OPTIMIZATION OF WEB SERVER THROUGH A DOMAIN NAME SYSTEM APPROACH

Dr varaPrasad .s. .Kondapalli [1]
[1] **Director&principal,G.H. Raisoni college of engg&mgmt.**
Ahmednagar,maharastra,india
Vara_sr@yahoo.com

Abstract

A clustered web system with one virtual URL-name is one of the possible approaches to handle ever increasing client requests to popular websites. This system maintains a single interface to the users & has the potential to provide better load balancing. The client HTTP requests can be assigned to the web server with the least load by the IP address dispatcher. The IP address dispatcher assigns client requests to the web server through the packet rewriting mechanism that modifies the destination address of each incoming packet to the address of the selected web server. However, the task of rewriting the address fields of all packets can cause the IP dispatcher to become a bottleneck when the system is overloaded with heavy client requests. The DNS maps the URL name to the IP address of one of servers in the web clusters through the round robin scheduling policy. The DNS dispatcher based cluster does not present risk of bottleneck but it control distribution of user requests in a limited way due to the IP address caching in the DNS.

Main concern is on an alternative approach that integrates the DNS based dispatching mechanism with redirection technique based on load information of the web servers. The local name server collects the load information from the web servers periodically.

The purpose of distributing Internet traffic between various web-server nodes in a web-server system, DNS plays an important role. DNS not only provides load balancing but network scalability and fault tolerance can also be achieved as well. In this project work the DNS is integrated with an adaptive load balancing approach that dynamically modifies zone records in authoritative name server that is based on Load Average (LA) value of various web-server nodes. By controlling Resource Record Set (RR Set) in zone records, client will get reference of web-server node that is having least value of load average, there-by achieving load balancing. A new web-server will automatically register itself in the zone records there-by achieving scalability. If any of the web-server node is down, then it will automatically removed from zone records so IP of this web-server node that will not accessed by any client.

We have measured performance parameters like Load Average (LA) and Throughput of the various web-server nodes. The experimental results show that the proposed scheme achieves better performance and provide scalable and fault tolerant system than the default load balancing scheme based on the random basis policy.

1. Introduction

Now days, many popular websites experience a high rate of traffic from the users. Typical example of this is like Google, Yahoo, and various sites accessed by the user all over the world. To distribute the load of high requests from the users, these sites use mirror servers.

The concept of load balancing is very important in order to distribute the load among various web-servers that serves one site. Load balancing technique is used to reduce the response time and provide users the best available quality of service. Several approaches can be used to achieve load balancing among the servers with different degree of effectiveness.

1.1 Problem Domain

As the World Wide Web (WWW) increases in size and complexity day by day, it is necessary to find the solution to manage high load traffic on web-servers during peak hours. One possible solution is to have a single web-server running on a highly configured machine with fast processor, high storage capacity etc. The problem solving in this way is limited because the solution is not much scalable.

As soon as the volume of traffic increases, there arises the need to modify the configuration of the machine in which web-server is running. There are cases in which one must even forced to replace hardware. So this solution is not fault tolerant as if there is problem arises during peak hours there is no backup. In the modern systems we are identifying three requirements that should meet in order to publish information:

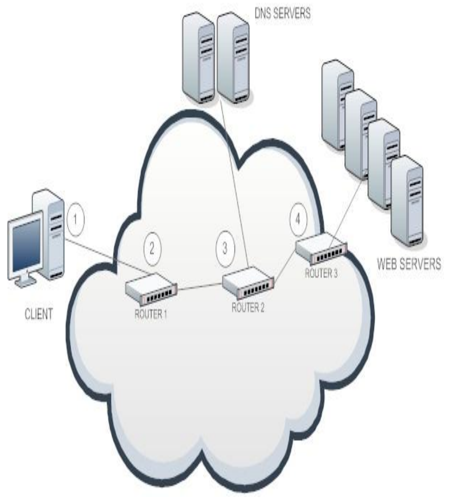
- Network Scalability: Preserving the used hardware architecture and adding only a new HTTP server running on a different machine, whenever the incoming traffic to an existing HTTP server increases.
- Load Balancing: Sharing traffic among a group of HTTP servers according to some policies which depend on local load or some pseudo random heuristic.
- Fault Tolerance: In case of fault of one of the servers we want to be able to recover, stopping its use and replacing it with one of alive servers automatically.

- **1.2 Solution Domain**
- A web-server system with one virtual URL-Name and multiple IPs is one of the possible approaches to handle ever increasing client requests to popular web sites. This system maintains a single interface to the users and has the potential to provide better load balancing. In the translation process from the symbolic name (URL) to IP address, DNS can select any node of the web-server system. In particular, this translation process allows the DNS to implement various RR Set scheduling policies at authoritative name server to select the appropriate web-server node. In this work, we will focus on an approach that integrates the DNS-based dispatching mechanism with a redirection technique based on the load average (LA) information of the web-server system. The local name server that is authoritative for this domain collects the LA information from various web-server nodes periodically and sorts the zone records with increasing LA. By controlling RR Set ordering at DNS, client will get reference of web-server node that is having least value of LA. Preserving the used hardware architecture and adding only a new web-server node with different IP, it will dynamically registered in the zone records at DNS, no manual modification in zone record is required. Firewall is used at DNS to prevent any unauthorized modification in zone records. These techniques provide a scalable web-server system. In case of fault of any of the web-server node, it will be removed from the zone records dynamically and no client will get reference of this web-server node thus making system fault tolerant. This project work describes the performance of the proposed web-server system. We experiment the system using the request generator, JMeter.

