A Study of Replicated and Distributed Web Content

by

Nitin John

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

August 2002

APPROVED:

Professor Craig E. Wills, Major Thesis Advisor

Professor Micha Hofri, Head of Department

Abstract

With the increase in traffic on the web, popular web sites get a large number of requests. Servers at these sites are sometimes unable to handle the large number of requests and clients to such sites experience long delays. One approach to overcome this problem is the distribution or replication of content over multiple servers. This approach allows for client requests to be distributed to multiple servers.

Several techniques have been suggested to direct client requests to multiple servers. We discuss these techniques. With this work we hope to study the extent and method of content replication and distribution at web sites. To understand the distribution and replication of content we ran client programs to retrieve headers and bodies of web pages and observed the changes in them over multiple requests. We also hope to understand possible problems that could face clients to such sites due to caching and standardization of newer protocols like HTTP/1.1. The main contribution of this work is to understand the actual implementation of replicated and distributed content on multiple servers and its implication for clients.

Our investigations showed issues with replicated and distributed content and its effects on caching due to incorrect identifiers being send by different servers serving the same content. We were able to identify web sites doing application layer switching mechanisms like DNS and HTTP redirection. Lower layers of switching needed investigation of the HTTP responses from servers, which were hampered by insufficient tags send by servers. We find web sites employ a large amount of distribution of embedded content and its ramifications on HTTP/1.1 need further investigation.

Acknowledgements

The masters thesis has been a unique learning experience, and it will be the highlight of my time here at WPI. Both the gathering and analysis of the data and the writing of this document stressed my ability to stay focussed on a large project. I cannot thank my advisor Craig Wills enough, in getting me to pursue the thesis option, to patiently advise me on the various aspects of the work and finally helping me conclude the thesis with my report. There have been several instances where I thought that any sane being would give up on me, he did not and I thank him from the bottom of my heart for his patience. I thank Prof. David Finkel, who read my results and thesis report in the minimal time it takes to read the documents, being constrained by my deadlines. Thank you for providing me with your comments on the report and the work. I thank Mikhael who provided us with helpful hints and for a couple of his PERL scripts I re-used for my work.

I thank my family for their love, support and prayers through my time here. My parents for encouraging me to pursue a higher education, then supporting me in my decison to take up the thesis option and bearing with me during stages of dissapointment. I would like to thank my brother for being my best critic. The acknowledgements would not be complete without acknowledging all the help and support from friends, family and loved ones. I would like to thank my past and present room mates for bearing with me during my long stints away from home, the professors and students that formed the research group PEDS in the CS department and especially Badri who I relied on for guidance on the approach to research work and the late hours of work on our respective thesis. I believe that I could not have completed this work without the help of everyone mentioned here.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The traffic over the Internet has grown at a fast pace [1]. However, the increase in traffic has caused a disproportionate increase in traffic to popular web sites. A single server at such web sites cannot handle the large loads due to such traffic, which has resulted in servers at such sites being over-loaded and becoming slow. The natural approach to solve the problem is to scale up the number of servers at the web sites. The challenge however lies in being able to distribute the client requests to these servers by a suitable technique.

Some of the approaches suggested by research to distribute the load are to replicate or distribute the content of a site onto multiple servers. Such a collection of servers are called server farms or web clusters and the technique by which the client is directed to one of the servers in a cluster is called load balancing. Load balancing techniques can be broadly classified as transparent and non-transparent load balancing. The basis of transparency is from the client perspective. Sites that do transparent switching do it at a layer below the application layer and hence the response from the server to the client seems to come from the same server the request was made to by the client. Non-transparent switching involves the use of application

layer protocols to direct a client to one of the multiple servers forming a cluster at web sites. Content apart from being replicated, is also divided into smaller sets and distributed over multiple servers. In such cases locality-aware switching is needed where the switching mechanism is aware of the content requested by clients. Servers can also be selected to serve a client based on the Quality of Service (QOS) requirement of the client. We will discuss the various techniques suggested by research in detail in the chapter on related work. One of the aims of this work is to investigate if the techniques suggested by research are implemented by web sites.

The main motivation for our work lies in understanding the correlation between the suggested techniques in research and actual implementations of load balancing techniques by web sites. We were aware of the increase in traffic to popular sites being web users ourselves. There were noticeable delays in accessing popular web sites and sometimes such web sites became intolerably slow after being overwhelmed by the large number of requests. It became evident that a new architecture compared to a single server handling all requests is needed. Research studies provided techniques recommending several methods to achieve scalability. Our primary interest is to identify if web site implements the suggested techniques. To accomplish the identification we run our tests in the client space and make HTTP requests on servers. By studying the responses and investigating web pages we identify sites that did load balancing. Once such sites are identified we are interested in understanding the techniques used by such sites. Based on the techniques used by the web sites we classify the sites and study the implementations of each technique. The study allows us to gauge the pros and cons of each implementation and potential enhancements or issues with a particular form of implementation.

We expected to encounter difficulties in identify sites doing certain forms of transparent switching and also in identifying the exact technique by which the servers are

being selected from a cluster. However an inability to identify these sites would be a reflection on the correctness of the implementation at those sites. In cases where we could identify sites that attempt doing transparent switching it would indicate a loss of transparency to the client.

The other aims of the study are to identify problems with current implementations of load balancing with caching and newer protocols like HTTP/1.1. Clients use tags like the last modified time and ETags to validate caches. These tags are provided by the HTTP/1.1 and HTTP/1.0 protocols and are used by caches on subsequent requests for servers to validate the cache. In previous studies [2] it was found that the tags varied even though the resource remained the same. The change in tag is a potential problem with implementations and it is necessary to identify if the problem is due to multiple servers serving the same content due to replication of data. The problem of varying tags is likely to occur when multiple servers are serving the same content, since the tags vary because of differences in the update time or an improper calculation of the resource identifier tag. Problems found in caching of content from sites doing load balancing needs to be rectified, since it would be difficult to correctly validate the cache contents from such sites. It is interesting to try to understand the interplay of caching, which is provided to reduce traffic load, with methods of load balancing used at web sites. No previous work, to the best of our knowledge, was done in understanding the correlation between caching and load balancing schemes on real web sites. Knowing that maintaining identifiers across multiple servers in a cluster would prove challenging, we hope to test if sites are able to efficiently maintain content identifiers in the new architecture.

HTTP/1.1 provides a method of persistent connections for a client. Persistent connections are provided in HTTP/1.1 so that a client can establish a persistent connection to a web site and retrieve HTML pages with its embedded content on

a single TCP connection. The provision for persistence in HTTP/1.1 is done to speed up the page retrieval as compared to HTTP/1.0, where a page is retrieved by establishing a separation connection for each object. In the initial tests that we ran it was observed that a single page was retrieved by objects from multiple servers. The spread of objects onto multiple servers is the case of distributed content over multiple servers. The problem with this method of load balancing is that the client needs to break the connection to the server and establish a separate connection to a different server. The breakdown and reconnection causes a performance overhead and defeats the purpose of persistent connection in HTTP/1.1. However multiple servers provide an increased parallel processing of client requests to such sites. We feel it necessary to study the distribution of the objects needed to render a page in current implementations of load balanced web sites and hence gain a clearer understanding of the issue. To study the distribution of objects we examine the objects that form a web page and by comparing the servers needed to retrieve these objects we get a measure of the amount of distribution.

To further understand the techniques of load balancing, tests were also conducted to observe if the servers used to serve objects or provide a traversal link to a HTML page changed over multiple retrievals. If different servers are seen in the such tests it would indicate another approach to distribute client request.

The completion of the work allows us to understand the practical implementations of load balancing as compared to research work. It facilitates methods by which future work on web sites can be based in terms of understanding the load balancing techniques. In addition, problems identified enable us to gain insight into problems in practical setup of load balanced web sites. The influences of load balanced web sites on caching and HTTP/1.1 are relevant in investigating corrective actions in future research work. In summary the work allows a deeper understanding of load

balanced web sites and its influences on caching and HTTP/1.1 protocol.

The chapters are arranged with Chapter 2 dealing with techniques of load balancing suggested by research. It covers dynamic load balancing and mirrors. Chapter 3 covers the problems we are trying to uncover and the reasons that motivated us to target the specific areas. Chapter 4 covers our methodology or approach in investigating the specific areas and the details of our experimental setup. Chapter 5 covers the results from our tests while Chapter 6 covers the conclusions, future work and a summary of the thesis.

Chapter 2

Related Work

In this chapter we discuss the related research work done in the area of load balancing. The main areas of concentration in load balancing research has been in developing techniques to efficiently and transparently switch clients to a back-end server and avoid overloading a server. The web content is distributed or replicated over multiple servers. Distribution of content refers to the partitioning of a data set on the back end, while replication refers to replication of data on multiple machines. Redirection of clients at web sites having replicated content requires a fairly uniform distribution of requests to replicated servers to prevent the overloading of a single server. In the case of distributed content, the redirection mechanism must be aware of the content requested by the client and a corresponding transfer of the client must be made to the server providing the content. We begin by discussing the methods by which a client is directed to a server and go on to explain the basis by which a server is selected to serve a client and finally cover other related areas relevant to our work.

2.1 Redirection Techniques on Load Balanced Sites

2.1.1 Overview and Classification of Redirection Techniques

The section covers the different forms of dynamic load balancing techniques. Later sections cover each technique in detail. Figure 2.1 provides an overview of the classification [3] [4] of redirection techniques. There are several methods by which redirection techniques can be classified, based on the layer the switching takes place, the algorithm used in selecting the server etc. We use the layer at which the switching takes place, as the basis of classification. The redirection techniques can be classified as application layer or non-transparent switching and network layer or transparent switching. The network layer is called transparent switching because the client making the request makes just one TCP request and the server at the web site transparently directs the request to different servers in the cluster. The application layer switching is more visible to the client. The client is still dynamically switched to one of the servers in the cluster but it can identify that it is being directed to a different server during the process of retrieving the web page. The non-transparent form of switching will become clearer as we describe the various techniques.

Application layer switching shown in Figure 2.1 has four forms of switching. The Domain Name Server (DNS) based switching uses the DNS server to switch among the server clusters. The DNS server resolves the URL of the site to different IP addresses on a client lookup and hence balancing the load since the client then makes a request to the resolved IP address. The DNS redirection is classified into two types based on the time to live (TTL) value set by the DNS server. Client based switching has the client browser take a more active role in the selection of a server. The client must have knowledge of the servers that form the cluster and it then

selects a server based on information it obtains or discovers. The dispatcher based approach involves a single server at the cluster redirecting the client to one of the cluster servers. The dispatcher that dispatches the client to a server in the cluster can do so transparently, as well as non-transparently. The form of dispatcher based redirection that falls under the non-transparent switching is the packet forwarding scheme of HTTP redirection. The HTTP redirection method uses a 302 response code to the client to indicate the server has moved. The fourth form of redirection under the application layer classification is the HTTP form of redirection in the server based redirection scheme. The scheme gives every server in the cluster the ability to redirect the client to any other server. Similar to the dispatcher based scheme each server can redirect the client to a different server transaparently as well as non-transparently.

The Network layer and MAC layer switching happens at a layer lower than the application layer and hence the client does not know that it is being redirected. The two forms of redirection schemes are dispatcher and server based redirection schemes. As described earlier the dispatcher forwards the client requests since every request is send to it. The decisons are based on statistics on server load gathered by the dispatcher. Since the server based redirection allows every server to participate in the redirection scheme each server is aware of every other servers load condition. The different forms of dispatcher based redirection are packet single rewrite, packet double rewrite and packet forwarding as in Figure 2.1. The packet single rewrite has the dispatcher rewrite the destination of the TCP packet to the IP of one of the servers in the cluster, with the cluster server responding to the client directly. The packet double rewrite has the dispatcher write both the incoming packet from the client to the cluster server and the outgoing packet from the server to the client. Packet forwarding does not involve the rewriting of packets as in the previous two

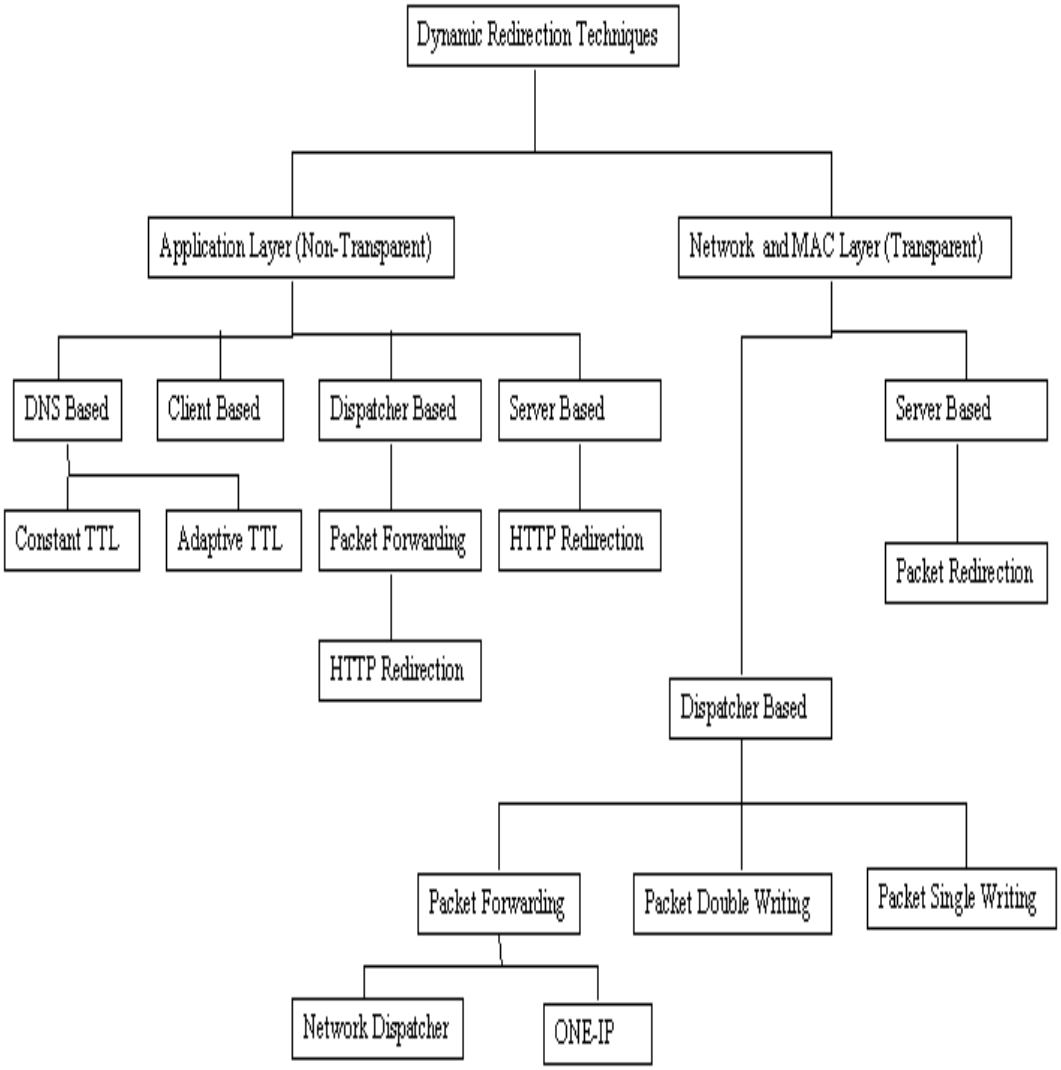

Figure 2.1: Redirection Technique Classification

cases. The network dispatcher form of packet forwarding involves the dispatcher doing a MAC layer broadcast to the servers and hashing the client to one of the MAC addresses. The ONE-IP approach involves all servers in the cluster having a common secondary IP address and the dispatcher forwards the client request based on the primary address or by doing a broadcast on the LAN, while the servers respond using the secondary IP address.

