMTP server (mail service) also known as a Web server system. Each of these services has integrated in a separate computer or in multiple computers in ISPs or large companies. A data centre for a large public Web site could contain hundreds and thousands of Web servers.

Web Servers System is a group of Web Servers running a web application simultaneously, and at the same time appearing to the user world as if it is a single server. It is one of the biggest applications of the Distributed System. In the today's popular world of Web where users are increasing at the lightening fast rate, Web servers system is very useful to provide higher availability, easier manageability, and greater scalability. Load Balancing comprises of two words load and balance. Load is something that requires effort and balance is to compare and equate. Thus Load Balancing is simply an activity or process which is used to distribute the load which is constantly applied to single point to the different points. This technique is useful in many fields like in mobile communication, LAN Servers, WAN Servers, and Web Servers etc. It is required in websites, Inter Relay Chat, high bandwidth FTP servers, NNTP and DNS servers In computer networks, it is a technique to spread work between two or more computers, links, CPU's, hard drives or other resources in order to get optimal resources utilization, maximizing throughput, and minimizing response time.

Load Balancing can be achieved either by static approach or by dynamic approach. Static approach is the one in which load is known in advance and is constant while dynamic approach is the one in which load is not known in advance and may vary from time to time. Moreover in dynamic approach the load is distributed at runtime depending on different criteria while in static approach load is distributed equally. Drawback of dynamic load balancing is that it is costlier than static one. Due this factor both static and dynamic approaches have their own areas of implementations.

2. Related Works

2.1 Client-based Approach

Document requests to popular Web sites can be routed from the client side in any replicated web server architecture even when the nodes are loosely (or not) coordinated. Routing to the Web cluster can be provided by Web clients or by client-side proxy servers. These approaches are not universally applicable [1]. The way of accessing the Netscape Communication site through the Netscape Navigator browser is the first example of a client based solution [8].

2.2 Web Clients

Web clients, if they are aware of the Web-server system's replicated servers, can actively route requests. After receiving a request, the Web client selects a node of the cluster and, after resolving the address mapping, submits the request to the selected node, which is then

responsible for responding to the client. The same client can send another request and reach another server. Netscape Communications' approach is one example of Web-client scheduling. The load-balancing mechanism for the Netscape Web-server system's multiple nodes is as follows.

When a user accesses the Netscape home page (http://www.netscape.com), Navigator selects a random number i between 1 and the number of servers and directs the user request to the node wwwi.netscape.com. This approach, which has very limited practical applicability and is not scalable, might offer utility to corporate intranets. A Second example of Web-client scheduling is via smart clients. Unlike the traditional approach that does not involve the Web client, this solution migrate server functionality to the client through a Java applet. The increased network traffic due to the continued message exchanges among each applet and server node to monitor node states and network delays, however, is a major drawback. Moreover, although this solution provides scalability and availability, it lacks client-side portability [1].

2.3 Client-side Proxies

From the Web cluster point of view, proxy servers are similar to clients. The proxy server is an important Internet entity that can route client requests to Web-server nodes. Like all Web-client approaches, this one has limited applicability. We do not further investigate proxy servers because any load balancing mechanism they carry out requires Internet component modification. Typically, the same institution or company that manages the distributed Web-server system does not control the modification.

2.4 DNS-based Approach

Distributed Web-server architectures that use request routing mechanisms on the cluster side are free of the problems of client-based approaches. Architecture transparency is typically obtained through a single virtual interface to the outside world, at least at the URL level. (Other approaches provide a single virtual interface even at the IP level, as we will explain). The cluster DNS—the authoritative DNS server for the distributed Web system's nodes—translates the symbolic site name (URL) to the IP address of one server [13].

This process allows the cluster DNS to implement many policies to select the appropriate server and spread client requests. The DNS, however, has a limited control on the request reaching the Web cluster. Between the client and the cluster DNS, many intermediate name servers can cache the logical-name-to-IP address mapping to reduce network traffic. Moreover, every Web client browser typically caches some address resolution. Besides providing a node's IP address, the DNS also specifies a validity period (Time-To-Live, or TTL) for caching the result of the logical name resolution. When the TTL expires, the address-mapping request is forwarded to the cluster DNS for assignment to a Web-server node; otherwise, an intermediate name server handles the request—Figure 1 shows both resolutions. If an intermediate name server holds a valid mapping for the cluster URL, it resolves the address-mapping request without forwarding it to another name server. Otherwise, the address request reaches the cluster DNS, which selects the IP, address of a Web server and the TTL. The URL-to-IP-address mapping and the TTL value are forwarded to all intermediate name servers along the path and to the client. Several factors limit the DNS control on address caching. First, the TTL period does not work on the browser caching. Moreover, the DNS might be unable to reduce the TTL to values close to zero because of non cooperative intermediate name servers that ignore very small TTL periods. On the other hand, the limited control on client requests prevents the DNS from becoming a potential bottleneck.

We distinguish the DNS-based architectures by the scheduling algorithm that the cluster DNS uses to balance the Web-server nodes' load. With constant TTL algorithms, the DNS selects servers on the basis of system state information and assigns the same TTL value to all address-mapping requests. Alternatively, adaptive TTL algorithms adapt the TTL values on the basis of dynamic information from servers and/or clients [1][2][3].