- **1.3 System Domain**
- Load balance is an important step for internet growth. In this project work we review the solutions proposed by researchers. We concentrate our attention to WWW, since it is the most used internet service. Appropriate solution for other services such as FTP, NTTP or proprietary protocols may also be focused in same manner.
- **1.4 Application Domain**
- The DNS makes it possible to assign domain names to organizations independent of the physical routing hierarchy represented by the numerical IP address because of this, hyperlinks and internet contact information can remain the same, whatever the current IP routing arrangements may be, and can take a human-readable form (such as "`example.com`"). These internet names are easier to remember than the IP address 203.45.118.126. People take advantage of this when they recite meaningful URLs and e-mail

addresses without caring how the machine will actually locate them.

- The Domain name System distributes the responsibility for assigning domain names and mapping them to IP networks by allowing an authoritative name server for each domain to keep track of its own changes, avoiding the need for a central register to be continually consulted and updated.
- **1.5 Scope of Work**
- In the current work, we have focused on DNS based load balancing approach, with a redirection technique based on the Load Average (LA) information of the Web-Server nodes. The local name servers collects the LA information from web-servers node periodically and arrange zone records, so that client request can be directed to the web-server node that is having least value of LA.
- The measured performance of the DNS-based distributed web-server system. Moreover, we used the request generator, JMeter in order to generate client requests. The experimental results show that the proposed scheme achieves better performance than the default load balancing scheme based on the random policy.
- **2.1 Load Balancing- An Overview**
- The traffic on the World Wide Web increases and causes increase in client requests to popular websites, when special events happen such as Olympic Games and General Elections. To improve the capacity of server, Site administrators face many problems. One approach to improve the capacity of server is to replicate information across a mirrored server. In this approach users manually select alternative URLs for a web-site. However, this technique is not user-transparent, and also not allows controlling requests distribution. Another solution to balance the load among the web-servers is a DNS based, that can distribute incoming request from the client among several web-server nodes.



- **Figure 2.1: Topology for Load balancing**
-
- A In normal practice client-side load-balancing is not involved, but it is indeed possible, when using Netscape browser in which a simple balancing algorithm is incorporated in their navigator browser, making it to choose random web-servers.
- B Routing protocols are used, such as Border Gateway protocol, used for data-exchange between large internet operators.
- C Client request for an ip address by typing website name in URL.
- D DNS server is used to convert the website name typed by the client into valid ip and give back this ip address to client.
- E Various web-servers is used to serve the one site.
- **2.3 Load Balancer Components**
- A few fundamental components or concepts are used to build any type of load balancer, regardless if it is implemented in hardware or software. A balancing device must be able to receive and send packets through some form of data-forwarding plane. It must have an algorithm that decides how the load should be balanced between available the nodes. These two components are basically sufficient for a load balancer

to work. Depending on the requirements of network traffic and load distribution, they can have varying degrees of sophistication. A third component found in many balancers is a health-check mechanism that enables the load-balancing algorithm take into account server health, e.g. availability and load, when distributing traffic.

- **2.4 The Domain Name System**
-
- In the internet the Domain Name System (DNS) is the global name lookup service. It is a hierarchical, redundant and distributed database running on thousands of servers worldwide, each responsible for one or more DNS zones. When a client issues a request for a URL in the form of any website name, the browser will first try to resolve the hostname in the URL into an IP address, so that it knows where to send the request. This is where it is possible for a DNS server to influence the outcome of a query, effectively directing the client to a least loaded web server, if the requested site more than one web server. However, the DNS approach based on load balancing is not without challenges, as will be discussed shortly.
- If we go more in analyzing on how DNS works, it is easier to understand the challenges. Figure 2.3, shows a step by step diagram of a DNS query for a site.
- A. First, the browser extracts the hostname www.example.net from the URL and runs it through a gethostbyname() system call.
- B. The operating system redirects the request to the local resolver (typically ISP name server), asking for the A-record for www.example.net – an A-record is a standard hostname-to-IP mapping. The request to the resolver is recursive, which enables a flag meaning "I only want the final answer."
-

Figure 2.2: Standard DNS Lookup Procedure.

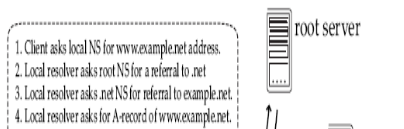
C. The local resolver performs several *iterative* queries; iterative meaning "direct me to a better match." First it asks one of the thirteen root DNS servers. These know which servers control the top-level domains like com, org and net. The local resolver caches the response.

D. Further, the resolver asks one of the .net DNS servers for further directions to the example.net domain. Again, it caches the response.

E. Then when the iteration process reaches one of the example.net DNS servers, it will know the answer to the query, and reply with an IP address, e.g. 192.0.34.166. Yet again, the resolver caches the response.