The server based packet redirection scheme rewrites packets as in the packet single rewrite or packet double rewrite described above. The difference here is that every server in the cluster has the ability to redirect the packet to any other server. The following section goes into the details of each scheme.

2.1.2 Client Based Redirection

Client based redirection scheme shown in Figure 2.1 as the second type under the Application layer switching involves the use of web client or client side proxy in deciding the server to go to in a replicated web server architecture. The client must be aware of the structure of the server architecture, to select a node to make a request.

One of the methods that Netscape Communication implements is by making the Netscape Navigator browser select a random number i between 1 and the number of servers. It then makes a request to that particular node wwwi.netscape.com[5]. This is not implement able for all corporate intranets and is hence not considered a useful solution.

The other method suggested involve using applets having information about the various nodes at the web site. The basic idea lies in extending the web browser by a downloadable applet. The applet being aware of the individual servers that comprise the service would select the fastest server based on network load [6]. Another approach with the applet based scheme is by extending the server by a servlet in addition to downloading an applet. The servlet then sends a list of URLs having the replicated resource available. the client applet then selects one of the URLs and makes a request for the resource [7].

Client side proxies are also used for load balancing where instead of a client making a request the proxy makes a request to the replicated server. This however needs

Internet component modification since the proxy must intercept all client requests and make requests to servers as well as respond to clients while maintaining a TCP connection to both ends. The advantages of the client based redirection schemes are that they push the load of server selection on to the client. The disadvantage lies in the processing and transactions needed by the server in providing the client with the server architecture at a web site and the latency in making a final request to the selected server and retrieving a web page from the selected server.

2.1.3 DNS Based Redirection

The cluster Domain Name Server (DNS) maps the site name or Uniform Resource Locator (URL) to the servers IP address. The DNS based redirection scheme[3] achieves load balancing by mapping the URL to the desired server nodes IP address. This scheme achieves transparency at the URL level by providing a virtual interface to the outside world. Some of the constraints in this approach is the fact that the URL to IP mapping by the DNS server is often cached by the intermediate name servers or by the browsers local cache. To overcome the constraint the DNS server often sets a Time to Live (TTL) value, which sets the time for which the name servers holds a valid address mapping for an entry. At the end of the TTL period the address mapping is again passed on to the cluster DNS. The setting of TTLs to expire DNS entries however does not consider non-cooperative name servers, which ignore small TTL values.

Figure 2.2 shows the client making a request to the DNS server to resolve a URL. The name servers between the DNS server and the client cache the resolution. The DNS server is updated with the server states to make decisions to redirect the client based on the server states. The client then makes a request to the server IP the DNS server returns on a query. The DNS based redirection scheme is classified based

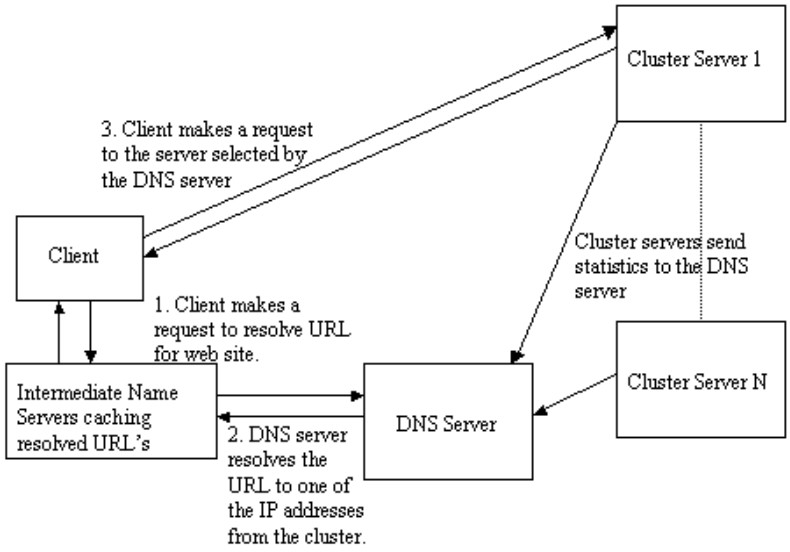

Figure 2.2: DNS Based Redirection

on the constant or adaptive TTL values set by the DNS server. The DNS based redirection scheme is advantageous since the clients need to resolve the URL of a web site before making HTTP requests and the scheme does not incur the latency due to additional transactions. The scheme can however be hampered by caching of mappings at name servers, which can be mitigated by the methods described below.

Constant TTL Algorithms

Constant TTL algorithms set the TTL values based on either no state information, a combination of client load and its location or a combination of server load and its location. The stateless algorithm uses a round robin scheme where the DNS maps the server nodes by a round robin algorithm. This scheme fails in avoiding excessive loads from client requests on a server coming from a particular client domain due to caching of the address mapping. Clients could make requests to servers that are not operational based on cached mappings, which is undesirable. Server state based algorithms require the DNS to maintain the load conditions on each server and assign

lower TTL values for heavily loaded server nodes. Client state based algorithms use the statistic of the requests coming from certain domains and assign a weighted load index [8] for certain domains. Based on the load index values different server round robin chains may be assigned. The TTL values are assigned such that a lower TTL value is set for domains making large requests. Ciscos distributed director [9] uses the proximity of the client to the server and the latency between the server and the client is calculated by evaluating the clients location by its IP address. Schemes using both client state and server state to set TTL values[8] are also used.

Adaptive TTL Algorithms

Adaptive TTL algorithms use some server and client state based DNS policy to select a server and dynamically adjust the TTL values. The TTL values are changed for each address request. The adaptive algorithm maintains a hidden load weight index similar to the constant client state based algorithm. The adaptive algorithm is a two step process [3]. In the first step it selects a server to service a request based on load weights assigned to the client domain. All subsequent request are adaptively modified such that TTL values for domains making a large number of request are reduced.

2.1.4 Dispatcher Based Approach

The dispatcher based approach implements redirection at the IP level. The IP level redirection is more transparent and hence hides the load balancing implementation at web sites as compared to the DNS based redirection. The dispatcher in this approach is assigned a single virtual IP address. The URL is mapped to this IP address and hence all packets arrive at the dispatcher. The redirection can happen at different levels of the TCP stack and the different dispatching mechanisms are

classified on that basis. The dispatcher selects a server node based on various simple algorithms such that the load is balanced evenly. The algorithms are simple in nature to help minimize request processing. The various methods in the dispatcher based architecture are explained below.

Packet Single-Rewriting

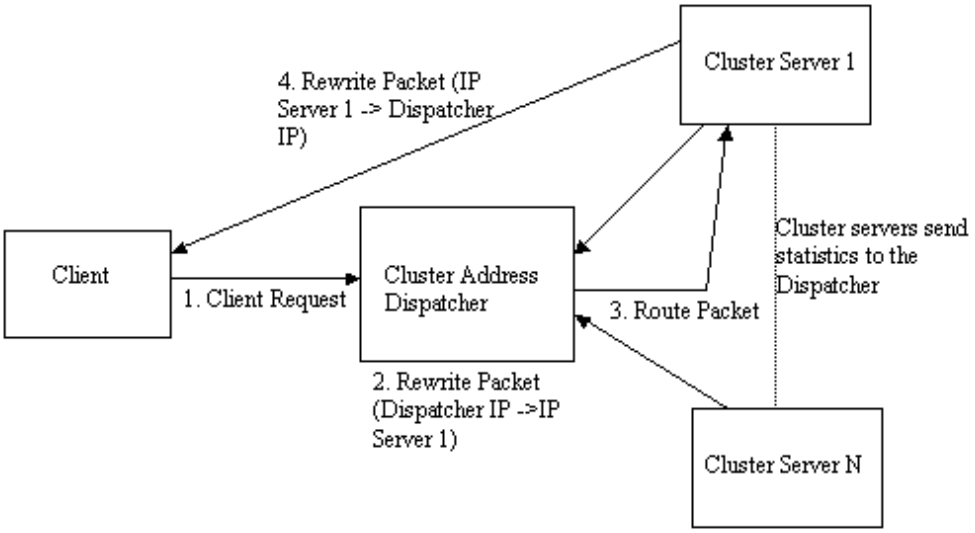

Figure 2.3: Dispatcher Based Redirection: Packet Single Rewriting

Figure 2.3 shows one of the methods by which a dispatcher routes a packet to the server nodes. The dispatcher does the routing by rewriting the IP address in the packet header. The dispatcher behaves in a manner similar to an edge router. It changes the packet IP address from its own virtual IP address to that of the selected destination server. The TCP router maintains a table of all established connections so that a client is always directed to the same server. The selected server must then replace its IP address with the virtual IP address before sending the response packets. In this manner the client is transparent as to which server is serving the request. This technique has high availability since a failed server can

be removed from the table. However since all the processing happens through the single dispatcher it could potentially become a bottleneck.

Packet Double-Writing

The packet double writing uses a dispatcher to schedule and control client requests. It differs from the packet single rewriting in that the dispatcher changes the address in the response packets as well as the client requests. In this manner the dispatcher as shown in Figure 2.4 performs both functions of rewriting the IP in the headers in the request as well as the response packet as compared to the single write in the earlier case. This technique increases the possibility of a bottleneck at the dispatcher. The server node can be selected on the basis of the least loaded server, round robin mechanism or in a random manner. The local director[10] and the magicrouter[11] architecture of Cisco uses this mechanism.

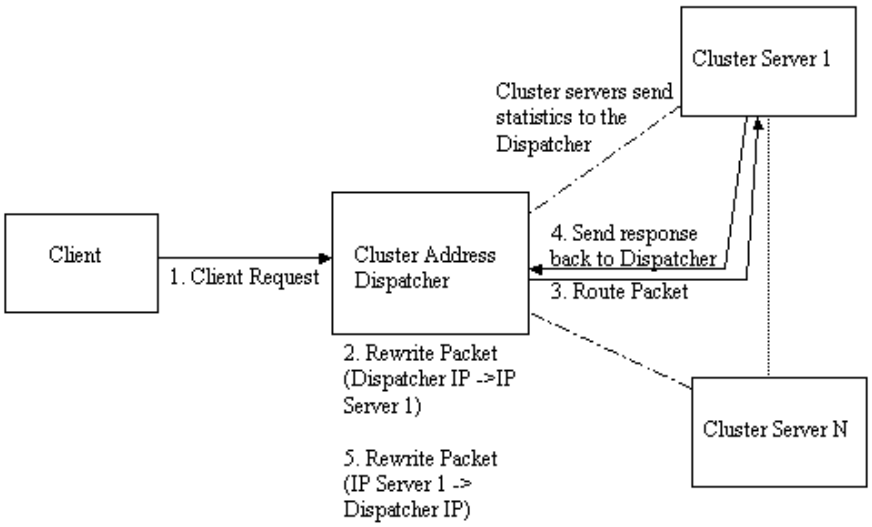

Figure 2.4: Dispatcher Based Redirection: Packet Double Rewriting

Both the packet single rewriting and the packet double writing schemes require the TCP stack to be modified which involves modifying the kernel of the dispatcher.

Packet Forwarding

Packet forwarding approach involves forwarding the packets to the servers without rewriting the address. The two mechanisms this is done by are described below.

Network Dispatcher In the network dispatcher approach the servers on the cluster LAN share the same virtual IP address. The ARP mechanism is switched off in the server nodes and the dispatcher distributes the client requests based on the MAC address of the server node. The architecture in this case could also be hierarchical with the dispatcher forwarding all packets to a second level dispatcher. The techniques requires the server to be on the same LAN if a single dispatcher is used. Several second level dispatcher can allow for a WAN architecture with the first dispatcher rewriting the IP address by packet rewrites, as in the first two dispatcher techniques described above.

ONE-IP Address The ONE-IP [12] approach requires the dispatchers IP address to be configured as the secondary address of the server nodes. The servers set the secondary address as the virtual IP address by the ifconfig alias option in Unix. The ifconfig alias approach command allows the user to specify multiple IP addresses on an interface. The dispatcher sends the client requests based on the secondary address. A hash function is applied on the client IP address to determine the server node which will server the client requests. This eliminates the need to maintain a table for each connection. However since the distribution is static the loads on the servers could be skewed.