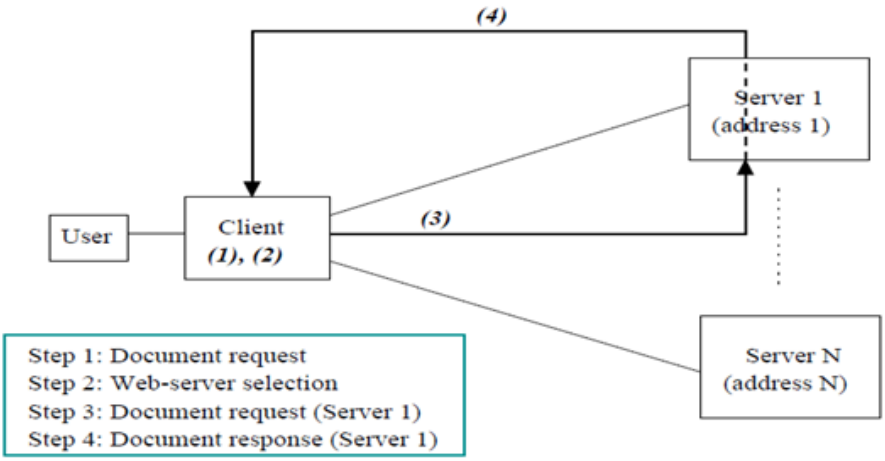

Figure 1: Web Clients Based Approach

2.5 Constant TTL Algorithms

These algorithms are classified by the system state information that the DNS uses to select a web server node -none, client load or client location, server load, or a combination [1].

2.6 System-stateless Algorithms

The Round Robin DNS (RRDNS) approach, first implemented by the National Centre for Supercomputing Applications (NCSA) to handle increased traffic at its site, is for a distributed homogeneous Web-server architecture.4 NCSA developed a Web cluster comprising the following entities: a group of loosely coupled Web-server nodes to respond to HTTP requests; a distributed file system that manages the entire WWW document tree; and one primary DNS for the entire Web-server system. With an overloaded or nonoperational server, no mechanism can stop the clients from continuing to try to access the Web site by its cached address. The RR-DNS policy's poor performance needs research into alternative DNS routing schemes that require additional system information.

2.7 Server-state-based Algorithms

Knowledge of server state conditions is essential for a highly available Web-server system to exclude servers that are unreachable because of faults or congestion. DNS policies, combined with a simple feedback alarm mechanism from highly utilized servers, effectively avoid Web-server system overload [19][20]. The DNS, after receiving an address request, selects the least-loaded server. To inhibit address caching at name servers, the lbm named algorithm requires that the DNS sets the TTL value to zero.

2.8 Client-state-based Algorithms

Two kinds of information can come from the client side: the typical load that arrives at the Web-server system from each connected domain and the client's geographical location. The hidden load weight index measures the average number of data requests sent from each connected domain to a Web site during the TTL caching period following an address-mapping request. (A normalized hidden load weight represents the domain request rate.) Proposed DNS scheduling policies, which chiefly use this information to assign requests to the most appropriate server, try to identify the requesting domain and the hidden load weight imposed by this domain. One example of this algorithm is the multitier round-robin policy, which uses different round-robin chains for requests in domains of different hidden load weights [10][11][12].

The Distributed Director evaluates the approximate client location using the IP address of the client's local DNS server. A third alternative is the Internet2 Distributed Storage Infrastructure system (I2-DSI), which proposes a smart DNS that implements address resolution criteria on the basis of network proximity information, such as round-trip delays. These geographic DNS-based algorithms do not work well if URL-to-IP-address mapping is always cached by the intermediate name servers. To make them work, the cluster DNS sets the TTL to zero. However, this solution is limited by non cooperative name servers [1][2].

2.9 Server and client-state-based Algorithms

DNS algorithms are most effective when they use both client and server state conditions. For example, the Distributed Director DNS algorithm uses server availability information along with client proximity. Similarly, the hidden load weight may be insufficient to predict the load conditions at each Web server node. An asynchronous alarm feedback from over utilized servers lets newer DNS policies exclude those servers from request assignments during overload conditions and instead use the client-state-based algorithms to select an eligible server from the non overloaded servers. To acquire the system state information needed by the enhanced DNS scheduling algorithms requires efficient state estimators, and several dynamic approaches have been suggested.

2.10 Adaptive TTL Algorithms

To balance the load across multiple Web-server nodes, the DNS can exert control through both the policy for server scheduling and the algorithm for selecting the TTL value. Constant TTL algorithms cannot adequately address client request skew and probable heterogeneity of server capacities, so we have devised dynamic (adaptive) TTL scheduling policies. The algorithms use some server- and client-state-based DNS policy to select the server, and dynamically adjust the TTL value. Therefore, by properly selecting the TTL value for each address request, the DNS can control the subsequent requests to reduce the load skews that primarily cause overloading.

Adaptive TTL algorithms use a two-step decision process. First, the DNS selects the Web-server node similarly to the hidden load weight algorithms. Second, the DNS chooses the appropriate value for the TTL period. Adaptive TTL algorithms can easily scale from LANs to WANs because they require only information that can be dynamically gathered by the DNS; namely, the request rate associated with each connected domain and the capacity of each server [4].