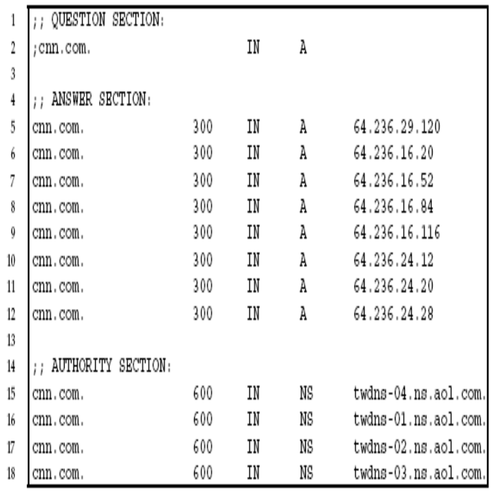
F. The response is sent back to the client operating system, which also caches the response.

G. The IP address is returned to the browser, which in turn can contact the server and retrieve the content. In addition to the operating system caching the response, most popular browsers will do so as well. Note how this rather simple example implies at least five levels of caching, not counting potential intermediate proxy nodes, e.g. http proxies. To give an insight into how name servers respond to queries, consider the listing below, showing what a client would receive when asking for the


1. Client asks local NS for www.example.net address.
2. Local resolver asks root NS for a referral to .net
3. Local resolver asks .net NS for referral to example.net.
4. Local resolver asks for A-record of www.example.net.
root server

IP address of the cnn.com host. This particular output is from the ‘dig’ application, and shows three main sections: The question section echoes the request information; the answer section shows the matching records (if any); the authority section lists the authoritative name servers for the domain.

Figure 2.3: Response of “DIG” Application



```
;; QUESTION SECTION:
;cnn.com.                   IN    A

;; ANSWER SECTION:
cnn.com.            300     IN    A       64.236.29.120
cnn.com.            300     IN    A       64.236.16.20
cnn.com.            300     IN    A       64.236.16.52
cnn.com.            300     IN    A       64.236.16.84
cnn.com.            300     IN    A       64.236.16.116
cnn.com.            300     IN    A       64.236.24.12
cnn.com.            300     IN    A       64.236.24.20
cnn.com.            300     IN    A       64.236.24.28

;; AUTHORITY SECTION:
cnn.com.            600     IN    NS      twdns-04.ns.aol.com.
cnn.com.            600     IN    NS      twdns-01.ns.aol.com.
cnn.com.            600     IN    NS      twdns-02.ns.aol.com.
cnn.com.            600     IN    NS      twdns-03.ns.aol.com.
```

There are several interesting properties of the DNS response in the listing. First of all, it is clear that the name servers return multiple A-records for the cnn.com Host name – this is known as a resource record set (RR set). Also, the addresses appear to be within the same provider network; the provider is verified to be America Online by querying the *whois* database for the IP addresses. Now, even if all addresses are within a very close range, this does not mean that they are geographically close to each other (for reasons which will be discussed in a later section about any cast routing). However, tracing the path to each IP from multiple locations around the world shows that they are most probably located in the same city, and maybe even the same data centre.

The authoritative name servers for the cnn.com domain have more varied IP addresses (not shown here). Therefore it seems the name servers are more geographically dispersed – closer inspection reveals that they are hosted in different operator networks. In other words, AOL cooperates with other operators to provide a redundant DNS service, a very common practice.

Going back to the answer section, it is obvious that some form of load balancing scheme is running, though it is difficult to determine exactly what kind it is. Consecutive queries to one of the main DNS servers show that for each request, the A-record set is returned in a seemingly shuffled order. This could mean that the balancing mechanism relies on the DNS server responding with an address set in a given order, be it random or otherwise. Clients typically traverse the set sequentially, starting from the top. That is, if the first address on the list does not work, the client tries the next one, and so on. This is however highly implementation-specific.

The numbers in the second column of the response show the time-to-live integer value (TTL) in seconds. This value governs how long the answer is cached in intermediate nodes, e.g. local resolvers. As long as the TTL is 0 or higher, queries will be answered from the cache instead of being redirected to other servers. In the example, the A-records have a TTL of 5 minutes. Comparably, single host, low-traffic sites may operate with a TTL in the range of hours to days – informational documents recommend a value in the range of minutes when deploying DNS-based load balancing.

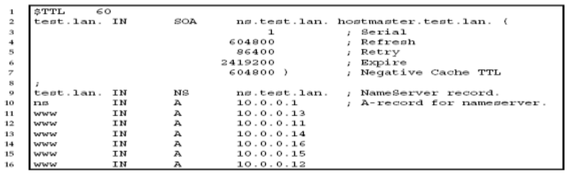
A low TTL ensures that DNS servers are queried often, and are hence given the possibility to influence the answer over relatively small time intervals. Low TTLs come at the cost of higher frequency of queries, which can add a considerable delay to the total page response time.