The two methods by which the ONE-IP switching is done is the routing based approach and the broadcast based approach.

In the routing based approach the client request are forwarded to a dispatcher by

the router. The dispatcher does a hash function on the client request and selects a server and forwards the client requests based on the primary address of the server. The server node receives the client requests and responds directly to the client using its secondary address.

In the broadcast based dispatching scheme the router broadcasts the requests to all the server nodes. The server nodes then apply a hash function on the client IP address. A filter is applied so that just a single machine accepts the client requests. The server node then responds like a normal web server.

The advantage of the network and ONE-IP based dispatching is that the dispatcher does not have to rewrite packets as in the packet rewrite schemes. The mechanisms are transparent to the client and the algorithms used to redirect clients are simple hash based schemes. The disadvantage lies in the requirement that all servers are required to be in the same LAN and cannot be distributed in geographically different locations.

HTTP Redirection The HTTP redirection scheme as shown in Figure 2.5 uses the HTTP 302 response code provided for in the HTTP specifications[13]. The HTTP 302 response code was provided to indicate that a server has moved temporarily to a new URL. Load balancing schemes use the provision to avoid modifying the IP address of the packet. The redirected location is specified in the "Location" field of the HTTP header. The client must then make a request to the location specified to get the web page. The requests are directed to a server based on the server state or the location of the client.

One of the disadvantages of this approach is the increased response time perceived by the client since the client has to make a second request to the redirected site. The advantage of the technique is that it is done at the application layer and

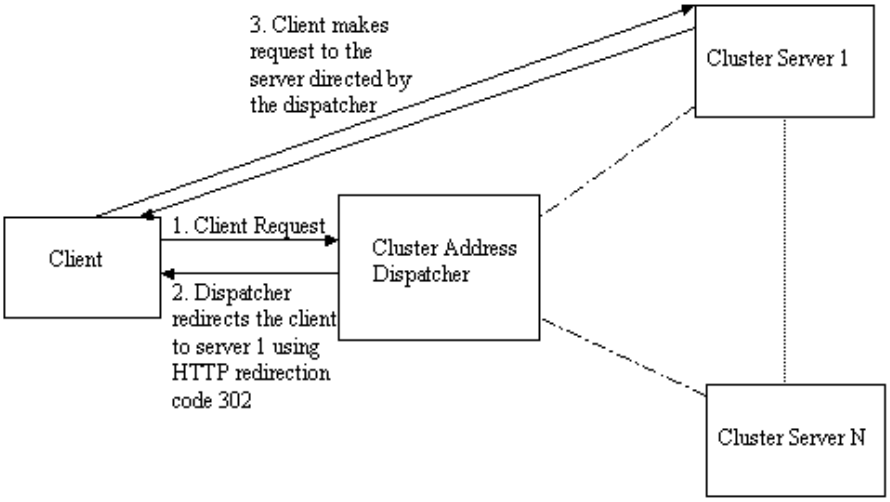

Figure 2.5: Dispatcher Based Redirection: HTTP

hence the TCP/IP stack can remain unchanged unlike the previous cases.

2.1.5 Server Based Approach

The server based approach uses multiple levels of redirection. The primary DNS of the web system assigns the client request to a web server node. Each server may then redirect the request to any other web server. The technique allows every server to participate in the load balancing scheme and it is an effective redirection scheme if content is distributed such that only a small set of servers serve certain content. The two mechanisms by which the server node redirects the client request to another node is as below.

HTTP Redirection

In this scheme[14] the second level of redirection is done by responding with a HTTP 302 response. The redirection mechanism may be done if the server finds

that another server in the cluster is more lightly loaded or is closer to the client. The mechanism can also be used in a distributed content architecture where the content can be served by a particular server node. [15] uses the multiple level of redirection. The DNS server in the architecture round robins the request or can make an intelligent decision based on the servers sending in the server states. However the state information results in increased traffic on the intranet.

Packet Redirection

Packet redirection uses a packet rewriting mechanism to direct a request to a different server. This mechanism is transparent to the client and can use hash function on the client IP address or base it on the servers current load state to select a server.

The advantages and disadvantages of HTTP redirection and Packet based redirection are similar to the same schemes uner the dispatcher based approach. The advantage of the server based approach is the scalability and granularity at which the redirection can be made. The drawbacks are due to the multiple levels of redirection, which results in user perceived delays.

2.2 Other Redirection Mechanisms

In the sections above we detailed the various solutions suggested in research for redirection schemes. Research has also been done to effectively combine two or more techniques to achieve a better load balancing strategy. A popular scheme as shown in Figure 2.6 has been using the DNS based redirection combined with HTTP redirection. The SWEB[15] server uses the DNS server to redirect request in a round-robin manner. Once the targeted web server receives the request it decides whether to process the request or forward it based on the server loads. Modules running

on each server maintains CPU, disk loads and network traffic information for each server in a table. The server chosen by the DNS uses the table to redirect the request using the HTTP 302 redirection code. [16] uses a similar mechanism to SWEB. It proposes two schemes. One where the DNS server has some state information on the server loads and intelligently redirects the client to a suitable server. Each web server in the cluster also has information on current client connections and the mechanism to redirect clients to other servers using the HTTP redirection scheme. Second an approach where the DNS server redirects using a round robin scheme wihtout any state information. The second paper [16] extends the scheme to allow servers to drop in and out of the cluster following which the DNS server changes its available server list. The reason for the dual mechanism is to overcome the individual drawbacks of each approach. The DNS redirection method suffers from the URL to IP name caching by intermediate name servers and can effectively reduce the control the DNS server has over the load balancing. The HTTP redirection scheme has ingrained in it the additional latency due to the client needing to make the additional HTTP request.

With web content becoming increasingly complex with multimedia and dynamic services, the need to maintain quality of service (QOS) becomes important. Several papers have worked towards achieving QOS requirements for users. A modification of the dual method of DNS and HTTP redirection method is described in [17]. Here increased granularity in selecting the web server is got by having a dispatcher in between the DNS and the cluster with servers doing HTTP redirection. Similarly [18] uses a simple architecture using DNS and dispatcher based redirection schemes based on geographically distributed, as well as local clusters. It defines SLAs or service level agreements between the client and server before a connection is established. The server selection is done based on servers that can fulfill the service

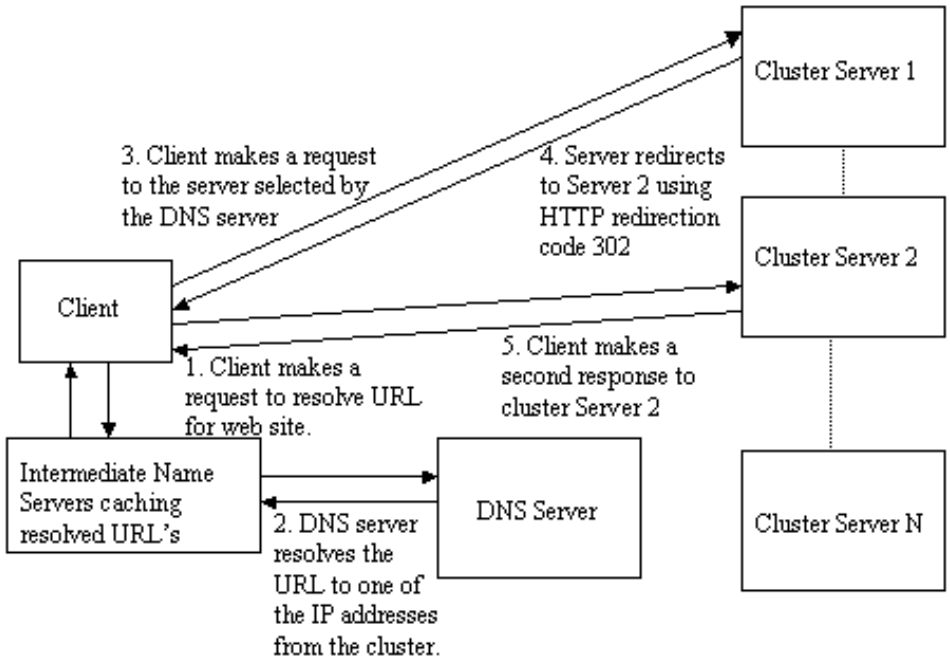

Figure 2.6: Combination of DNS and HTTP Redirection

requirements of the client.

More radical changes in server selection is done by other research which recommends modifying the architecture and conventional client access to achieve QOS requirements. [19] uses the anycast paradigm to select the most efficient server. The methodology uses a DNS form of hierarchy among anycast resolvers. Anycast resolvers map a domain name to an IP based on the metrics the resolver stores. The list of available servers are maintained at the resolver by service location protocols [20]. [21] basis its method on modifying the diffserv capable internet. A diffserv capable network consists of domains with the intelligence of setting QOS parameter on the ingress port of the edge router. The setup to achieve QOS is by having a resolver in the client domain and a server agent maintaining all the server states. The connection is established by the client by sending the request to the resolver and the resolver, through the server agent, selecting a server and reserving the QOS re-

quirement along the path to the server. QOS based server selection is more complex and requires a more involved setup as recommended by current research.

2.3 Methods and Factors Used in Server Selection

In the previous section, we explained the various techniques used to redirect a client to the server that can serve the client efficiently. We elaborate the basis on which a particular server is selected from a cluster here and the various parameters used in selecting a server. The selection of a server should be made such that no single server is overloaded. Servers can be selected randomly or based on the servers proximity to the client or a server that is least loaded. A good algorithm must strike a balance between a server that can serve the web page the fastest under no load, due to its proximity and other similar factors and a server that is least loaded.

Selecting a server can be based on an algorithm that can be based on different metrics got from the client or the server. On the other hand there are several methods in which the server is selected without using such metrics. Some of them are

Random Selection - The server is selected randomly based on the content requested as the seed to the random function.

Round Robin - The element selecting a server cycles client requests through a list of available servers. Variation of the round robin scheme are probabilistic round robin, weighted round robin (weighted based on load, proximity to client etc.) etc. Round robin schemes use the parameters described below to assign weights to each server and allow those factors to come into play during the selection.

The techniques described may help achieve efficient load balancing but most algorithms proposed use factors that have a direct bearing on server response times. All factors described below can be gathered proactively or reactively i.e. they can be gathered before a client request is made in a periodic manner or can be got on receiving a client request. The factors can be broadly put into three categories.

Least Loaded Server - Knowledge of the server state is important to know which servers are not available due to congestion or faults. The aim in using these set of parameters is to direct the client to a server that is least loaded and hence can serve the client the fastest. The parameter used are the number of active connections, disk load, disk I/O channels, CPU idle or busy states, load averages based on number of processes running on the system, etc. Using one or a combination of these factors a decision is made by the dispatcher.

Network Topology - Network topology is a crucial component to server selection decisions. [22] does a detailed study of network topology based factors used to make selections among replicated servers. Network factors can be got at the application or routing layer. At the routing layer the information can be gathered incrementally based on route updates etc. However this information is not easily available to the application layer. On the other hand application layer information is got at a higher network cost due to probes and other queries made.

Routing layer information gathering happens using multicast or anycast protocols where the client broadcasts, multicasts or anycasts the request and the fastest server response is chosen. Anycast is preferred since it allows the

client to query only a small amount of servers to get the information.

Among the more common factors in application layer gathering, are the hop count from the server to the client got through functions like trace-route and round trip times using pings. Hop counts are however preferred because of poor clock resolutions associated with pings. Redirection based on the client domains requesting documents, so that they can be mapped to servers close to their domain is also frequently used. Time dependent models have also been proposed where the client location and the time at the location is used to direct the client. This method is based on varying client requests coming from different regions based on the place and time there. Also Autonomous system (AS) [23] path routing information of the Border Gateway Protocol(BGP) is also suggested to evaluate logical distance of the network.

In addition to the server-to-client network factors, in-site network factors are also used to monitor the state of the cluster. To do so collisions, packet errors and traffic are measured. Network factors can be arbitrarily expanded to consider carrier fees, routing policies etc. as well.

QOS and Content - With quality of service and location aware distribution being studied, server selection may have to be based mainly on these factors regardless of the ones mentioned above. However multiple machines offering the same QOS or content can allow the use of the factors mentioned above. QOS based server selection[24] involves the client probing different servers to see which one meets its client requirements. Since the QOS requirement translates to the entire path from the server to the client; for effective service, elements along the path must be QOS enabled as well. Location aware distribution tries to achieve high locality to the back-end caches. The

aim to achieve localization of content is to allow for faster response from the server, since the server can serve the content from its cache much faster. The scheme allows every server in the cluster to serve the request as well as the server designated to serve a particular content. Sites where content is exclusively distributed, only a certain number of servers are selected to serve a particular content. The fewer number of servers available to serve a particular content, sets more limitations on the choice of nodes available to serve content. With increasing multimedia traffic and large databases, QOS and content based server selection play an important role.

2.4 Content Aware Request Distribution

Content aware request distribution involves the distribution of requests at web sites based on the content requested by the client. First proposed in [25] it recommended using the technique when the data content of the entire server exceeded the memory cache of one node in a cluster. The solution suggested was to partition the content of the database and distribute it across servers and then have a content aware switch that would strip the client request, detect the content requested and forward it to the appropriate server as shown in Figure 2.7. The technique could also be used to employ servers designed to serve special requests like audio and video.