2.11 DNS-based Architecture Comparison

DNS policies based on detailed server state information (for example, present and past load) do not effectively balance client requests across servers. The policies are ineffective because with address caching, each address mapping can cause a burst of future requests to the selected server and quickly obsolete the current load information. The domain request rate estimates the impact of each address mapping and is more useful to guide routing decisions. Scheduling algorithms based on the domain request rate and alarms from overloaded servers can lead to better load balancing than RR-DNS and maintain high Web site availability. However, they give less satisfactory results when generalized to a heterogeneous Web-server system through probabilistic routing. To balance requests among distributed Web server systems, adaptive TTL algorithms are the most robust and effective, despite skewed loads and non cooperative name servers. The algorithms, however, do not consider the client-to-server distance in making scheduling decisions. Furthermore, such policies do not consider the client level address caching, resulting in subsequent requests from the same client (browser) being sent to the same server as shown in figure 2.

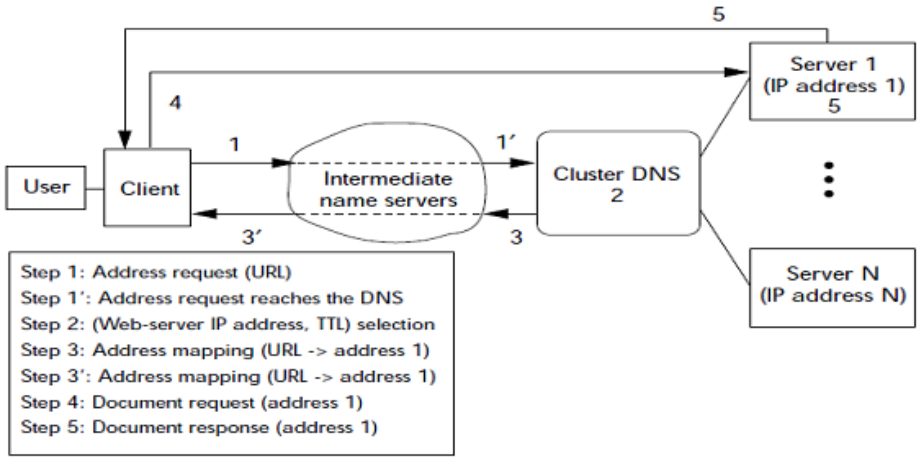

Figure 2: DNS-based Approach to Load Balancing

Problems exist even at the network-level of address caching because most intermediate name servers are configured such that they reject very low TTL values [1][2][12].

2.12 Dispatcher-based Approach

To centralize request scheduling and completely control client-request routing, a network component of the Web-server system acts as a dispatcher. Request routing among servers is transparent—unlike DNS-based architectures, which deal with addresses at the URL level, the dispatcher has a single, virtual IP address (IP-SVA). The dispatcher uniquely identifies each server in the system through a private address that can be at different protocol levels, depending on the architecture. We differentiate dispatcher-based architectures by routing mechanism—packet rewriting (single-rewriting or double-rewriting), packet forwarding, or HTTP redirection. Dispatcher-based architectures typically use simple algorithms to select the Web server (for example, round-robin, server load) to manage incoming requests, as simple algorithms help minimize request processing [1][2][4][6].

2.13 Packet single-rewriting

In some architecture the dispatcher reroutes client to-server packets by rewriting their IP address, such as in the basic TCP router mechanism. The Web server cluster consists of a group of nodes and a TCP router that acts as an IP address dispatcher. Figure 3 outlines the mechanism, in which address i becomes the private IP address of the i-th Web server node. All HTTP client requests reach the TCP router because the IP-SVA is the only public address. The dispatcher selects a server for each HTTP request through a round-robin algorithm and achieves routing by rewriting each incoming packet's destination IP address. The dispatcher replaces its IPSVA with the selected server's IP address. Because a request consists of several IP packets, the TCP router tracks the source IP address for every established TCP connection in an address table. The TCP router can thereby always route packets regarding the same connection to the same Web server node.

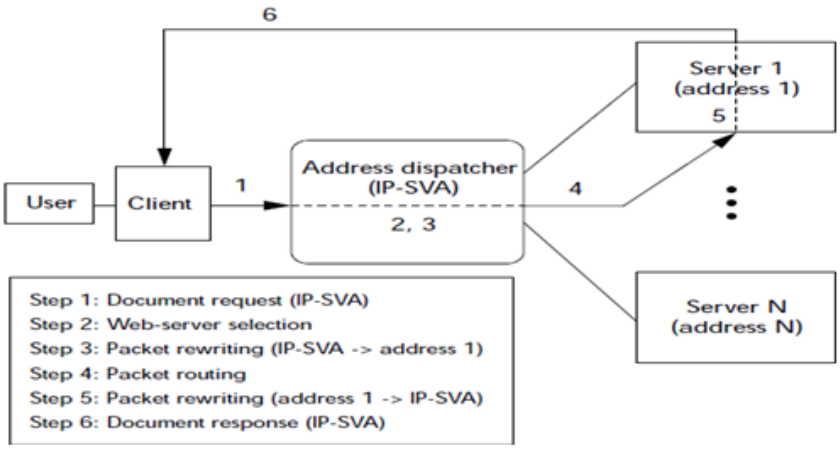

Figure 3: Packet single-rewriting by the Dispatcher

Furthermore, the Web server must replace its IP address with the dispatcher's IP-SVA before sending the response packets to the client. Therefore, the client is not aware that its requests are handled by a hidden Web server. This approach provides high system availability because, when a front-end node fails, its address can be removed from the router's table to prevent further request routing. Moreover, the TCP router architecture can be combined with a DNS-based solution to scale from a LAN- to a WAN-distributed Web system.