In conclusion, we observe that load-balancing using DNS can adopt two basic mechanisms. First, delivering a resource record set in a given order; second, setting the TTL to a relatively low value. Unfortunately, these are by no means reliable. Considering the first point; no Internet standards or authoritative documentation require DNS implementations to preserve the order of resource record sets. Even if it could be considered a rule of thumb in the Internet community to leave any record set ordering intact, there is no reason to assume that all DNS software would follow such recommendations. As for TTL values, multiple levels of caching make it a challenge to predict and control the actual TTL observed by the end us

2.4.1 Example of Uses of DNS

Basic DNS-based load balancing does not require any complex configuration to work. The following listing is an example taken from a normal BIND zone file with standard syntax, and it describes the fictional test.lan zone:

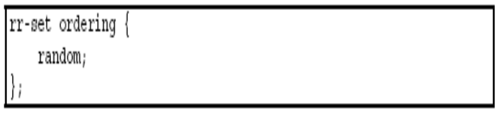
.



```
1   $TTL     60
2   test.lan. IN       SOA      ns.test.lan. hostmaster.test.lan. (
3                                        1          ; Serial
4                                   604800          ; Refresh
5                                    86400          ; Retry
6                                  2419200          ; Expire
7                                   604800 )        ; Negative Cache TTL
8   ;
9   test.lan. IN       NS       ns.test.lan.    ; NameServer record.
10  ns        IN       A        10.0.0.1        ; A-record for nameserver.
11  www       IN       A        10.0.0.13
12  www       IN       A        10.0.0.11
13  www       IN       A        10.0.0.14
14  www       IN       A        10.0.0.16
15  www       IN       A        10.0.0.15
16  www       IN       A        10.0.0.12
```

Figure 2.4: Example "ZONE" File

The first line defines the time-to-live value to use for all the records contained within the Zone file. Next, on lines 2 through 7, is the start of authority record, which describes behavior of slave name servers. The relevant records, however, are the six A-records that make up the RR set for the www.test.lan hostname. As a result, any lookup for the A-record for www.test.lan would return the entire set of addresses to the querying client. This leads us to the question of RR set ordering, i.e. in which order are the records returned to the client. As previously mentioned, RR set ordering is not governed by any authoritative or recommended standards, so, it is entirely up to the implementation of DNS software how to handle ordering of RR sets. BIND, major version 9, provides three basic methods of ordering, designated fixed, cyclic and random. The desired ordering can be specified in the BIND Configuration file.



```
rr-set ordering {
    random;
};
```

Figure 2.5: Example RR-Set Ordering

2.5 DNS-Based Approach for Load Balancing

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. 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. 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 [2] .

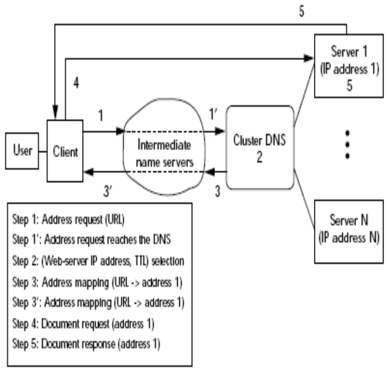

5
Server 1
(IP address 1)
5
4
1
1′
User
Client
Intermediate
name servers
Cluster DNS
2
3′
3
Server N
(IP address N)
Step 1: Address request (URL)
Step 1′: Address request reaches the DNS
Step 2: (Web-server IP address, TTL) selection
Step 3: Address mapping (URL -> address 1)
Step 3′: Address mapping (URL -> address 1)
Step 4: Document request (address 1)
Step 5: Document response (address 1)

Figure 2.6: DNS-Based Approach to Load Balancing.

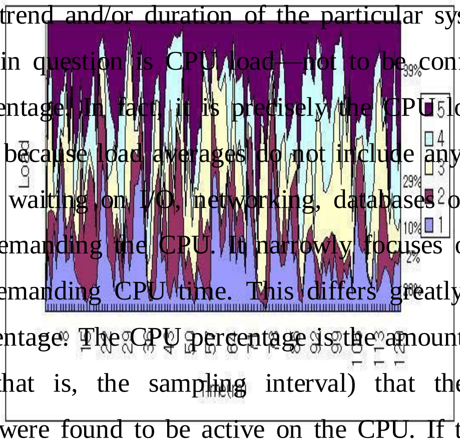
Figure 2.7 shows both resolutions. This figure, like those in the following sections, shows the different approaches for distributing requests on the basis of a protocol-centered description. If an intermediate name server holds a valid mapping for the cluster URL, it resolves the address-mapping request 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.

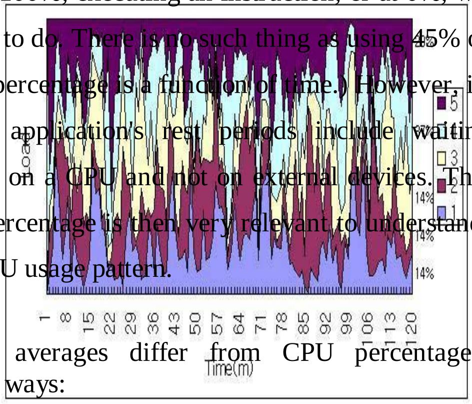
On the other hand, the limited control on client requests prevents the DNS from becoming a potential bottleneck .We distinguish the DNS-based architectures by the scheduling algorithm that the cluster DNS uses to balance the Web-server nodes' load. With constant TTL algorithms, the DNS selects servers 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.

2.6 Understanding Load averages as opposed to CPU usage

Many Linux administrators and support technicians regularly use the "**top**" and "**uptime**" utility for real-time monitoring of their system state. Top is rich with information—memory usage, kernel states, process priorities, process owner and so forth all can be obtained from top [16].