The setup in content aware clusters is such that each server can serve the complete content of the web site. The content aware distributor partitions the content based on algorithms using hash functions on the URL of the contents, server states and other parameters explained in the load distribution algorithm section above. One of the challenges a locality aware algorithm is to not have a large number of

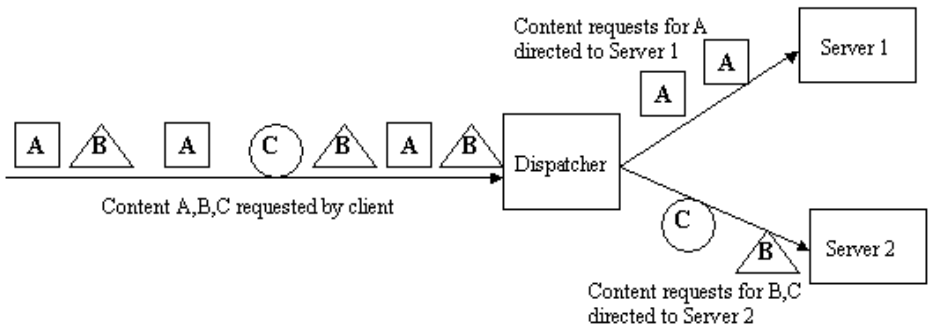

Figure 2.7: Content Aware Request Distribution

requests directed to a single node. The increased load on a single node would defeat load balancing strategies. Hence distributors maintain load characteristics of the server and distribute client requests for popular content on to servers that are not heavily loaded. Since every server is capable of handling the entire content it would involve content being pushed out in memory and replaced by popular content. The method by which the request is directed can be using one of the schemes described earlier. While [25] recommends using TCP handoff to the back end server, [26] uses packet rewriting at the dispatcher. FLEX [27] was designed as a scalable server to do content aware switching and it pushes the dispatcher functionality to the DNS. In their architecture the DNS using their designed algorithm does the content hashing of a web site and also maintains the access rates for each content so that no single server is loaded. [28] elaborates the algorithms used by the FLEX server to replicate popular content across all server and the caching policies like least recently used (LRU) to achieve better performance.

2.5 Mirrors

In understanding methods used to solve the problem with rising Internet traffic at web sites we studied methods where clients are directed to servers best suited to handle client needs dynamically. By dynamic we mean the architecture of the server cluster is such that it directs the client request to the appropriate server without the user having to manually select a server. Mirrors take a more direct approach to client redirection. Mirrors are servers, which have replicated content across several servers. The user is displayed a web site, listing all the available servers with parameters of interest that they can use to select a server. Typically the parameter involves the geographical location of the server. The user can then select a server based on their proximity to the server. Since the process is not dynamic and is divided into two steps with user interaction required it is much slower than the dynamic techniques. The advantage of the method however is in the way the servers at different geographical location can format the content based upon the region.

Research work has been done on the performance as well as techniques to place mirrors on the Internet. [29] uses a slightly different definition of mirrors where they define mirroring as any replication of content across servers. The research involves trying to understand the amount of replication by getting web pages from a large list of urls and analyzing the amount of replication of content between similar pages. For our purposes we stick to our previous definition of mirrors where the user has to manually select a server from a replicated set. [30] studies different algorithms to decide on the number and effective placement of the mirrors on the web. The results show that beyond a number, increasing the number of mirrors placed does not significantly increase the performance perceived by the

client. The paper arrives at an algorithm to decide on the number and placement of mirrors. [31] studies performance characteristics of mirrors. Their study noticed large differences in performance between mirrors of a web site and they recommend that clients use a small set of the total servers mirrored to decide on the server to use for page retrieval. The popularity of a document was also found to be independent of the server selected. Mirroring has not seen the depth of research seen in dynamic forms of directing clients transparently to a server. The method also entails the inherent delay in the time for the user to select a server.

2.6 Related Areas in Load Balancing

In the previous section we discussed the various techniques adopted by servers to distribute client requests to multiple servers. Although most of the work in load balancing has been concentrated on accomplishing the switching efficiently. A few related issues have been looked into. Some of these issues are particularly relevant to our work and we shall discuss them here.

2.6.1 HTTP/1.1

A significant drawback to HTTP/1.0 [32] is the multiple connections it establishes. Web sites typically have a large number of embedded images and HTTP/1.0 breaks down and opens a new connection for each object, which causes significant CPU processing on the server. With multiple connections implemented on popular browsers this drawback is exacerbated. HTTP/1.1 [13] is the RFC standardized protocol to resolve this issue. HTTP/1.1 provides for persistent connections which allows for clients to make multiple requests on a single connection. Also the protocol allows pipelining of requests where multiple HTTP request can be send

before receiving a response to an earlier request. HTTP/1.1 hopes to improve performance and several studies have been done to understand if the design goals of HTTP/1.1 have been achieved.

[33] suggests several policies to manage TCP connections by servers with persistent connections of HTTP/1.1. It aims to change the way servers break down connections with holding times and LRU policies. [34] and [35] both compare performance of HTTP/1.0, HTTP/1.1 and pipelined HTTP/1.1. [34] studies the performance on three popular servers and details TCP performance data. [35] studies the performance from client sites around the world to seven hundred popular web sites. Both studies confirmed that HTTP/1.1 performs better than HTTP/1.0 even when multiple HTTP/1.0 sessions are present. Pipelined HTTP/1.1 had significant performance and end to end timing improvements over both methods.

Studies at Rice University [36] were done related to effects of persistent HTTP and cluster of servers. It mainly compares different load balancing techniques and provides data to suggest that the back-end TCP handoff mechanism is superior. Here the server receiving the client connection may not be able to service the request if content is distributed and the server must handoff the TCP connection to the other servers. The problem with HTTP/1.1 and clusters is that the load balancing mechanism must work at the granularity of a single TCP connection. Tearing down of connections and establishing a new connection to an appropriate server would cause unwanted delays for the user.

2.6.2 Caching

Caching involves the storing of frequently used data to prevent the delay in requesting for the data during subsequent requests. Caching on the web happens

at various levels and is mainly distinguished as client side or server side caching based on the proximity of the cache to the server or client. Server side caching involves the cache from the memory of the server or from a co-operative server closer to the client. Client side caching can be as close as the browser on the client or the proxy caches on the intranet or service providers.

Caching policies[37] involves the algorithms that are used to determine which content remains in cache and which are removed and are aimed at retaining content that is most likely to be reused. This policy is significant in reducing loads on servers by reuse of content which havent changed. It is of particular interest in the context of load balancing and server clusters since both techniques must act to complement each other. We shall not delve into caching policies and architectures used to provide efficient caching but focus on relevant work to server clusters and load balancing. [2] studies various web sites to understand the changes of content on popular web sites. The study gathers header information and the possible implication of changing content to caches and servers. A study [38] correlating server side caching and clusters have shown that a combination of state information of each server and design of algorithms to get high server cache hit rate is required. Analysis was done on factors that affect good load balancing and caching techniques. Caching plays an important part to reduce load on web servers and hence becomes an important factor to consider in replicating and distributing content on the web.

2.7 Summary

We discussed the various techniques used for relaying a client to a back-end server dynamically. Broadly the techniques are classified as transparent and

non-transparent techniques based on the layer the switching takes place. The different techniques use the network elements along the path of a web page retrieval process to redirect a client. Hence we see the client, the DNS server, dispatcher in front of the cluster or the servers that form the cluster make the redirection decison.

Due to advantages and disadvantages of these techniques we find a combination of the techniques mentioned above employed. Newer forms of load balancing techniques use content aware and QOS based server selection in addition to selecting a server that can serve the client the fastest. Finally we discussed related areas in load balancing relevant to our work. The HTTP protocol and its newer version with the provision of persistence and pipelining and studies investigating the effectiveness of the design goals were mentioned. Caching allows clients from re-requesting content that have not changed and hence plays an important role in reducing server loads.

The Approach chapter describes our motivation to investigate load balanced sites. We discuss some of our preliminary tests and our classification scheme for the list of web sites we test. We further detail the approach we take to investigate each of our classified set of servers.

Chapter 3

Approach

In the previous chapter, we discussed research work done in areas close to our work. In this chapter we shall discuss the motivations for our work and the approach we took to investigate the issues surrounding our research. We shall initially discuss the motivations for our work, the initial tests we ran and the methodology we adopted based on the results we got.

In the beginning, as we researched several papers on load balancing and content replication, we became aware of this new area in web architecture. Being web users ourselves we were familiar with the delay in getting to popular web sites and resources online. To scale up to the increase in client requests web sites use multiple servers to serve web pages. With sites using multiple servers to scale up to client requests, research was done to propose and study alternate and newer architectures to load balance servers. The solution on the server side involved replicating and distributing content over multiple servers. The aim of replicating and distributing content is to distribute client requests to multiple servers and hence distribute the load.

Accompanying such work brings in challenges of maintaining content consistency

across servers and providing an efficient method to redirect clients without delaying the user response time. Using the different schemes presented we could have done a comparison of which architecture is better. Alternatively we could have tried to arrive at our own unique approach to improve load balancing. Since several architectures were studied and proposed by research, we decided to investigate the techniques used by popular web sites. In addition the aim was to identify sites doing load balancing and if the techniques employed by such sites matched the techniques suggested in research. In the world wide web we had all the elements needed on the server side with popular web sites having real clients and a large amount of traffic. Servers at such sites would need to employ some mechanism to handle the large number of requests. The aim of our approach became to try to act as clients and try to identify among popular web sites, sites doing some form of load balancing. To identify such sites, elements in the chain of getting a web page need to be investigated. These elements would be the DNS and HTTP servers. The content being passed by them needs to be analyzed to find hints of multiple servers serving content.

Without access to the servers at such sites or knowledge of what techniques are being used we are constrained to analyzing data that is sent to us in response to HTTP requests and DNS resolutions. The constraint however puts us in the same domain in which typical web users would find themselves in and allows us to study the web server from the point of view of the web user. The techniques like HTTP-based switching and DNS-based redirection fall into categories which can be detected. We can detect such techniques because querying DNS servers and looking at HTTP return headers, are accessible methods to us.

We are limited to data at the application layer and techniques like the dispatcher mechanism using packet forwarding or the One-IP approach, which use the

network layer to switch clients would be transparent to us, in compliance with the design of the redirection technique. Such sites that use network layer switching intend to keep the user transparent to the process by which content is served from multiple servers. Our inability to find such sites would imply a good implementation of the switching, if in fact the web site was using the technique. The interesting case would be with sites providing content from a single IP address with differences in the IP, HTTP headers or web page content. Such sites, although intending to be transparent, allow clients to identify the differences on retrievals and potentially the differences could prove to hamper the aim of reducing load at such sites. The differences in content would point to an implementation that is not entirely true to its design aims.
In addition to the aim of finding sites that do load balancing, we were also targeting areas which are used as alternative methods in reducing load at web servers.

3.1 HTTP/1.1 and Load Balancing

HTTP/1.0 was designed for the early days on the web where the number of users was low. Clients have to make individual request for each object, hence a client needs to make a request for the web page followed by each embedded object within the page. This approach results in multiple TCP connection and consequently increases processing and response time to the user. HTTP/1.1 was designed to correct this situation by providing for persistent HTTP connections. Persistent HTTP allows users to make requests sequentially over one TCP connection. Clients are also allowed to pipeline requests without having to wait for a response to a previous request.

Studies detailed in the related work chapter have shown that HTTP/1.1 performs better than HTTP/1.0 and HTTP/1.1 with pipelining provides a much better performance than that of HTTP/1.0. Although pipelining is not always supported on all HTTP/1.1 servers, it is important that load balancing techniques aim to ensure that this protocol works in complement to the cluster architecture, in helping reduce loads at servers. The main aim is to allow a client to set up a TCP connection and maintain it. After looking at various load balancing techniques proposed, we found that some techniques could potentially defeat the client maintaining a persistent connection.

Techniques like the dispatcher-based packet redirection where the TCP connection is handed off allow users to maintain a single TCP connection through HTTP transactions. However techniques that did HTTP redirection or distributed embedded content like images across different servers cause the client to break an active connection and establish additional connections. Sites that distribute embedded content over several servers work against the design intent of HTTP/1.1, which is a cause for concern.

3.2 Caching and Load Balancing

Caching can be of two forms, server side and client side caching. Server side caching is to allow servers to move frequently accessed content on to servers closer to the clients. Such servers store and respond to clients making requests for the content. For our purpose we look at client side caching. Client side caching is the process by which clients store content they have accessed, in the hope that a subsequent request for the same content would allow the client to serve it out of the local storage. This storage may be in the form of the local disk or a machine

close to the client (typically on the same LAN).

There are several algorithms by which clients decide to keep and remove content from the storage to maximize content reuse or cache hits. To identify content which is not stale, the client requests the web server for object in cache with the relevant tags attached to the request. The servers then use ETags, Last modified times (LModT) or other header tags to validate the object in cache. ETags are unique identifiers to the content on a web site and the last modified time is the last time the server changed the content. These tags are used to validate cache content. On a client request for a cached content the cache makes a request to the server with the unique tags corresponding to the object. To avoid sending the entire response, the server responds with a HTTP code 304, indicating the content is not modified. On changes to the content the server should send a complete response with HTTP code 200 and newer tags. Similarly the ETag and last modified time remaining the same would indicate that the objects present in the client cache are still valid, verified by the 304 response from the server and the content can be re-used instead of retrieving them again. This process reduces loads on the server by not requiring them to send all the objects forming a web page to a client caching such objects.

Load balancing results in content being replicated and distributed across several machines. Web sites store the same content across all servers in a cluster. If the unique tags in the header of such content do not remain consistent it defeats caching. Clients have to retrieve the entire content even though the content has not changed. In our methodology we filter out relevant tags from HTTP responses to identify if the issue exists.

3.3 Initial Tests

To sanity test our approach we set up a preliminary test to observe if we can identify at least some form of load balancing. The test was done as a proof of concept to our research proposal. The tests were done in September 1999 and the web sites used were different from the ones finally used. The results from the tests allowed us to formulate the final test suite described in future sections. The final test suite was run twice in April, 2000 and in May, 2002.

The early tests were on a list of 122 most popular sites rated by www.100hot.com[39]. To get data on each web site we ran client requests on the web-sites using the "GET / HTTP/1.0" request. We resolved the URL of the web site and then studied the number of IP addresses mapped to the URL. The returned header on the packet returned by the server on the GET request was also studied. The "GET / HTTP/1.0" request is used by HTTP clients to request for the root HTML page of the URL.