2.14 Packet double-rewriting

This mechanism also relies on a centralized dispatcher to schedule and control client requests but differs from packet single-rewriting in the source address modification of the server-to-client packets. Instead, the dispatcher modifies all IP addresses, including those in the response packets. Packet double-rewriting underlies the Internet Engineering Task Force's Network Address Translator, shown in Figure 4. The dispatcher receives a client request, selects the Web-server node and modifies each incoming packet's IP header, and also modifies the outgoing packets that compose the requested Web document. Two solutions using this approach (with a server fault-detection mechanism) are the Magicrouter and Cisco Systems' LocalDirector [14].

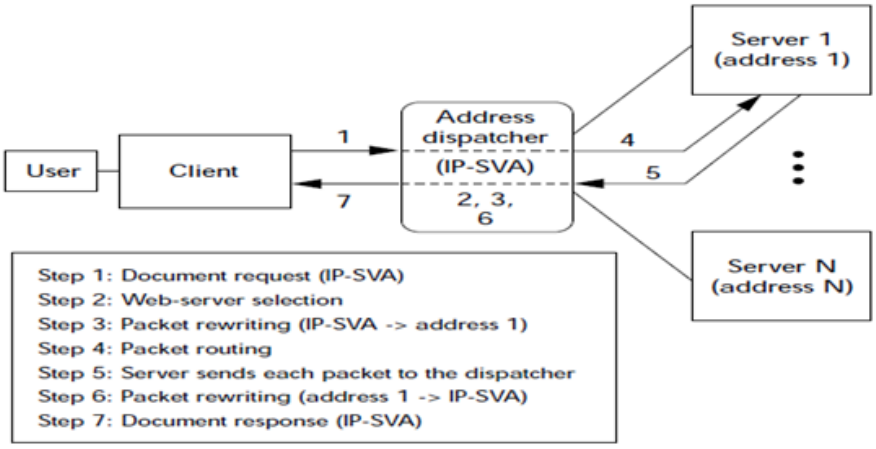

Figure 4: Packet double-rewriting by the Dispatcher

2.15 Packet Forwarding

Several other approaches may use the dispatcher to forward client packets to the servers as an alternative of rewriting their IP addresses.

2.16 Network Dispatcher

IBM's Network Dispatcher extends the basic TCP router mechanism. The Network Dispatcher works with both LANs. The LAN Network Dispatcher assumes that the dispatcher and the server nodes are on the same local network. All share an IP-SVA address; however, the client packets reach the dispatcher because the Web nodes have disabled the Address Resolution Protocol (ARP) mechanism. The dispatcher can forward these packets to the selected server using its physical (MAC) address on the LAN without IP header modification. Unlike the basic TCP router mechanism, neither the LAN Network Dispatcher nor its Web servers need modify the response packets' IP header. With this mechanism, similar to that shown in Figure 2, address i is the private hardware MAC address of the i-th Web-server node. This solution is both client- and server-transparent because it does not require packet rewriting. The dispatcher's scheduling policy can be dynamically based on server load and availability [1][2][21].

2.17 ONE-IP Address

Another forwarding approach uses the ifconfig alias option to configure a Web server system with multiple machines. This solution publicizes the Web-server system's IP-SVA as the same secondary IP address of all nodes and can be implemented with two techniques. Routing-based dispatching requires that all packets directed to the ONE-IP address are first rerouted by the sub network router to the IP address dispatcher of the distributed Web-server system. The dispatcher selects the destination server based on a hash function that maps the client IP address into a server's primary IP address, then reroutes the packet to the selected server. This approach provides a static assignment; however, it guarantees that all packets belonging to the same TCP connection are directed to the same server. Broadcast-based

dispatching requires that the sub network router broadcast the packets having ONE-IP as a destination address to every server in the Web system [9].

2.18 HTTP Redirection

A centralized dispatcher receives all incoming requests and distributes them among the web server nodes through the HTTP's redirection mechanism. The dispatcher redirects a request by specifying the appropriate status code in the response, indicating in its header the server address where the client can get the desired document. Such redirection is largely transparent; at most, users might notice an increased response time. Unlike most dispatcher-based solutions, HTTP redirection does not require IP address modification of packets reaching or leaving the Web-server system. HTTP redirection can be implemented with two techniques. Server-state-based dispatching, used by the Distributed Server Groups architecture, adds new methods to HTTP protocol to administer the Web system and exchange messages between the dispatcher and the servers. Since the dispatcher must be aware of the server load, each server periodically reports the number of processes in its run queue and the number of received requests per second as shown in figure 5 [23].