The three load-average values in the first line of top output are the 1-minute, 5-minute and 15-minute average. (These values also are displayed by other commands, such as uptime, not only top.) That means, reading from left to right, one can examine the aging trend and/or duration of the particular system state. The state in question is CPU load—not to be confused with CPU percentage. In fact, it is precisely the CPU load that is measured, because load averages do not include any processes or threads waiting on I/O, networking, databases or anything else not demanding the CPU. It narrowly focuses on what is actively demanding CPU time. This differs greatly from the CPU percentage. The CPU percentage is the amount of a time interval (that is, the sampling interval) that the system's processes were found to be active on the CPU. If top reports that your program is taking 45% CPU, 45% of the samples taken by top found your process active on the CPU. The rest of the time your application was in a wait. (It is important to remember that a CPU is a discrete state machine. It really can be at only 100%, executing an instruction, or at 0%, waiting for something to do. There is no such thing as using 45% of a CPU. The CPU percentage is a function of time.) However, it is likely that your application's rest periods include waiting to be dispatched on a CPU and not on external devices. That part of the wait percentage is then very relevant to understanding your overall CPU usage pattern.





The load averages differ from CPU percentage in two significant ways:

A. Load averages measure the trend in CPU utilization not only an instantaneous snapshot, as does CPU percentage.

B. Load averages include all demand for the CPU not only how much was active at the time of measurement.

Taking the discussion back to the machinery at hand, the load averages tell us by increasing duration whether our physical CPUs are over or under-utilized. The point of perfect utilization, meaning that the CPUs are always busy and, yet, no process ever waits for one, is the average matching the number of CPUs. If there are four CPUs on a machine and the reported one-minute load average is 4.00, the machine has been utilizing its processors perfectly for the last 60 seconds. This understanding can be extrapolated to the 5- and 15-minute averages. In general, the intuitive idea of load averages is the higher they rise above the number of processors, the more demand there is for the CPUs, and the lower they fall below the number of processors, the more untapped CPU capacity there is. But all is not as it appears.

Load average

- Load average is intended to provide some kind of information about how much work has been done on the system in the recent past 1 minute, the past 5 minutes and the distant past 15 minutes.
- Load average is not about utilization but the total queue length.

Figure 2.7: Shows the Average CPU Load of Each Web-Server in the Web-Server System Using the DNS Based Round-Robin Scheduling Policy.

Figure 2.8: **Average CPU Load of Web-Servers Based on the Proposed Load Balancing Scheme**

3.2 Distributed Web-server System

We consider a distributed Web-server system with a generic structure as shown in Figure3.1 Generally a Web-server system architecture consists of three entities: the Client, the Domain Name Server (DNS) and the Web-Server. The cluster Web-server system can be organized into several Web-Servers nodes and a DNS that resolves all initial address resolution requests from local gateways. Each client session can be characterized by one address resolution and several Web page requests. At first, the client receives the address of one Web-server of the cluster through the DNS address resolution. Subsequently, the client submits several HTTP requests to the Web server. In addition to resolving the URL-name to the IP address of a Web-server, the DNS of a cluster Web-Server system can collect information from Web-Servers for various statistics. Moreover, the DNS can select the address of a Web-Server based on the collected Information. In order to select the address of the suitable Web-Server, the DNS could use some scheduling policy to balance the load among several Web-Servers to avoid becoming overloaded**.**

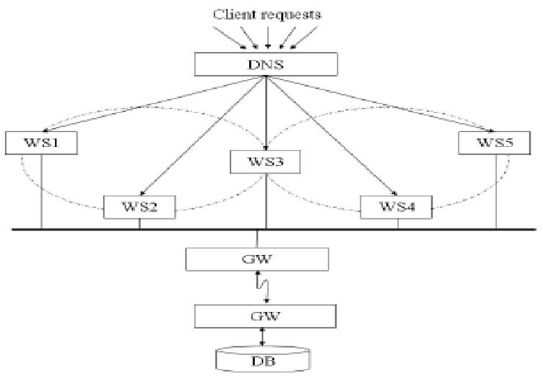


Figure 3.1: Generic Structure of the Web-Server System

Many existing distributed Web-server cluster systems assign the client requests arriving at the DNS in a random manner among the Web-servers. The Random DNS policy is efficient in the system where the client requests from local gateways are uniformly distributed due to IP-address caching mechanism at the client. Another approach to the DNS scheduling policy is to allow the DNS to select a Web-server from the cluster based on some load information from the Web-servers. The DNS can collect various kinds of data from the Web servers such as history of server state, the number of active server connections or detailed processor loads. Most conventional load balancing schemes have used this kind of approach using the load information from servers. We present some simple strategies in order to improve the performance of the distributed Web-server cluster system. In this work, we focus on Load Average (LA) value from various web servers. The load average represents the number of computers you would need to be able to run all of the processes at the same time.

3.4 Hardware Consideration

An Ethernet based LAN with six Pentium Machines will be used. One will be used as Client with Ms Windows XP, one as DNS Server with LINUX OS and BIND 9, four Machines will be used as Web-Servers with LINUX OS and Apache 2.0 as Web-Server. Load generator JMeter is used on Windows Client. The hosts within the network are interconnected according to the figure 3.2.Web-Servers are running on IP 192.168.1.2, 192.168.1.3, 192.168.1.4, 192.168.1.5. DNS Server is running on IP 192.168.1.100. Web-client is having an IP of same network segments.