We identified at least 33 sites using some form of load balancing. We saw 17 instances of HTTP redirection using return code 302 to direct the client to a different location. On checking the locations returned by the server we confirmed one site using HTTP redirection to direct the client to a different server each time. Further investigation and tests were required to investigate the remaining servers to see if the redirected site do load balancing in some form. The scripts to test the redirected sites were written in the final test suite.

The remaining instances were of servers with a single IP address and showed no indication of load balancing techniques. Further investigation was required in the cases where the load-balanced servers could have been using transparent routing. To find sites doing transparent switching, we needed to identify if there are

multiple servers behind a single server. Entity Tags was decided as one possible method that could be used to detect transparent routing. In a previous study it was found that the ETags of some resources were changing even when the resources remain the same [2]. We planned to study the sites to see if this effect is due to resources being obtained from different servers and we wrote test scripts that extracted content of interest from the web pages. Some of the other possible methods to detect transparent routing was to look at the headers from the various servers and look for changes in the information returned by the servers over multiple requests.

Thus with our initial tests we were able to ascertain that it was possible to at least detect some form of load balancing and it provided us with the grounds for further tests and analysis.

3.4 Test Methodology

In this section we list out the tests we conduct and the motivations behind having them. We divide our tests into two parts with data collection and then the analysis.

The general process of getting a web page given a URL is as follows.

1. Retrieve the IP address associated with the URL through a DNS.

2. Establish a TCP connection to the IP address on port 80, which is the standard HTTP port.

3. Make a GET request with the web page path in the request e.g. "GET /URL HTTP/1.0", where URL is the complete path to the resource being requested. The HTTP 1.0 indicates the 1.0 protocol, to make an HTTP/1.1

request we replace the 1.0 with 1.1. We conducted all our tests with the HTTP 1.0 requests.

4. We receive the response into a buffer until the entire web page is received. The response includes the HTTP headers and body. The HTTP header has the various tags which are not displayed on browsers. The browser interprets the HTML tags if present and displays the web page. For our test purposes we wrote a HTTP client which made the GET requests and stored the responses in the server. The responses were stored as received and hence provided us with the HTTP headers and other HTML tags inline with the web page content.

After describing the process to access a web page we describe below the steps we take in arriving at our approach to investigate load balanced sites.

Our first step is to collect a list of web site URL's which are popular. We used the www.100hot.com web site to get a list of these sites. The web site provides a ranking of the most popular web sites on the web. We understand that a ranking of popular web sites can be open to discussion but the aim is not to test against the top 100 web sites on the web but to get a list of web sites, which we can expect to have a large traffic because of their popularity. Based on the rankings done by www.100hot.com we, use their top 100 sites as a list of highly popular sites with large client traffic. The popular sites are more likely to employ some scheme of load balancing and hence we stand a better chance to investigate the mechanisms used by testing such sites.

Having got a list of such sites we need to find out if DNS based redirection is being done. By doing resolves on the list of web site URL's we found the IP address returned by the DNS was different each time. This result indicated load balancing

is happening based on DNS based redirection. Nslookup is a command line tool on Unix based systems which reolves a given URL with a default DNS server. We were able to validate our results using the tool, since we found that a list of IP addresses mapped to the URL was got. This list was the one used by the web site to round robin between different IPs for each consequent retrieval attempt. On investigation into functions in the C programming language library we found a function gethostbyname(), which resolved the URL to a similar list of IP addresses as nslookup. As with our initial test on the 122 sites we were able to identify DNS based redirection policies using this call. In our preliminary tests we used the three methods to retrieve and validate that sites use multiple IPs mapped to a single URL. Our final scripts used the C function call gethostbyname() to automate the resolution.

After resolving the URL to the IP addresses we have a classified set of sites whose URLs mapped to multiple IP addresses and sites that whose URLs mapped to a single IP address.

Once the IP addresses to which the URL maps is obtained we can deal with the multiple IP set and single IP set differently. Our interest in the single IP set is to find out if some form of transparent redirection mechanism is being used to redirect the client. While our interest in the multiple IP address set is to investigate the headers in the HTTP response, identify differences and infer if the differences would affect the client.

During the initial testing for the proof of concept, we identified servers that returned 302 return codes. This response code is provided as part of the HTTP protocol to indicate that the content has temporarily moved. The 302 code value is returned by servers doing load balancing using HTTP-based redirection. The difference lies in the location tag in the HTTP header pointing to a different

machine on multiple requests. The subset of servers returning the 302 code is interesting and needs to be investigated to identify sites that use the response code to load balance using HTTP-based redirection. To do this we had to make multiple HTTP request on servers that returned the code and observe if we are being directed to the same site or a different site. If we are being redirected to different sites it indicates the HTTP-based redirection scheme for load balancing. By the end of the first phase we classify our set into three sets, namely single IP, multiple IP and the 302 response set. With each of these three sets our steps of investigation were different.

Single IP Address Set - The single IP set constituted the set of servers the DNS server mapped the URL to a single IP and a HTTP request on the server gave a 200 response (the HTTP OK response when no error is encountered in executing the GET). This set of IP addresses needed to be further investigated to identify if load balancing is being done at the network or lower layers. The method we adopted to investigate this set was to do multiple retrievals of the web page from the single IP address. We then compared the header and the links constituting the HTML page to understand if load balancing is being done.

A web page consists of several links. These links are references to other locations that can be manually clicked by the user to go to a different page or an inline link, where the object referenced by the link form part of the page. We call the links that requires users to select and be directed to, as traversal links. We call the links like images and other objects that form the web page and need to be retrieved after the main body of the page is got, as embedded links.

We were aware of companies like Akamai[40] that provided content distribution. Web sites having heavy traffic could have the embedded links placed on Akamai servers. The Akamai servers would place the embedded content geographically distributed across the globe. They would then modify the web page being served to the client by changing the links of such objects to be served by a server closer to the client or using other heuristics. The modification allows the main page to be served by the web server at the site and the components that form the web page to be served from the image servers close to the client. The process places less load on the web site and can achieve load distribution.

If web sites use different servers to provide a single embedded object or reference a different server for a traversal link on the same page for multiple requests, the server is doing a form of load balancing by rotating the server providing the embedded object or the content pointed to by a traversal link. We investigate the embedded and traversal links to verify if the servers serving embedded and traversal links are in fact being changed on multiple requests. Providing distributed content like Akamai has implications for HTTP/1.1, since clients has to now break down the connection to the server from the main web site and establish connections to the servers providing the inline objects. While the distribution of the embedded objects reduces traffic on the main web site, it defeats the HTTP/1.1 design, which allows sequential and pipelining of requests over a single connection.

To understand caching implications of getting different content from the server we look for changes in content in the web page. As mentioned earlier caches are validated by clients by looking at the header tags like ETags and Last Modified Times. On request of the content, by the client, the server

returns the entire content if the tags vary. A change in ETag values or the LModT times would mean that the server would need to resend the content again.

When we make multiple requests to a single IP addresses and if the web site is indeed implementing a transparent switching mechanism, we have a situation where the site must update each of the servers in its clusters with the content. Depending on how the ETag or the last modified time is calculated on the web server software, these values could be different since the content is put on different machines at different times. So in the case that we do find that the content retrieved on the web page remains the same but these tags have changed, it would indicate that either the single HTTP server is generating new tags for each request or more likely that we are being directed to a different server and the change in tags is caused because of the content being added on to a different server at a different time. If this is found to be true we identify the sites that do load balancing for the single IP address as well as find a negative impact of load balancing to caching.

The negative effect of tags changing with the content remaining the same are on caching. Clients will not be able to validate the caches correctly and servers will resend the cached content again. The re-sending because of change in tags, in some ways, hampers an efficient load balancing scheme. The server noticing the change in tag values would resend content again, while in reality this is not needed, since the content has remained the same and the client has requested content from a different server than it made a request to earlier causing the change in tags. To identify cases where content remained the same with tags changing, we did a checksum on the body of the web page on multiple retrievals and compared the tags mentioned above.

If we did find ETags or Last Modified Times change with the checksum on the body remaining the same we most likely have a site that does load balancing using a transparent mechanism of redirection. Also the negative effects on caches due to the erroneous tags as described above would apply to clients accessing the site.

To understand the amount of content distribution, we looked at the number of servers the client would have to access to retrieve embedded content from a non-base server. The content distribution on multiple servers has negative effects as described earlier.

The single IP address sets form the most difficult of our server sets due to the possibility of switching done at lower layers, distributing load at clusters. The investigation to discover sites that do load balancing and the effects on clients become subsequently more difficult at such sites.

Multiple IP Address Set - For the multiple IP set we already identify the sites that are doing DNS based redirection. Since we identify the set as doing a form of load balancing we can investigate the different servers forming the cluster for issues more clearly than the single IP address set. Although the steps are similar to the single IP address, the tests are done on each individual IP address returned by the DNS server as compared to investigating the differences in multiple retrievals from the single IP address in the above set.

To collect the web pages we make HTTP requests to each of the IP addresses that form part of the cluster. Once the web pages are retrieved we try to identify if some form of further load balancing is done. The testing is done in a similar manner to the one mentioned above. We extract the embedded

links and compare if the same embedded link on the page retrieved from one IP address is different from the page retrieved from the another IP address part of the cluster. Similarly we extract the traversal links and apply the same methodology. If we do identify differences it indicates that each of the servers are further using different servers to serve its embedded content and reference the traversal links. This result indicates multiple levels of load balancing and also carries the negative implications to HTTP/1.1 and distributed content as explained above.

We investigate caching issues by comparing the content retrieved from each IP address returned by the DNS. The retrievals are done sequentially and the time between retrievals are minimal. The difference between retrievals is limited to the time to get a web page from the server. The content can be expected to be almost similar, except for changes like objects in the HTML page that change dynamically on each retrieval. Advertisement images are one such object and typically web sites rotate the advertisements on each retrieval. However if we do find the content remaining the same and the HTTP headers like ETags and last modified time used to validate caches changing it indicates a negative effect on caching. In the multiple IP set it also confirms our hypothesis that the change in tags is because the contents are being placed at varying times on different machines and hence cause the HTTP server software to generate a different tag.

Similarly, we also investigate the number of embedded objects the client has to retrieve from a server different from the one serving the web page. A large set of servers used to serve a page would help in reducing traffic to the main web server but could have negative effects for HTTP/1.1 as described earlier.

302 Redirection Set - The set forms the set that respond with a 302 response code. This HTTP code indicates that the resource requested has moved from the present location. The set is interesting because the servers responding with the 302 response could potentially be doing load balancing using HTTP based redirection. In this method the server could redirect the client to a different server with the 302 response and hence achieving a distribution of load. The mechanism is known for the client perceived delayed, in having to establish a second HTTP connection to the redirected server to retrieve content.

This set is investigated by making multiple requests to the server and extracting the URLs that we are redirected to. We compare the different URLs to see if we are being directed to a different server and subsequently doing load balancing. When we find cases that the server we are directed to, responds with another 302 response, we continue making requests to the redirected location until we get a server from which we get a 200 (OK) response. These sites could potentially be doing HTTP redirection at different levels. By making several attempts at finding the server we obtain the 200 response from, we can identify if the HTTP-based redirection is being used to do load balancing.

The problems with HTTP redirection is that it defeats HTTP/1.1, with the client needing to break the connection for a redirection. Also with several levels of HTTP redirection, the client perceived delay increase by having to form a new connection and make multiple requests. The issues of caching and distribution of embedded content on the server serving the web page still exist.

3.5 Summary

This chapter described how we approached our investigation of web sites replicating and distributing web content. The reasons to decide on investigating sites that do load balancing was to understand the techniques being used and to find similarities with techniques suggested in research as well as investigate potential issues with implementations.

To arrive at our test suite we did a preliminary investigation on 122 sites. We were able to classify the sites into three categories, Multiple IP set, Single IP set and 302 Redirected set. For each of the sets we formulated a different approach. For the single IP set we make multiple retrievals on the server and attempt to detect transparent load balancing. With the multiple IP set we know that the site does DNS based redirection and we make retrievals on each of the IP addresses of the cluster to determine issues with content from multiple servers. Finally with the 302 set we try to identify if the server we are being redirected to changes on each request.

The next chapter details our implementation by going into the details of our test setup, scripts and steps in the investigation.

Chapter 4

Implementation

In the previous chapter, we discussed our approach and our motivations for selecting the methodology to collect and analyze web sites for load balancing. This chapter concentrates on the details of our test setup, the algorithms for collection of the data and for analyzing the data. The chapter is divided with flow charts detailing each phase of testing. The ordering of the sections starts with our test setup and our first run of tests on a list of sites. The first run provides us with data to classify our set of sites into different sets based on the redirection method that was used. Once the sets are identified we indicate separate procedures for data collection for each set and subsequently the analysis phase for each set.

4.1 Test Setup

The tests were conducted on a Sun Microsystems Ultra Sparc 2 machine running SunOS version 5.7 at the Worcester Polytechnic Institutes computer science department. The machine hardware or the operating system is not expected to affect the tests and the tests can be run on any machine with a reasonable data connection to the Internet. The machine is not particularly relevant due to the

fact that we do not do any time or performance analysis on the web site. Instead, we are interested in the content returned by the web sites and as long as the data connection is reasonably fast to ensure fairly quick retrievals from web sites, the integrity of the data is achieved. The setup involved the test programs acting as the client to the web servers. The web servers consisted of the machines providing the services at a list of popular sites from www.100hot.com. The complete list of sites numbered 123 and the entire test suite was repeated with a gap of two years. The first test run was conducted in April of 2000 while the second set was conducted in May of 2002. The repetition of tests conducted over the same set of web sites with a gap of two years allowed us to understand the changes in the replicated and distributed content at the web sites over two years. The results would indicate if a greater number of sites do load balancing and if the issues we see in the first set of tests is corrected the second time over.

We used two different programming languages and Unix shell scripts to automate the entire test suite. The languages were the C programming language and Perl. The reasoning behind using different languages was to harness the power that each of the languages offered. The use of the C language is due to the well used network libraries that enabled us to do sockets programming for our HTTP client to retrieve web pages and to make DNS resolutions. Perl is known for its parsing abilities and our use of the language was to take advantage of that power to parse and extract content of interest from the web pages. Since all our programs were run on the SunOS and it being a Unix based machine we used shell scripts under the tcsh shell. The entire test suite of programs could be written in Perl if needed with its available socket library and its close growth in Unix environments but the path of using multiple languages was chosen.