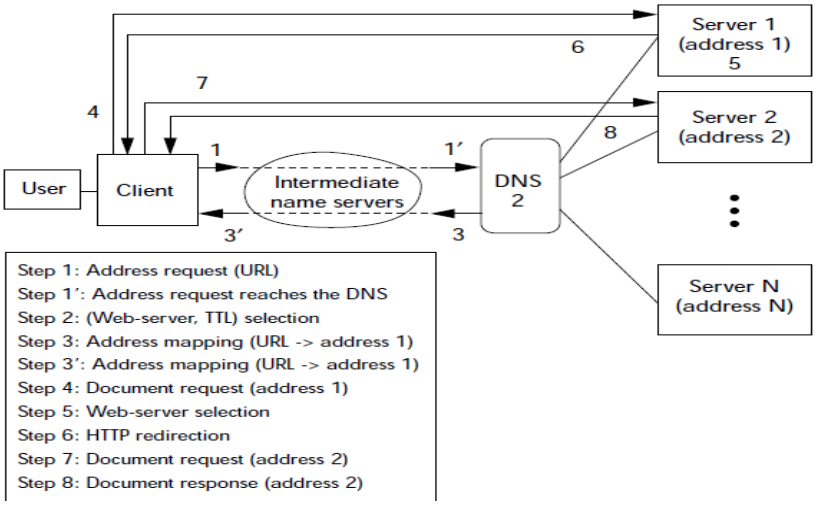

Figure 5 : HTTP redirection by the server

2.19 Dispatcher-based Architecture Comparison

Packet double-rewriting by the dispatcher presents problems because the dispatcher must rewrite incoming as well as outgoing packets, and outgoing packets typically outnumber incoming request packets. Packet single-rewriting, which the TCP router architecture uses, sustains the same overhead of rewriting in both directions but it reduces dispatcher bottlenecks because the Web servers rewrite the more numerous servers-to client packets. The ONE-IP approach, however, can have problems with the static scheduling algorithm, which does not consider the server state for routing decisions. While routing-based dispatching requires double rerouting of each packet, broadcast-based dispatching broadcasts all packets to every server, thus causing even higher server overhead. The LAN Network Dispatcher architecture avoids most of ONE-IP's traffic problems, and TCP router and double-rewriting overheads, but it lacks geographical scalability as it requires the same network segment to

connect the dispatcher and the Web server nodes. HTTP redirection can be finely applied to LANs but it duplicates the number of necessary TCP connection [1][2].

2.20 Server-based Approach

A request is served or redirected depending on several factors. Redirection mechanisms are synchronously or asynchronously activated, and redirected entities can be individual clients or entire domains. Asynchronous activation occurs when the DNS-selected server determines that another server would better answer the client request—perhaps one server is overloaded, or another server is closer to the client domain. Redirecting individual client connections is crucial to better load balancing at a fine granularity level [15][16][17][18].

Each Web server redirects requests according to server selection that minimizes the client request's response time, a value estimated on the basis of server processing capabilities and Internet bandwidth/delay. These mechanisms imply an overhead of intra cluster communications, as every server must periodically transmit status information to the cluster DNS or other servers, but such cost only negligibly affects client-request-generated network traffic. To users, HTTP redirection's main drawback is increased response time, since each redirected request requires a new client-server connection [1][2][22].

2.22 Packet Redirection

Distributed Packet Rewriting (DPR) by the server uses a round-robin DNS mechanism to schedule the requests among the Web servers. The server reached by a request reroutes the connection to another server through a packet-rewriting mechanism that, unlike HTTP redirection, is transparent to the client. Two load-balancing algorithms spread client requests. The first uses static (stateless) routing, where a hash function applied to both the sender's IP address and the port number determines each packet's destination server. DPR can be applied to both LAN- and WAN-distributed Web-server systems, but the packet-rewriting and -redirecting mechanism causes a delay that can be significant in WAN-distributed Web-server systems [24][25][26][27].

Table 1: Comparative Study of Load Balancing Approaches

Approach	Scheduling	Pros	Cons
Client-based	Client side	No server overhead	Limited applicability
	Distributed	LAN and WAN solution	Medium-coarse-grained balancing
DNS-based	Web system side	No bottleneck	Partial control
	Centralized	LAN and WAN solution	Coarse-grained balancing
Dispatcher-based	Web system side	Distributed control	Latency time increase(HTTP)
	Centralized	Full control	LAN solution packet rewriting overhead
Server-based	Web system side	Distributed control	Latency time increase (HTTP)
	Distributed	Fine-grained balancing, LAN and WAN solution	Packet rewriting overhead (DPR)

3. Proposed Architecture

The models and various other aspects specific to architectural design and other information that we are going to implement for the development of software model where the incoming load from the client can be distributed efficiently over the web server system. The HTTP request coming from the client would be distributed by following the DNS Dispatching method using the Round Robin and proximity based Scheduling algorithm and output would be load balanced servers. The initial architecture would use the open source software such as BIND for DNS, JMeter for generating processes for client request, coding for the Dispatcher and Apache Tomcat Web Server for simulation. The software development model used to remove the defects as they come and enhance the system as much as possible [28][29][30][31].

3.1 Architectural Design of the Proposed Model

On the basis of several architectural design of various models and knowing there pros and cons, we have tried to design a model that should fulfill our requirements as shown in figure 6. We have designed an architecture that consists of following components:

- A DNS Server, Dispatcher Selector,
- N- Dispatcher (corresponding to N Clusters i.e. one for each cluster),
- Load Collector for each Dispatcher, Alarm Monitor for each Dispatcher,
- Load Checker for each Server, Request Counter for each Server.

A DNS server is one which is initially communicating with client for the request to convert its required URL into IP Address. Since every cluster has single Dispatcher, therefore Dispatcher Selector is directly connected to every Dispatcher. Each Dispatcher is directly connected to all the Web Servers in its cluster. Each Dispatcher consists of one Load Collector which collects the load of each server. Each Dispatcher also consists of one Alarm Monitor which checks the load on every server and temporarily stops the service of Web Server whose load becomes high. Whereas each Server consists of one Load Checker and one Request Counter which calculates and sends the information of load on server.