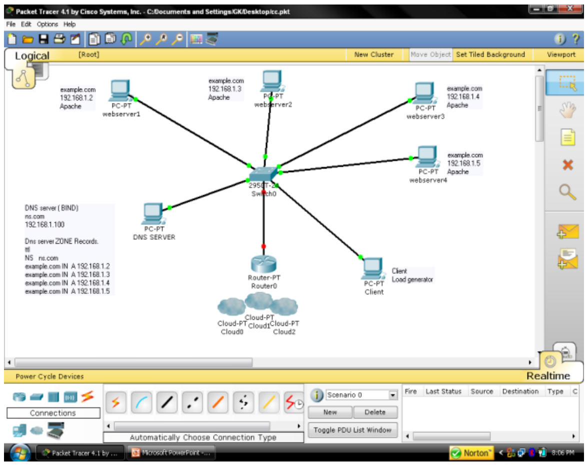
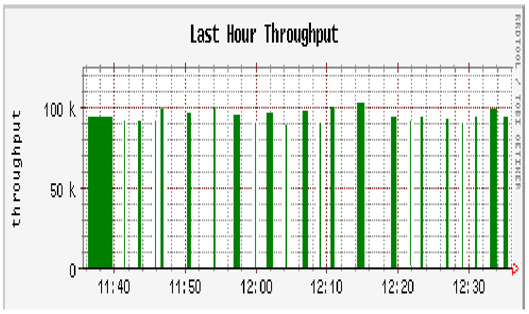


Figure 3.2: Experimental Setup for Web-Server System

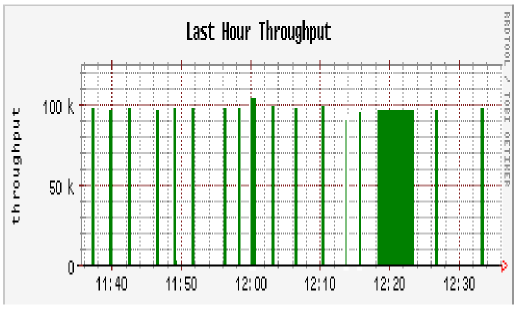
Figure 3.2: Experimental Setup for Web-Server System

4.2.1 Measurement of Throughput on each Web-Server node with Fixed Zone record and Random RRSet Ordering:



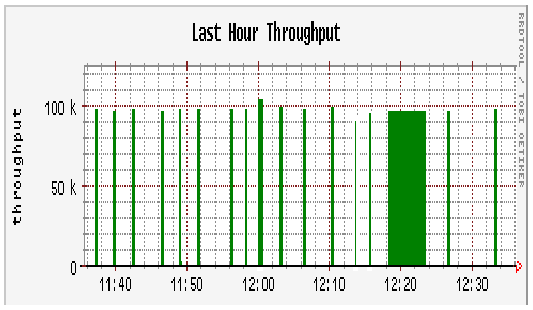
Graph 4.9: Throughput on Web-Server Node 192.168.1.2 Average Throughput 36.31 KB/S

Graph 4.9 shows throughput on vertical axes against time on horizontal axes for first Web-server node. Average value of throughput is 36.31 KB/S. As shown in curve, transportation of the data is random, and it is non uniform with time. At any instance of time, high value of throughput shows that HTTP request is being processed by node and zero throughputs is indication that no HTTP request is reaching on this node



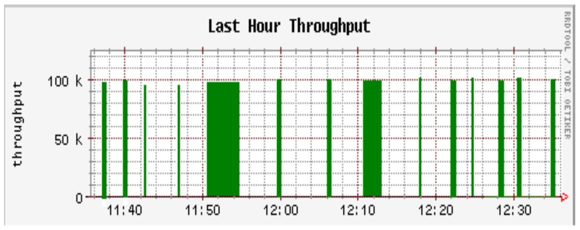
Graph 4.10: Throughput on web-server node 192.168.1.3Average Throughput 25.45 KB/S

Graph 4.10 shows throughput for second Web-server node. Average value of throughput is 25.45 KB/S. As shown in curve, transportation of the data is random, and it is non uniform with time.



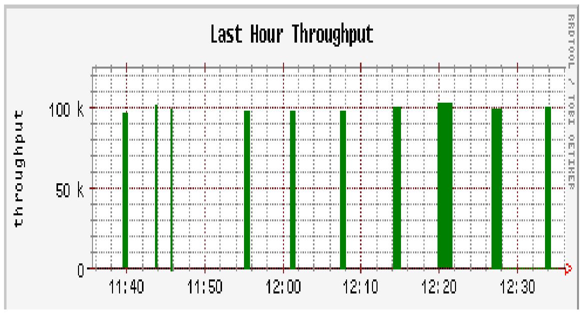
Graph 4.10: Throughput on web-server node 192.168.1.3 Average Throughput 25.45 KB/S

Graph 4.10 shows throughput for second Web-server node. Average value of throughput is 25.45 KB/S. As shown in curve, transportation of the data is random, and it is non uniform with time.