The entire test suite consists of an initial run and classification, data collection

and analysis for single IP, multiple IP and 302 redirection and data collection and analysis of distributed content. The test suite was run in the two periods mentioned above.

4.2 Initial Run and Classification

The first step in the test process was to run the initial tests on the list of 123 sites got from www.100hot.com. In this stage the following steps are followed as shown in figure 4.1

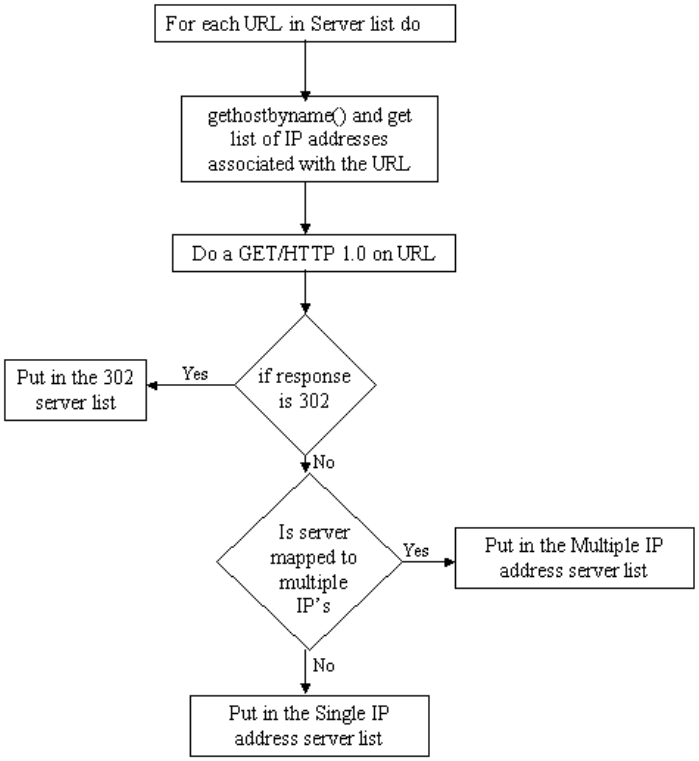

Figure 4.1: Initial Run and Classification of Web Sites

1. The web site list is in the form of a URL and a DNS lookup is need to find the URL to IP mapping.

2. The C function gethostbyname() is done on each server name and the structure passed into the function returns a list of IP addresses the DNS server resolves. The IP address list may contain just one IP address or a list of several IP addresses. When a list of IP addresses are returned it indicates the web site is doing DNS based redirection and the DNS server either uses some heuristics or a round robin policy to return a different IP on each resolution.

3. The HTTP client, written in C, is then used to do a GET / HTTP 1.0 request on the URL. We use HTTP 1.0 since servers implementing HTTP 1.1 are backward compatible with HTTP 1.0 clients. Also we were not interested in pipelining requests, since the interest was in the content returned by the web server and we did not retrieve the inline objects in the HTML page returned.

4. Based on the response we get from the GET request above, we classify the list of servers into two sets. If we find a HTTP redirection response with the 302 code on a request to a particular web site we put it into the 302 server list. The response and the URL the web site redirects us to, is got by parsing the HTTP response using a Perl script. A typical 302 header response is as shown below.

```
HTTP/1.1 302 Found
Date: Wed, 12 Apr 2000 03:24:19 GMT
Server: Apache/1.3.6 (Unix) mod_ssl/2.2.8 SSLeay/0.9.0b
Location: http://lc3.law5.hotmail.passport.com/cgi-bin/login
Connection: close
Content-Type: text/html
```

   This list may have multiple or single IP addresses mapping to the URL but we analyze the set differently from the multiple and single IP set.

5. The web sites that do not fall in the above category are examined for the IP addresses returned by the DNS server on resolution. For URLs that returned a list of IP addresses we put the web site into the multiple IP address list. The remaining set of web sites have a single IP address mapping to its URL and is put in the single IP list.

After classifying the web site list based on the methodology used above, we have three lists namely the single IP list, the multiple IP list and the 302 list. We divided the further tests into two parts the data collection and the analysis phase. The following section contains the data collection steps for the single IP, multiple IP and the 302 list and the analysis steps for the same.

4.3 Data Collection - Single IP Set

The single IP list form an interesting set of web sites, since there is no clear form of identifying which form of redirection scheme is being used. We try to look at the response from multiple retrievals to find differences and investigate these sites. The data collection and formatting steps are as follows and are shown in Figure 4.2.

1. For each server in the Single IP list we make 5 consecutive GET / HTTP 1.0 requests to the IP address the server name maps to and store the responses in separate files. The C client is used to request and retrieve the response.

2. For each of the 5 responses we follow two paths.

3. We split the HTTP response into its header and body. The header contains the IP address of the server responding, the HTTP protocol used, response code, date and other tags allowed by the HTTP protocol. While the body

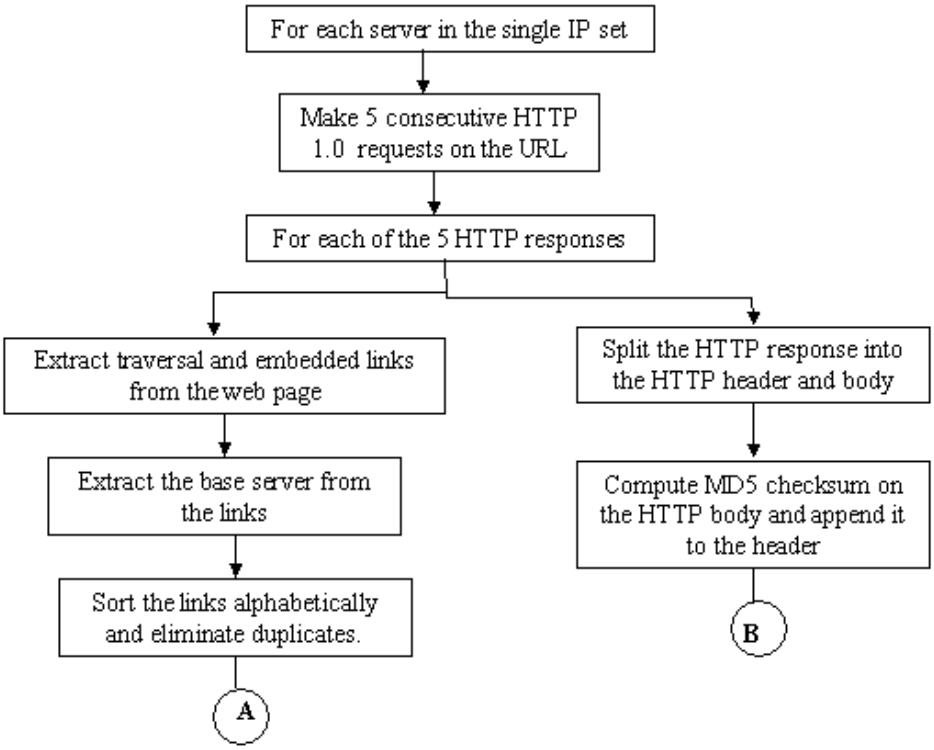

Figure 4.2: Data Collection from the Single IP Set of Web Sites

contains the HTML page that is displayed by browsers and contains the content of interest to a user. A Perl script does the splitting.

4. We compute an md5 checksum on the HTTP body. The md5 checksum is done using the Unix based "md5sum" command. The md5sum computes an 128 bit md5 checksum digest. The checksum is highly likely to be unique to the content of the file, that in this case is the HTTP body. We use the checksum as a unique identifier to the HTML page content and it is appended to the HTTP header for later analysis.

5. The other path we follow is to extract the embedded and traversal links in the web page. The links are removed to separate files. A Perl script does the

extraction. The traversal links are links that reference objects outside of the displayed web page and are intended for users to select, to retrieve the object associated with the link. The tags in the page are identified by searching for the following pattern ‘‘<␣a␣href=‘‘linkname’’>reference␣name’’. Embedded links are objects to be retrieved as part of the web page to render the page. They are characterized by the following tags “img”, “link”, “body”, “frame”, “input” preceded by the less than sign ‘‘<’’.

6. Once the embedded links and traversal links are extracted, we extract the base server from the link. For example for the link “image1.yahoo.com/img23.gif” we remove all text following the first forward slash and retain “image1.yahoo.com”. The reasoning behind extracting just the base servers is to identify the set of servers in the HTML page used for embedded and traversal links.

7. To get a unique set of servers we sort and remove the duplicates from the set of base servers from above. The sorting is done by applying the “sort” Unix command to the files containing the embedded and traversal base server lists. The sort command sorts the list of servers alphabetically. After sorting the base server list, we have all the same base servers bunched together on consecutive line. We eliminate the duplicates by running the “uniq” Unix command. Uniq goes through a file and replaces lines which are duplicates with a single instance of the line, thus eliminating the duplicate base servers for the embedded and traversal set.

After the completion of the data collection phase for single IP we have three files for each retrieval. The first one contains the HTTP header and the md5 checksum on the HTTP body appended to it. The second and third files consist of the unique

set of base servers used in the HTML page retrieved on the multiple retrievals to the single IP. These files are then used for analysis as explained in the later section.

4.4 Data Collection - Multiple IP Set

The steps taken closely follow that of the single IP list. Instead of making multiple requests to a single IP, we make a request to each of the IP addresses returned by the DNS server. We then make a request to the URL after doing a DNS lookup. The DNS lookup of the URL would return one of the IP addresses part of the list. We then compare the response from that IP to the responses from the list of IP's. The Figure 4.3 shows the algorithm followed for the Multiple IP set.

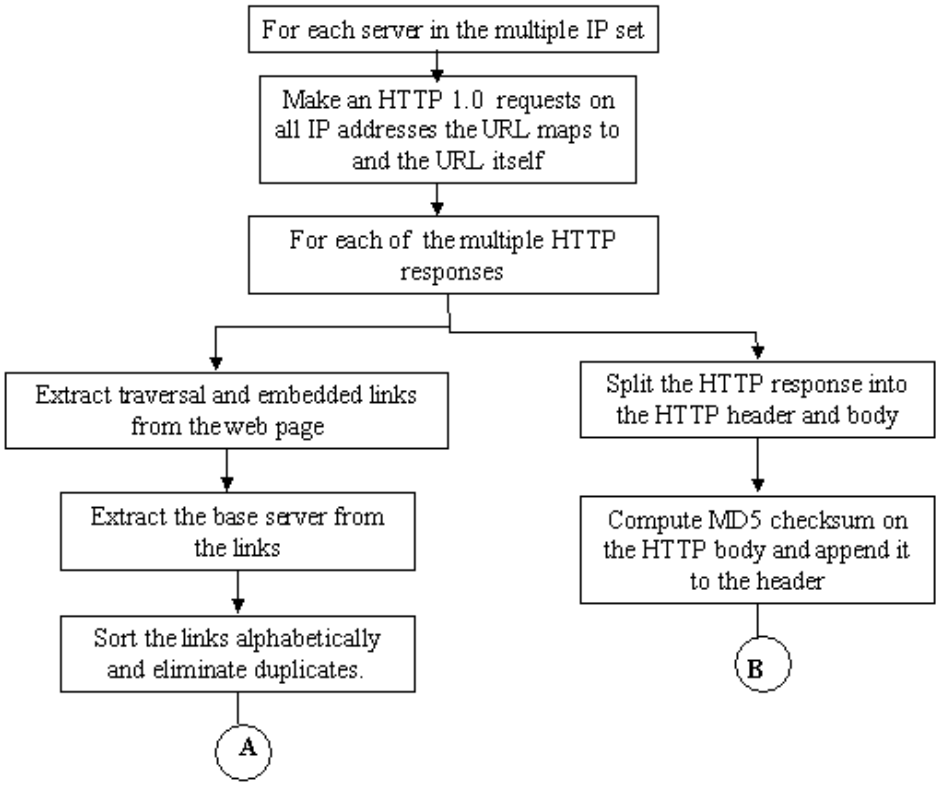

Figure 4.3: Data Collection from the Multiple IP Set of Web Sites

The details of each step in Figure 4.3 follow the steps in the single IP set. The difference is that the comparisons are made between retrievals from different IPs that form the cluster in the multiple IP set as compared to multiple retrievals on a single IP in the single IP set. After the completion of the data collection phase for multiple IP, we have three files for each retrieval. The first one contains the HTTP header and the md5 checksum on the HTTP body appended to it. The second and third files consist of the unique set of base servers used in the HTML page retrieved from the IP list and the URL. These files are then used for analysis as explained in the later section.

4.5 Data Collection - 302 Set

The 302 set consists of sites that returned a 302 response on a GET request. We separate them into a different set, to identify if the redirection is to a different web site on multiple GET requests or if the resource is indeed moved to a different location. Following is the algorithm we use to collect data from the 302 set.

1. For each web site part of the 302 response set we make a GET /HTTP 1.0 request as shown in Figure 4.4. The response as expected, from the first attempt, is a 302 response.

2. The redirected URL is presented as a new link in the "Location" tag in the HTTP header. We extract the link and make a GET request to the link. If we get a 200 response, we go to step 3, else we continue making HTTP GET requests to the link we are directed to until we get a 200 response.