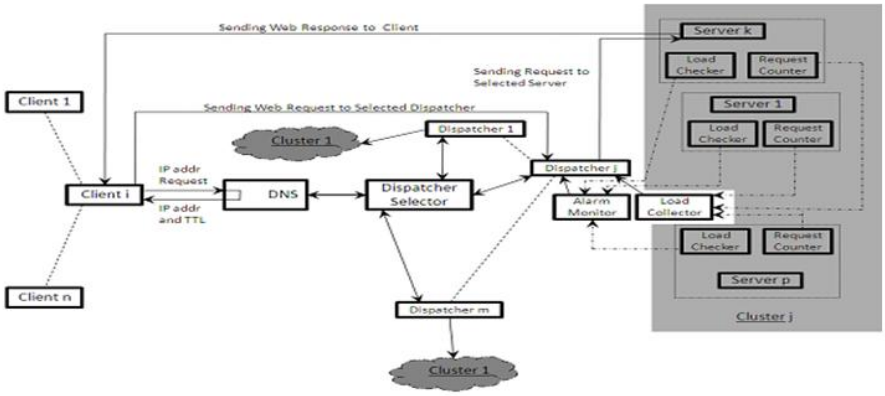

Figure 6 : Architectural design of dispatcher based dynamic load balancing model

4. Implementation and Testing

4.1 Modules Implemented

The algorithm of the proposed system is further alienated in various modules and later all of them were integrated into the whole architecture comprising of the dispatcher, server, and clients. The architecture was capable enough of deciding the best suited server for a particular client, on the basis of various chosen parameters like response time, CPU utilization, throughput and bandwidth. The modules are as follows:

- Establishing the architecture involving client, server and dispatcher.
- Calculation of response time of all the participating servers.
- Calculation of the CPU utilization of all the participating servers.
- Calculating the throughput of all the participating servers.
- Sending all the valid calculated data to the dispatcher which would further act as a decisive system and decide the best suited server for the requesting clients.

4.2 Concepts involved during implementation

CPU utilization is one of those performance factors that is both grossly underrated and overrated at the same time. Most people have never even heard of it; it often seems though that a big percentage of those who do understand its role worry about it way too much. Like most performance issues, sweating small differences in numbers is usually pointless; it doesn't matter much if your CPU utilization is 5% or 10%; but if it is 80% or 90% then you are going to see an impact on the usability of the system if you multitask. Another key issue is that faster drives transfer more data, and more data--all else being equal--requires more processing time. It's totally invalid to compare the CPU utilization of drives of different generations without correcting for this very important consideration. In this system we have used CPU as a factor for deciding the least loaded server. Following are the basic steps:

- Dispatcher is the decisive system that requests each of the servers for their respective CPU utilization.
- Each of the servers calculates their CPU utilization and sends them to the dispatcher.
- The whole process uses the concept of RMI i.e. Remote Method Invocation.
- The main formula used for this process is :

Response time measures the performance of an individual transaction or query. Response time is typically treated as the elapsed time from the moment that a user enters a command or activates a function until the time that the application indicates the command or function has completed. The response time for a typical Dynamic Server application includes the following sequence of actions. Each action requires a certain amount of time.

Throughput is essentially synonymous to digital bandwidth consumption. Since a client server interaction is not a single interaction since we are using Single packet rewriting approach a single server would be catering to the same client over a period of time and hence we need to calculate the throughput i.e. the number of successful requests send in a particular time period.

There are following steps involved in the throughput calculation:

- We have assigned an arbitrary time period to our dispatcher of 2 sec for the calculation of throughput.
- The dispatcher records the initial time and starts sending requests to the server and along with it computes the number of successful responses.
- At the end of 2 sec the number of successful responses is computed and this is our throughput.

The function and formula used for calculation of throughput are:

- Take a loop starting from 1 to 1000000
- Calls an arbitrary function on the server.
- Record the no. of successful connection established.
- Divide the no. of successful connections made by 1, to get the throughput.

Apache JMeter may be used to test performance both on static and dynamic resources (files, Servlets, Perl scripts, Java Objects, Data Bases and Queries, FTP Servers and more). It can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types. You can use it to make a graphical analysis of performance or to test your server/script/object behavior under heavy concurrent load [7].

4.3 Test Plan

To test the model, we have used a distributed testing approach of JMeter. In this approach, we have made one master system and several other slave systems. Then a test plan is designed on a master system in such a way that multiple threads (users) are generated on each slave on command given through master system. Each thread is capable of sending different types of requests (in our case it is HTTP request) on the system under test (in this case it is System of Web Servers). Thus creating load on the specific system under testing.

To test our we designed a test plan with one master system and three slave system and created total of 17000 threads (users) on the Web Server System to create load and evaluate the performance test of the System. The test plan which we have designed has following specification:

No. of master: 1

No. of slave: 3

No. of threads (user) generated: 17000

Loop count: 1

Server Name or IP: www.webservice.com

Path : website

4.3.1 Test Procedure and Results

To test the model with the above designed test plan, we had run the test plan under two different conditions and the results in various forms are collected. We have collected the result in following forms:

Throughput graph, % Error in responding from Web server, and Average response time of the System and its graph.