Graph 4.11: Throughput on Web-Server Node 192.168.1.4 Average Throughput 28.19 KB/S

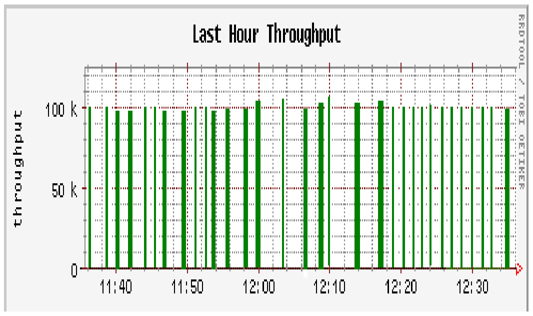
Graph 4.11 shows throughput for third Web-server node. Average value of throughput is 28.19 KB/S. As shown in curve, transportation of the data is random, and it is non uniform with time.



Graph 4.12: Throughput on Web-Server Node 192.168.1.5 Average Throughput 20.72 KB/S

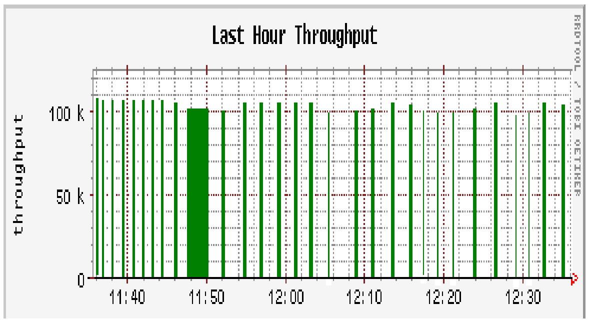
Graph 4.12 shows throughput for fourth Web-server node. Average value of throughput is 20.72 KB/S. As shown in curve, transportation of the data is random, and it is non uniform with time.

4.2.2 Measurement of Throughput on each Web-Server node with Varying Zone record and Fixed RRSet Ordering:



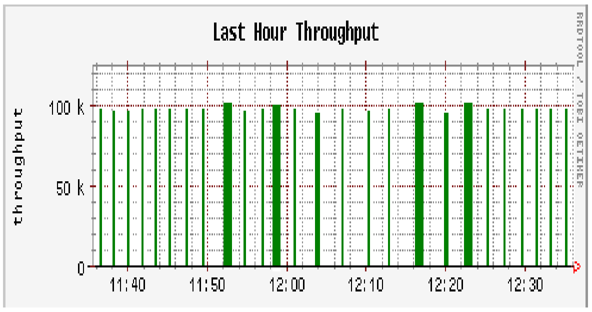
Graph 4.13: Throughput on web-server node 192.168.1.2 Average Throughput 40.90 KB/S

Graph 4.13 shows throughput for first Web-server node. Average value of throughput is 40.90 KB/S. As shown in curve, transportation of the data is regular, and it is more uniform with time.



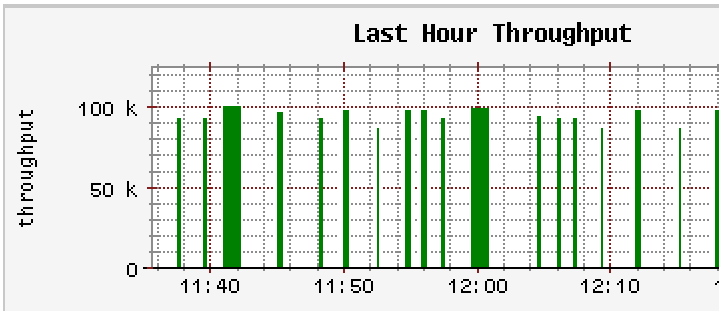
Graph 4.14: Throughput on web-server node 192.168.1.3 Average Throughput 42.72 KB/S

Graph 4.14 shows throughput for second Web-server node. Average value of throughput is 42.72 KB/S. As shown in curve, transportation of the data is regular, and it is more uniform with time.



Graph 4.15: Throughput on Web-Server Node 192.168.1.4 Average Throughput 34.54 KB/S

Graph 4.15 shows throughput for third Web-server node. Average value of throughput is 34.54 KB/S. As shown in curve, transportation of the data is regular, and it is more uniform with time.



Graph 4.16: Throughput on Web-Server Node 192.168.1.5 Average Throughput 35.73 KB/S

Graph 4.16 shows throughput for fourth Web-server node. Average value of throughput is 35.73 KB/S. As shown in curve, transportation of the data is regular, and it is more uniform with time.

With Random RRSet ordering at DNS, average value of Throughput is different on each web-server nodes. Maximum average value is 36.31 KB/S and minimum average value is 20.72 KB/S. Moreover, throughput is not uniform for the complete 60 minutes interval. With Fixed RRSet ordering at DNS, Maximum average value is 42.72 KB/S and minimum average value is 34.54 KB/S. So throughput is improved on each web-server node. Moreover, throughput is more uniform for the complete 60 minutes interval.

Table 4.3.1 Comparison between Random Vs Fixed RR-set Ordering in Load Average

Web-server Node	Average value of LA with Random RRset Ordering (LA / S)	Average value of LA with Fixed RRset Ordering (LA / S)
192.168.1.2	0.681	0.216
192.168.1.3	0.521	0.218
192.168.1.4	0.341	0.187
192.168.1.5	0.262	0.220

Table 4.3.2 Comparison between Random Vs Fixed RR-set Ordering in LA Throughput

Web-server Node	Average value of Throughput with Random RRset Ordering. (KB/S)	Average value of Throughput with Fixed RRset Ordering (KB/S)
192.168.1.2	36.31	40.93
192.168.1.3	25.45	42.72
192.168.1.4	28.19	34.54
192.168.1.5	20.72	35.73

We compared the three different parameters for the various Web-Server nodes based on fixed Zone records and random RR set ordering vs. the varying Zone records and fixed RRSet ordering, in Authoritative Name Server. The proposed Web-server system based on varying Zone records and fixed RR set ordering at DNS can achieve better performance than the system based on the fixed zone record and Random RRSet ordering at DNS. This result is likely due to the Load Average information collected from each Web-server nodes, by the authoritative name server.