3. Once we get a 200 response from a URL, we use the URL that returned the location of the final content for further testing. Essentially we consecutively

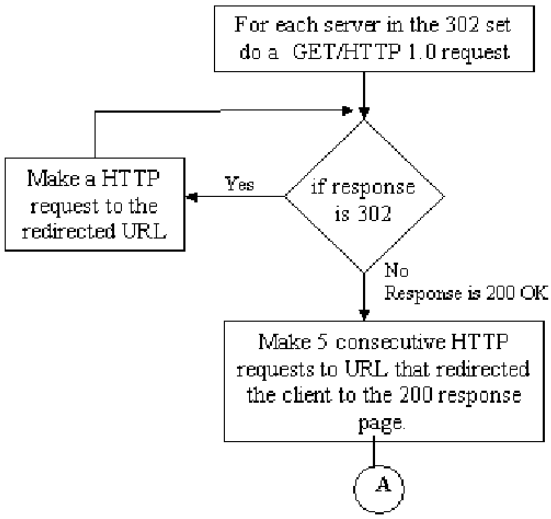

Figure 4.4: Data Collection of 302 Web Sites

make 5 GET requests on the last link that returned a valid location. The reasoning behind doing the multiple requests, is to find out if by making consecutive GET attempts, we are in fact directed to different sites and if the site does HTTP redirection to achieve load balancing by distributing users to different servers. We collect the responses from the 5 GET requests and store it in a file for further analysis

4.6 Analysis - Single IP Set

The analysis section explains the analysis we do on the data gathered in the previous section. The data collection on the single IP set gives us three files each, for the 5 retrievals for every single site. The files include the unique set of embedded and traversal link base servers. It also includes the HTTP header and the md5 checksum on the body for each retrieval. The list below indicates the steps done for the analysis and the report we generate from the analysis. As in the

case of the data collection test suite we automate everything in the analysis section using Unix and Perl scripts. Figure 4.5 shows the steps we take for analyzing this set.

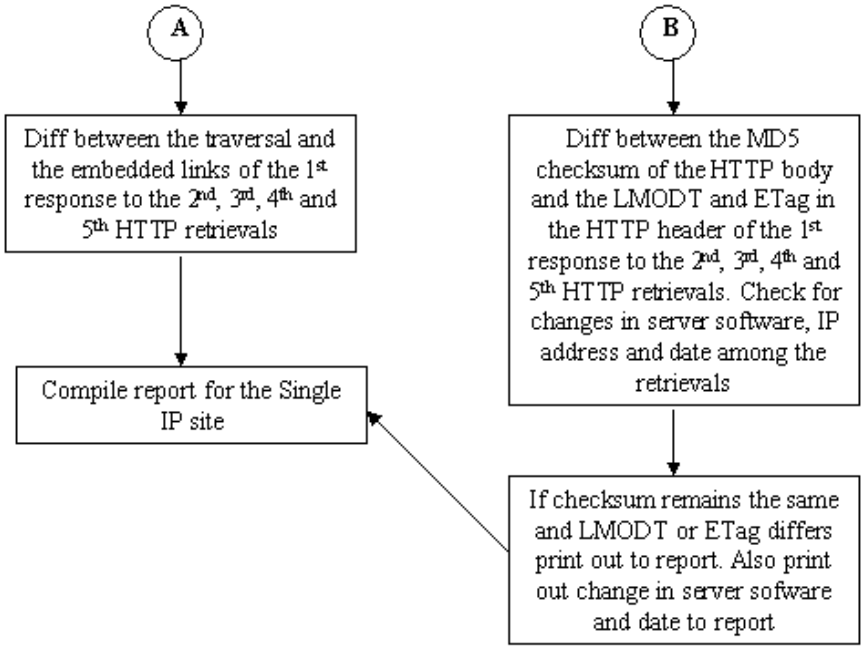

Figure 4.5: Analysis of Single IP Set of Web Sites

1. From point A in Figure 4.2, for the single IP set we obtain two files containing the unique set of base servers used by the HTML page. We diff the files obtained from the embedded base servers got from the first retrieval to that of the 2nd, 3rd, 4th and 5th retrieval. We do a similar comparison for the traversal links. The "diff" utility is a Unix based command that compares two files on a line by line basis and outputs the differences. Since we have sorted and removed duplicates, the server list should be similar on multiple retrievals. A difference in the servers served between two retrievals would indicate that the server rotates the servers that are used to server the

embedded and traversal links. The difference found are output to a report for the site.

2. From point B in Figure 4.2 for Single IP set we have the HTTP header and the checksum for the body in the HTTP response for each retrieval. We compare the Header tags and the checksum of the first retrieval from the server to that of the 2nd, 3rd, 4th and 5th retrieval. Specifically we look at the ETag and LModT tags and compare it between retrievals to the MD5 checksum. This test is to understand the implications on caching and replicated and distributed content. If we find that the ETag or LModT differs and the md5 checksum on the body remains the same it indicates that on consecutive retrievals the content has not changed, while the tags used by caches to validate cache content has changed. Although not needed, the change in tags causes the server to send the a complete response to a client. Also the change in ETag and LModT with the checksum remaining the same, indicates a transparent form of load balancing.

   The reason for the conclusion is because the multiple retrievals are done consecutively, with little time delay, and a response that does not change in content but changes in tags most likely indicates a retrieval from a different server. If this phenomenon is seen in the single IP set and is also seen in the multiple IP set, we can validate the difference in tags being, because of going to multiple servers. We use a Perl script to extract and compare the tags and checksum. We print into the report that we did find a difference in the tags with the checksum remaining the same. In addition to the caching implication of the above changes, we also looked for changes in the software the server runs and the date returned. A major change in the date in

consecutive retrievals could indicate that we have been routed to a different server in which the date and time is not the same as in a previous attempt. The change would also help us identify sites that do transparent load balancing.

With the report containing the results for the analysis, we have a report containing the relevant information from the data collection and the analysis phase. The compilation of the results allows us to easily go through the results from the data collection and the analysis phase pretty easily at the end of the tests.

4.7 Analysis - Multiple IP Set

The analysis section for the multiple IP set is similar to the Single IP set, as with the data collection section for the two sets. The data collection on the multiple IP set gives us three files each, for the retrievals from every IP in the list resolved by the DNS and the response from the URL of the multiple IP set. The files include the unique set of embedded and traversal link base servers. It also includes the HTTP header and the md5 checksum on the body for each retrieval. The steps described in the single IP set indicates the steps done for the analysis in the multiple IP set and the report we generate from the analysis. As in the case of the data collection test suite we automate everything in the analysis section as well using Unix and Perl scripts. Figure 4.6 shows the steps we take for analyzing this set. The comparisons are done with the retrievals from each of the IPs that form a cluster in the multiple IP set.

With the report containing the results for the analysis we have a report containing the relevant information from the data collection and the analysis phase. The compilation of the results allowed us to easily go through the results for the

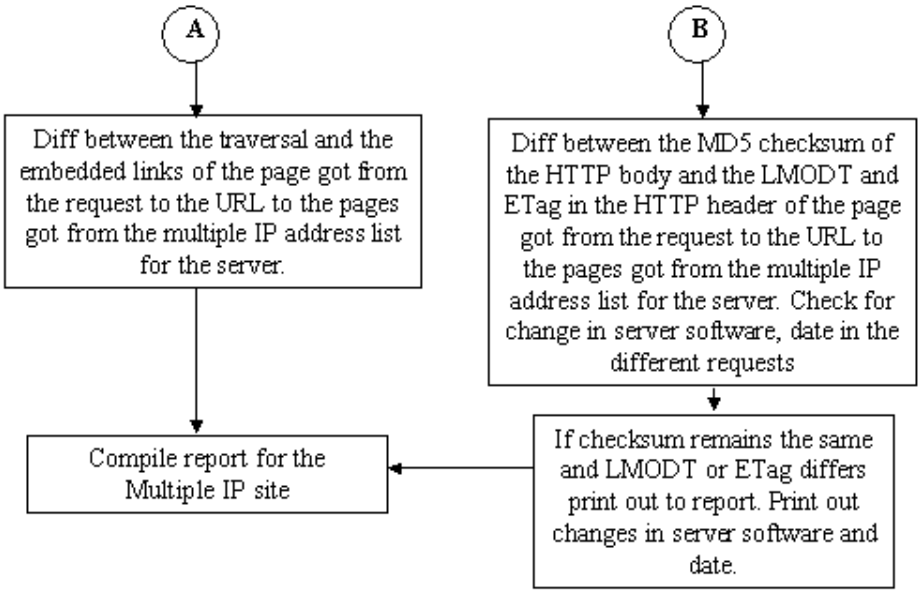

Figure 4.6: Analysis of Multiple IP Set of Web Sites

analysis at the end of the tests.

4.8 Analysis - 302 Set

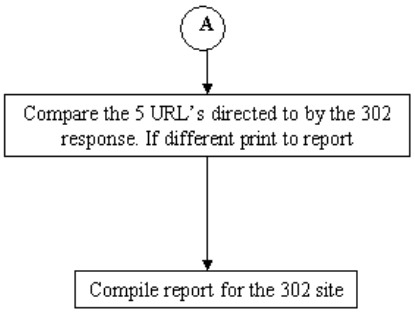

Figure 4.7: Analysis of 302 Web Sites

At the end of the data collection section in Figure 4.4, we have a file containing the responses from the 5 GET requests to the last link we are directed to. This is

indicated by point A in the Figure 4.4. The analysis for this set as indicated by Figure 4.8 compares all the responses and outputs a report if we do find a difference in the server we are directed to. Difference found in multiple responses indicate that we have found a site doing HTTP redirection for load balancing in the set of 302 response sites.

4.9 Data Collection and Analysis - Distributed Content

These tests are used to understand the amount of distribution of objects found in the site list we used. The objects we look at are the embedded objects, which are needed to display the web page and are retrieved by the client to render the web page. As explained earlier the retrieval of embedded objects are done by making subsequent requests for the particular objects that form part of the page. The steps shown by Figure 4.8 for the embedded objects are done.

1. For each URL in the 123 site list, we make a GET / HTTP 1.0 request to retrieve the content. In our case we reused the files we retrieved from our initial run and classification tests.

2. Once the web pages are retrieved, we extract the embedded links that are part of the web page and store it into a file. The Perl script used earlier to extract the embedded objects is used here.

3. Once the embedded links are extracted we remove the base server from the link and sort the servers without removing the duplicates unlike the earlier tests. We do not eliminate duplicates, since we wanted to understand the number of embedded images that needed to be retrieved from each server.

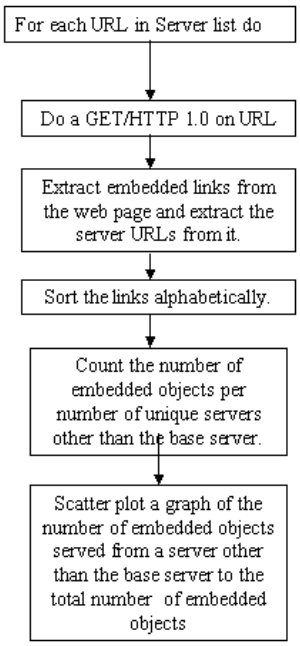

Figure 4.8: Analysis of Distributed Content on Web Sites

4. We then go through the list of embedded base servers and write out the base server and embedded object count pairs for each base server in our list. At the end of this test we have a list of base servers used by the web page and the number of objects needed to be retrieved from each server.

5. Finally we scatter plot a graph, with the number of embedded objects retrieved from the server other than the server we retrieved the web page from, to the total number of objects on the web page.

The graph allows us to understand the number of objects that needs to be retrieved from a server, other than the server we make the initial request. The larger the number of non-base servers used, the larger the number of connections we have to make to completely retrieve the web page.

4.10 Summary

The test programs written used C, Perl scripts and shell scripts. The initial tests in each test run, classify the results into three sets of web sites, the single IP set, multiple IP set and the 302 redirected set. For each of the sets a different test plan is formulated. The single IP set is studied for changes in header tags to indicate transparent switching. Changes in ETag and LModT tags are looked for with the checksum on the HTML body remaining the same. Similar tests to the single IP set are done on the multiple IP set except that the comparisons are done between retrievals from different IPs returned on the name resolution of the URL.
The 302 redirected set is investigated by looking for servers that redirect the client to different servers on multiple requests. Finally the number of embedded objects served from a server, other than the base server is gathered for the server list.
In the next chapter we compile the results obtained from the test run.

Chapter 5

Results

The last chapter explained the test scripts we ran and what we aimed to achieve with each test suite targeted at specific classes of web sites. The classification was done based on the sites doing HTTP or DNS redirection or sites mapping to a single IP address. We summarize the results from our tests in the following sections. Each section deals with the classes we defined. Also the section will be in two parts based on the two runs we did. Part one will cover the first set of tests done in April 2000 and the second part will cover the tests run in May 2002.

5.1 Initial Run and Classification Results

The first step in our tests was to classify the set of servers we were running. To do this we did DNS lookups on the URL, to see if it mapped to multiple or single IP addresses. We then made a single GET request on each URL and retrieved the web page from each of the 123 servers(Appendix A). Based on the responses we classified the sites that gave us a 302 response into a separate list and from the remaining servers classified the list into multiple and single IP address servers. The results of the phase one of our test run in April 2000 gave us the distribution

shown in Table 5.1.

Table 5.1: Classification of Web Sites: April 2000

Web site classes	**Number of sites out of 123**
Single IP set	83
Multiple IP set	22
302 response set with Multiple IP	4
302 response set with Single IP	14
Total	123

Out of the 18, 302 responses, we had 4 web sites that had their URL mapped to multiple IP addresses and they should belong to the multiple IP set based on our classification. We however investigate such sites in the 302 response set, to detect if they do load balancing using HTTP redirection

Phase 2 of the test run in May 2002 produced the following classification.

Table 5.2: Classification of Web Sites: May 2002

Web site classes	**Number of sites out of 123**
Single IP set	78
Multiple IP set	22
302 response set with Multiple IP	4
302 response set with Single IP	19
Total	123

As we see the results changed slightly with there being five web sites not part of the single IP set in the second run. The test was conducted on the same list of sites and we notice that the classification resulted in a total of 123 sites. Also we see an increase in the sites doing the 302 based HTTP redirection. Among the 23 servers that do 302 redirection 4 mapped to multiple IP addresses. The set of servers in the 302 response set that mapped to multiple IP addresses are different

from the previous test run. Appendix A has detailed classification of each site between each run.