These results are finally compared with each other in this section. The two conditions under which the test is done and there results are as follows:

4.3.2 Test under Simple Condition (without load balancing)

In this condition, test is performed on a simple single Web Server with apache tomcat web server software installed on it and a simple HTML website is hosted on it. This is the simplest possible web server's architecture without load balancing implementation. Various results are as follows:

Table 2: Results of Testing in Simple Condition

Parameters	Value
No of request Served	14164
Average Response time	2400ms
% Error	21.45
Throughput	53.9/sec

4.3.3Test Under the Architecture Condition (with load balancing)

In this condition, test is performed on web server's system with our architecture with on cluster having two web servers with apache tomcat web server software installed on it and a simple HTML website is hosted on it. Various result of this test is as follows:

Table 3: Results of Testing in Proposed Architectural Condition

Parameters	Value
No of request Served	16581
Average Response time	512 ms
% Error	15.28
Throughput	58.1/sec

Table 4: Comparison of Two Test Conditions

Parameters	Value (without load balancing Condition)	Value (with load balancing Condition)
No of request Served	14164	16581
Average Response time	2400 ms	512 ms
% Error	21.45	15.28
Throughput	53.9/sec	58.1/sec

The test results of the two conditions shows clearly that there is improvement in response time, % Error and throughput after implementing the model is the web server system's when there is the large load on these system.

5. Conclusion and Future Work

Dynamic Load Balancing on Web Server System has been used to improve the availability and reduce the overloading of the Web Servers. After studying the various classical approaches such as the Client based approach, DNS based approach, Dispatcher based and Server based approach for load balancing we developed our own model which is an amalgamation of the DNS and Dispatcher based approach. We have taken into consideration a few parameters such as CPU utilization and network traffic to select our Web Server to serve the request. During the development of the design and architecture we studied the various drawbacks of the existing models and tried to improve upon them and finally developed the load balancing model.

The testing of the model was done in a real time environment in the lab using testing tools such as the JMeter. The result of the tests were obtained and compared to prove the usefulness of our model. The work presented here is still preliminary and there are still areas where the outlined system could be improved upon. The fact that there are so many areas in the system that could be further investigated is also encouraging because it means that there is plenty of scope for future work in this area. Although the model proposed by us solves the purpose of Dynamic load balancing on Web Server System, it can be considered as a base model or preliminary work. The main goal of the model is to minimize the worst case scenario i.e., to avoid that server node drops request and eventually crashes. To this purpose we have taken into consideration the CPU utilization and network traffic, but there are several other parameters that can be considered such as the no of active TCP/IP connections and setting a limit for maximum cluster utilization.

Considering the network bandwidth and proximity of the web server is also an important aspect. Further emphasis can be given on improving the throughput and response time of the web servers. The area has a great potential and a very important aspect of distributed system. Load balancing is a concept which is still under research. Everyday new algorithms are being developed and existing models are studied. Hence there is a vast scope for future enhancement.

References

[1] Cardellini, Colajanni, and Philip S. Yu, "Dynamic Load balancing on web-server systems", published in *IEEE internet Computing*, vol. 3, no. 3, pp 28-39, 1999.

[2] Cardellini, Colajanni, and Philip S. Yu, "Dynamic Load Balancing in geographically Distributed Heterogeneous Web Server", published in the proceedings of IEEE 18th International Conference on Distributed Computing Systems, at Amsterdam, The Netherlands, pp, 295-302, May 1998.

[3] Dheeraj Sanghi, Pankaj Jalote, Puneet Agarwal, "A Testbed For Performance Evaluation of Load Balancing Schemes for Web Server Systems", Department of Computer Science & Engineering, IIT Kanpur, India.

[4] D. Anderen, T. Yang, V. Holmedahl, O. H. Ibarra, "SWEB: Toward a scalable World Wide Web server on minicomputers", Proc. IPPS'06, Honolulu, pp. 850-856, April 1996.

[5] O. K. Tonguz and E. Yanmaz, "On the Theory of Dynamic Load Balancing," in Proc. IEEE Global Telecom. Conf. (GLOBECOM'03), vol. 7, pp. 3626-3630, Dec. 2003.

[6] J. C. I. Chuang, "Performance Issues and Algorithms for Dynamic Channel Assignment," IEEE J. Sel. Areas Commun., vol. 11, no. 6, pp. 955-963, Aug. 1993.

[7] E. Yanmaz and O. K. Tonguz, "Dynamic Load Balancing and Sharing Performance of Integrated Wireless Networks," IEEE JSAC Special issue on Advanced Mobility Management and QoS Protocols for Next Generation Wireless Internet, vol. 22, no. 5, pp. 862-872, June 2004.

[8] E.Yanmaz and O. K. Tonguz, "Location Dependent Dynamic Load Balancing,"in Proc. IEEE Global Telecom. Conf. (GLOBECOM'05), Dec. 2005, in press. IEEE Globecom 2005 602 0-7803-9415-1/05/$20.00 © 2005 IEEE.

[9] Abrahams, D. M. and Rizzardi, F., The Berkeley Interactive Statistical System. W. W. Norton, 1988.

[10] Agrawal, R. and Ezzet, A., Location independent remote execution in NEST. IEEE Trans. Softw. Eng. 13, 8 (Aug.), 905–912, 1987

[11] Ahmad, I., Ghafoor, A., and Mehrotra, K., Performance prediction of distributed load balancing on multicomputer systems. In Supercomputing. IEEE, New York, 830–839, 1991

[12] Artsy, Y. and Finkel, R., Designing a process migration facility: The Charlotte experience. IEEE Comput. 22, 1, 47–56, 1989

[13] Barak, A., Shai, G., and Wheeler, R. G., The MOSIX Distributed Operating System: Load Balancing for UNIX. Springer-Verlag, Berlin, 1993.

[14] Bonomi, F. and Kumar, A., Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. IEEE Trans. Comput. 39, 10 (Oct.), 1232–1250, 1990.

[15] Bryant, R. M. and Finkel, R. A., A stable distributed scheduling algorithm. In the 2nd International Conference on Distributed Computing Systems. 314–323, 1981.