5 Conclusion

As seen in the previous chapter the experiment run on different PCs, that includes a DNS server (takes request from the different clients and forward this request to different Web Servers), A Web Server System (in which a site is running and these servers used to serve the client but the client they serve choose by the DNS), And a Client (Which generate load for different Web Severs). In our Experimental setup these all are connected through switch, in order to communicate with each other. In order to maintain load balance in different Web server, we use different Software and java programs that collect the load from the different Web server and dump it on the DNS server. So there is a use of two programs one is the Server program that run on DNS machine and another one is the Client program that run on the different web server.

And finally when the loads collect at the DNS server then DNS server able to decide that the Client request goes to which web server according to there loads.In this work our main emphasis is to balance the load in different web server by using the parameter called Load Average (LA). In order to show the difference between the load balancing mechanism used by DNS system previously and the mechanism used in this project , we plot a graph which shows that this mechanism is lot better than the previous one.

We have measured the performance of Web-server system, where Authoritative Name Server distribute client request between various Web-server nodes and thereby provide Load Balancing.No client request will be forwarded to a Web-Server node that is down and new web-server node will automatically registered in Zone records thereby providing fault tolerant (stopping use of a non working system) and scalable Web-Server system .The experimental results show that the proposed scheme achieves better performance than the default Load Balancing scheme based on the Random policy. Throughout this work, we have examined the use of Domain name system as mechanism for load balancing. In Authoritative Name server, TTL is kept zero so cashing of the records is not permitted. This work can be further extended in a WAN environment so that other parameter like Network Bandwidth, Geographical distribution of the various web-server nodes and Network traffic level can be taken into the consideration.

References

[1] Sven ingebright "High level load balancing for web services. Masters thesis" 2006,Uni. of Oslo.

[2]VALERIA CARDELLINI University of Rome Tor Vergata, MICHELE COLAJANNI University of Modena , PHILIP S. YU IBM T.J. Watson Research Center "dynamic Load balancing on web server system IEEE Internet Computing 1999.

[3]Y. S. Hong, J. H. No and S.Y. Kim "DNS-Based Load Balancing in Distributed Web-server Systems "Proceedings of the Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems and Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06) 2006 IEEE.

.[4] N. Aghdaie and Y. Tamir, "Performance optimizations for transparent fault-tolerant Web services",IEEE Pacific Rim Conference on communication, Computer and Signal Processing, Victoria, Canada,

[5] Roberto Baldoni, Simona Bonamoneta, Carlo Marchetti, "Implementing Highly-Available WWWServers Based on Passive Object Replication", Second IEEE International Symposium on Object-Oriented Real-Time Distribute Computing, Saint-Malo, France, (May, 1999).

[6] V. Cardellini, E. Casalicchio, and M. Colajanni,"The state of the art in locally distributed Web-server systems", ACM Computing Surveys, Vol. 32, No. 2, pp.
263-311, (Jun. 2002).

[7] V. Cardellini, M. Colajanni, P.S. Yu, "Request redirection algorithms for distributed Web systems", IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 4, pp. 355-368, (April 2003).

[8] V. Cardellini, E. Casalicchio, M. Colajanni, P.S. Yu, The state of the art in locally distributed Web-server systems, IBM Research Report, RC22209(W0110-048), (October 2001).

[9] M.Castro, M. Dwyer and M. Rumsewicz, "Load balancing and control for distributed World Wide Web servers", IEEE Int. conf. On Control Applications, Hawaii, USA, pp.1614-1619, (Aug. 1999).

[10] M. Colajanni, P. S. Yu, and D. M. Dias, "Analysis of task assignment policies in scalable distributed Web-server Systems", IEEE Transactions on Parallel and Distributed Systems, Vol. 9, No. 6, pp. 585-600, (June 1998).

[11] Y.S. Hong, J.H. No, and In Han, "Evaluation of fault-tolerant distributed Web systems", Proc. WORDS 2005 (IEEE CS 2005 Workshop on Object-Oriented Real-Time Dependable Systems), Sedona, U.S.A. (Feb. 2005)

[12] H. Yokota, S. Kimura, and Y. Ebihara, "A proposal of DNS-based adaptive load balancing method for mirror server systems and its implementation",18th IEEE Int. Conf. On Advanced Information Networking and Application, 2004.

Web References

[13] http:// jakarta.apache.org

[14] http://linux.com

[15] Pro DNS and BIND: Ron Aitcheson

[16] UNIX Load Average: How It Works by Dr. Neil Gunther.

[17] RFC 1794 (DNS support for Load Balancing).

[18] Java computing, http://sun.com/java

[19] Apache's site http:// apache.org

[20] RFC 1032 Domain Administrators Guide.

[21] Wikipedia DNS- Wikipedia, the free encyclopedia