[16] Cabrera, F. 1986. The influence of workload on load balancing strategies. In Proceedings of the Usenix Summer Conference (June). USENIX Assoc., Berkeley, Calif., 446–458.

[17] Casas, J., Clark, D. L., Konuru, R., Otto, S. W., Prouty, R. M., and Walpole, J., Mpvm: A migration transparent version of pvm. Comput. Syst. 8, 2 (Spring), 171–216,1995.

[18] Casavant, T. L. and Kuhl, J. G., Analysis of three dynamic distributed load-balancing strategies with varying global information requirements. In the 7th International Conference on Distributed Computing Systems (Sept.). IEEE, New York, 185–192, 1987.

[19] Chowdhury, S., The greedy load sharing algorithm. J. Parallel Distrib. Comput. 9, 93–99,1990.

[20] De Paoli, D. and Goscinski, A., The rhodos migration facility. Tech. Rep. TR C95-36, School of Computing and Mathematics, Deakin Univ., Victoria, Australia. Available via http://www.cm.deakin.edu.au/ rhodos/rh tr95.html#C95-36. Submitted to the Journal of Systems and Software, 1995.

[21] Douglis, F. and Ousterhout, J., Transparent process migration: Design alternatives and the Sprite implementation. Softw. Pract. Exper. 21, 8 (Aug.), 757–785, 1991.

[22] Downey, A. B. and Harchol--Balter, M., A note on "The limited performance benefits of migrating active processes for load sharing." Tech. Rep. UCB/CSD-95-888, Univ. of California, Berkeley, 1990.

[23] Eager, D. L., Lazowska, E. D., and Zhaorjan, J., Adaptive load sharing in homogeneous distributed systems. IEEE Trans. Softw. Eng. 12, 5 (May), 662–675, 1986.

[24] Eager, D. L., Lazowska, E. D., and Zahorjan, J., The limited performance benefits of migrating active processes for load sharing. In the ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems. ACM, New York, 662–675, 1988.

[25] Epema, D., An analysis of decay-usage scheduling in multiprocessors. In the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems. ACM, New York, 74–85, 1995.

[26] Evans, D. J. and Butt, W. U. N., Dynamic load balancing using task-transfer probabilities. Parallel Comput. 19, 897–916, 1993.

[27] Hac, A. and Jin, X., Dynamic load balancing in a distributed system using a senderinitiated algorithm. J. Syst. Softw. 11, 79–94, 1990.

[28] Hennessy, J. L. and Patterson, D. A., Computer Architecture: A Quantitative Approach. Morgan Kaufmann, San Mateo, California, 1990.

[29] Krueger, P. and Livny, M., A comparison of preemptive and non-preemptive load distributing. In the 8th International Conference on Distributed Computing Systems. IEEE, New York, 123–130, 1988.

[30] Kunz, T., The influence of different workload descriptions on a heuristic load balancing scheme. IEEE Trans. Softw. Eng. 17, 7 (July), 725–730, 1991.

[31] Larsen, R. J. and Marx, M. L., An Introduction to Mathematical Statistics and Its Applications. 2nd ed. Prentice-Hall, Englewood Cliffs, N.J., 1986.

[32] Leland, W. E. and Ott, T. J., Load-balancing heuristics and process behavior. In Proceedings of Performance ’86 and ACM SIGMETRICS. Vol. 14. ACM, New York, 54–69, 1986.

[33] Lin, H. C. and Raghavendra, C., A state-aggregation method for analyzing dynamic load-balancing policies. In the IEEE 13th International Conference on Distributed Computing Systems. IEEE, New York, 482–489, 1993.

[34] Litzkow, M. and Livny, M., Experience with the Condor distributed batch system. In the IEEE Workshop on Experimental Distributed Systems. IEEE, New York, 97–101, 1990.

[35] Litzkow, M. and Livny, M., and Mutka, M., Condor—A hunter of idle workstations. In the 8th International Conference on Distributed Computing Systems. IEEE, New York, 1988.

[36] Livny, M. and Melman, M., Load balancing in homogeneous broadcast distributed systems. In the ACM Computer Network Performance Symposium (April). ACM, New York, 47–55, 1982.

[37] Milojicic, D. S., Load distribution: Implementation for the Mach microkernel. Ph.D. dissertation, Univ. of Kaiserslautern, Kaiserslautern, Germany, 1993.

[38] Mirchandaney, R., Towsley, D., and Stankovic, J. A., Adaptive load sharing in heterogeneous distributed systems. J. Parallel Distributed Computing 9, 331–346, 1990.

[39] Powell, M. and Miller, B., Process migrations in DEMOS/MP. In the 6th ACM Symposium on Operating Systems Principles (Nov.). ACM, New York, 110–119, 1983.

[40] Pulidas, S., Towsley, D., and Stankovic, J. A., Imbedding gradient estimators in load balancing algorithms. In the 8th International Conference on Distributed Computing Systems (June). IEEE, New York, 482–490, 1988.

Authors

Harikesh Singh is working as Senior Lecturer in Department of Computer Science & Engineering, Jaypee University of Engineering & Technology, Guna (MP), INDIA since 2007. He is pursuing research work in the area of Grid Computing.

Dr. Shishir Kumar is working as Professor and Head of Department of Computer Science & Engineering in Jaypee University of Engineering & Technology, Guna (MP), INDIA since 2005. His area of interest is Distributed Computing.