Table 5.3 indicates the changes in server classification between the two runs. By adding up the diagonal we find that 92 out of the 123 sites remained in the same set between the two runs. The major shift between the two runs, are the 9 sites that formed the single IP set that started doing 302 redirection in the second run.

Table 5.3: Change in Web Classification Between the Two Test Runs

Test Run	**May 2002**				
	Classified Set	Single IP	Multiple IP	302 set	Total
	Single IP	67	7	9	83
April 2000	Multiple IP	7	13	2	22
	302 set	4	2	12	18
	Total	78	22	23	123

After identifying the sets we perform individual tests on each set. The results are presented in the next three sections.

5.2 Multiple IP Set

In the multiple IP set we perform the tests, as described in the previous chapter, in the data collection and analysis section of the multiple IP set. We retrieve web pages from the URL, as well as the list of IP addresses it maps to. We then extract the embedded and traversal links and the base server of those links to identify if the web site rotates web servers to serve the HTML references in the page. Also we compare the checksum of the content and the ETag and LModT tags in the header of each pages, to see if there are negative caching effects due to load balancing.
In addition, we also look for changes in server software and date since we wanted to identify if going to different servers in a cluster results in instances where these values change. The tests are used to validate the cases of change in the server

software or date values for the single IP set, where the sites could transparently direct us to different servers.

5.2.1 ETag and LModT Variations with the Same Content

In the phase 1 of our testing we found the following web sites returning the same content with a different ETag value.

Table 5.4: Multiple IP Set - Variation in ETag with Same Content: April 2000

Sites with variations in ETag with content remaining same
www.hp.com
www.dell.com
www.wwf.com
www.monster.com

While in phase 2 we found no sites doing ETag or LModT tag changes with content remaining same. We noticed an increase of web sites modifying content on each request and hence a relative change in ETag and LModT time is to be expected. Below is an example of the ETag remaining the same with content changing. The example is taken from our tests www.dell.com in the April 2000 run. The response is a typical header response and the last line indicates the checksum computed on the body of the HTML page by our scripts. The results indicate the checksum remaining the same while the ETag varies.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location: http://143.166.82.52/index.htm
Date: Mon, 10 Apr 2000 16:34:33 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Sun, 26 Mar 2000 11:51:49 GMT
Etag: "7a9dacab1997bf1:809"
Content-Length: 567
```

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location: http://143.166.82.53/index.htm
Date: Mon, 10 Apr 2000 15:30:42 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Sun, 26 Mar 2000 11:51:49 GMT
Etag: "7a9dacab1997bf1:7ec"
Content-Length: 567
```

We were curious to identify the changes that happened to servers that showed ETag changes in the April 2000 run with the content remaining same, to what they are returning now. We list all the servers and the change in their operation over 2 years.

Table 5.5: Multiple IP - ETag Server List Variations from April 2000

Sites with variations in ETag with content remaining same	**Changes in second test run**
www.hp.com	Content consistently changes with ETag
www.dell.com	Content consistently changes with ETag
www.wwf.com	Site permanently moved to www.wwe.com
www.monster.com	Content consistently changes with ETag

Since we did not find any ETag variations in the second run, all sites demonstrating the behaviour in the earlier test run fixed the issue.

5.2.2 Change in Server Software

Similarly we looked for server software changes and the following were the web sites, where changes were seen during the April 2000 run.
In the May 2002 run we found only www.microsoft.com with changes in server software.

Table 5.6: Multiple IP - Variation in Server Software: April 2000

Sites with variations in server software on multiple retrievals
home.netscape.com
www.sportsline.com
www.gamespot.com

Table 5.7: Multiple IP - Variation in Server Software: May 2002

Sites with variations in server software on multiple retrievals
www.microsoft.com

Although changes in server software in multiple retrievals does not affect the client to the best of our knowledge, we use the change in server software as a gauge to identify sites in our next set doing transparent redirection. Below is an example response from www.microsoft.com where the server software changes from IIS version 5 to IIS version 6.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
P3P: CP='ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD TAI TELo OUR SAMo
CNT COM INT NAV ONL PHY PRE PUR UNI'
Content-Location: http://tkmsftwbw16/default.htm
Date: Sat, 18 May 2002 23:09:06 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Fri, 17 May 2002 23:56:14 GMT
Etag: "1095d76dfefdc11:8a7"
Content-Length: 27749


HTTP/1.1 200 OK
Content-Length: 27749
Content-Type: text/html
Content-Location: http://207.46.197.113/default.htm
Last-Modified: Fri, 17 May 2002 23:56:14 GMT
Accept-Ranges: bytes
```

```
Etag: "1095d76dfefdc11:efe1"
Server: Microsoft-IIS/6.0
P3P: CP='ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD TAI TELo OUR SAMo
CNT COM INT NAV ONL PHY PRE PUR UNI'
Date: Sat, 18 May 2002 23:09:09 GMT
Connection: close
```

As in the previous section we observed the sites showing a change in the software in the previous run and found the following results.

Table 5.8: Multiple IP - Server Software Changes from April 2000

Sites with variations in server software	**Changes in second test run**
home.netscape.com	Consistent Server Software
www.sportsline.com	Consistent Server Software
www.gamespot.com	Site is now single IP and responds with consistent server software

We also identified that www.microsoft.com, which shows server software changes in the May 2002 run, displayed consistent software in the April 2000 run as shown in Table 5.9.

Table 5.9: Multiple IP - Server Software Behaviour in April 2000

Sites with variations in server software in May 2002	**Behaviour in earlier test run**
www.microsoft.com	Consistent Server Software

5.2.3 Changes in Date

The other tests done for the multiple IP set was related to looking for significant changes in date of retrieval. We did not see any significant change in this respect.

5.2.4 Changes in Embedded and Traversal Link Servers

Also we inspected for changes in the traversal and embedded link servers that could potentially be used for a form of load balancing. We found no major changes in the servers used to service the embedded and traversal links. The changes found were of servers serving images related to advertisement. In such cases we found some instances where different ad servers were being used. Also there were instances where the servers did change between retrievals. On a detailed inspection the changes were limited and did not indicate any signs of load balancing being done.

5.3 Single IP Set

The single IP set consisted of 83 sites in the April 2000 run and 78 sites in the May 2002 run. As part of the tests, we tried to identify sites doing transparent redirection and for issues with caching as in the case of the multiple IP set. The tests involved getting the web pages 5 times consecutively from the site and comparing the headers and links forming the web page. We looked for header tags like ETags, LModT, server software and date in the headers. We extracted embedded and traversal links to identify if servers used to serve the links on the web page are rotated. Following are the results of the tests.

5.3.1 ETag and LModT Changes

We tried to identify sites that returned different ETag and LModT values with the content remaining the same. Following were the web sites displaying the changes during the April 2000 run.

In the May 2002 run we found www.download.com showing changes in ETag with

Table 5.10: Single IP - ETag Variation with Same Content: April 2000

Sites with variations in ETag with content remaining same
www.spedia.net
www.entrypoint.com
www.ebay.com

content remaining same, while www.usa.net showed changes in LModT and ETag with content remaining same.

Table 5.11: Single IP - ETag Variation with Same Content: May 2002

Sites with variations in ETag or LModT with content remaining same	**Tags changed**
www.download.com	ETag
www.usa.net	ETag and LModT

With the above list, we find that these sites cause problems for cache validation because of an incorrect implementation of the ETag and LModT tags. Because the content is retrieved from the same IP address, we infer that the sites do some form of transparent load balancing. The inference is further validated with the way Apache, a popular web server, calculates the ETag value. It concatenates the inode value of the file, last modified time and size of file to compute the ETag [2]. Since the content and hence the modified times remain the same, the varying parameter would be the inode value, which would change if the file is moved or is retrieved from a different server.

Investigating sites that changed between the two runs gave us the following results For the two sites showing ETag and LModT changes in the May 2002 run we looked at the responses from April 2000.

Table 5.12: Single IP - ETag Variations since April 2000

Sites with variations in ETag with content remaining same	**Changes in second test run**
www.spedia.net	The URL now maps to the multiple IPs
www.entrypoint.com	No ETags are sent and content keeps changing
www.ebay.com	The URL now maps to the multiple IP

Table 5.13: Single IP: ETag and LModT Variation - Behaviour in April 2000

Sites with variations in ETag with content remaining same	**Behaviour in earlier test run**
www.homestead.com	The URL mapped to multiple IPs and no ETag was sent
www.usa.net	No ETag or LModT sent

5.3.2 Server Software Changes

We looked at sites that showed differences in the server software and in such cases the response clearly indicates a change in the server we are getting a response from. In the April 2000 run we found www.women.com showing a different software version, while the May 2002 run Table 5.15 showed changes in server software.

Table 5.14: Single IP - Server Software Variations: April 2000

Sites with variations in server software on multiple retrievals
www.women.com

The site www.women.com still shows a change in server software. The set of sites indicate transparent redirection done to clients making a request to such sites. The two sites in 2002 that showed changes in server software behaviour showed consistent server softwares in 2000.

Table 5.15: Single IP - Server Software Variations: May 2002

Sites with variations in server software on multiple retrievals
www.homestead.com
www.apple.com
www.women.com

Table 5.16: Single IP: Server Software - Behaviour in April 2000

Sites with variations in server software in May 2002	**Behaviour in earlier test run**
www.homestead.com	Consistent server software in response
www.apple.com	Consistent server software in response

5.3.3 Date Changes

To further identify sites showing significant changes in date, we inspected the changes in date over multiple request. Significant changes in date was measured as large changes in time in relation to the time intervals at which requests were made by the client. Such changes would indicate a different server serving the requests. We found no change in the dates that were returned on both runs.

5.3.4 Content Location Differences

Content location is a header tag provided by HTTP/1.1 and we found these tags in certain responses from servers implementing the HTTP/1.1 protocol. Although we did see cases of changes of content-location for the multiple IP set they corresponded to the IP address we were requesting the content from and hence was not of interest. The content-location is described by the RFC [13] as below.
"The Content-Location value is not a replacement for the original requested URI; it is only a statement of the location of the resource corresponding to this

particular entity at the time of the request. Future requests MAY use the Content-Location URI if the desire is to identify the source of that particular entity."

We found the following sites showing a change in content-location among multiple retrievals in the May 2002 run.

Table 5.17: Single IP - Variations in Content-Location: May 2002

Sites with variations in content-location
www.homestead.com
www.usa.net

The results indicate that the sites retrieve content from another machine. Consider below an example response from www.usa.net where the content-location changes and is different from the IP we make the request to. Also the site shows a change in the entity tag with content remaining same.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location: http://165.212.22.66/index.html
Date: Sun, 19 May 2002 00:54:43 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Sun, 19 May 2002 00:52:47 GMT
Etag: "12d8327ecffec11:90b"
Content-Length: 19916

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location: http://165.212.22.65/index.html
Date: Sun, 19 May 2002 00:56:24 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Sun, 19 May 2002 00:54:27 GMT
Etag: "7c622fbacffec11:a0e"
Content-Length: 19916
```

Content-location across retrievals must remain same for transparency. The table below indicates the behaviour of the two sites in the earlier test run. Both sites did not respond with the content-location tags in earlier runs.

Table 5.18: Single IP: Content-Location - Behaviour in April 2000

Sites with variations in content location in May 2002	**Behaviour in earlier test run**
www.homestead.com	No Content-location tag returned in response
www.usa.net	No Content-location tag returned in response

5.3.5 Changes in Embedded and Traversal Link Servers

We inspected for changes in the traversal and embedded link servers that could potentially be used for a form of load balancing. We found no major changes in the servers used to service the embedded and traversal links. The changes found were of servers serving images related to advertisement. In such cases we found some instances where different ad servers were being used. Also there were instances where the servers did change between the retrievals. On a detailed inspection the changes were limited and did not indicate any signs of load balancing being done.

5.4 302 Response Set

The 302 response set involved sites that returned a 302 response and sites that do a HTTP redirection. The set also indicated the largest increase between the two test runs from 18 to 23 sites responding with a 302 response.
The tests run on this set was to continuously make requests to the URL we were being redirected to until we got a 200 response. We then make 5 consecutive

requests to the URL we got the last URL from to see if we are being redirected to a different server and if the site is doing some form of load balancing.
In the first test run we found the following sites responding with a 302 response and redirecting the client to a different URL on multiple attempts

Table 5.19: Sites using HTTP Redirection for Load Balancing: April 2000

Sites load balancing by HTTP redirection
www.hotmail.com
www.msnbc.com
www.fidelity.com
www.mail.com

In the May 2002 run we had the following sites that did HTTP redirection to different sites.

Table 5.20: Sites using HTTP Redirection for Load Balancing: May 2002

Load balancing sites using HTTP redirection
www.msnbc.com
www.fidelity.com
www.passport.com

We found an similar set of sites doing load balancing in the May 2002 run. An interesting case is www.hotmail.com, which redirected the client to the same URL during the May 2002 run of tests but had a special tag "HMServer" in the response. The HMServer returned a URL which seemed to indicate that the site was internally redirecting the client and serving the request from a different server. Shown below are tags when we make a request to the redirected URL "http://lc2.law13.hotmail.passport.com/cgi-bin/login".

```
``HMServer H: LC3-LFD82.law5.internal.hotmail.com ...''
```

```
‘‘HMServer H: LC2-LFD96.law5.internal.hotmail.com ...’’

‘‘HMServer H: LC1-LFD51.law5.internal.hotmail.com ...’’
```

Following is an example 302 response from www.fidelity.com.

```
HTTP/1.0 302 Moved Temporarily
Server: WebSpective Interceptor 3.0
Location: http://www100.fidelity.com:80/


HTTP/1.0 302 Moved Temporarily
Server: WebSpective Interceptor 3.0
Location: http://www300.fidelity.com:80/
```

The drawbacks of HTTP redirection lies in the delay perceived at the client. The delay is due to the additional time needed to break a connection and re-establish a new connection to the redirected URL. The breaking and re-establishment of the connection also defeats the persistent connections available with HTTP/1.1. The Tables 5.21 and 5.22 indicate the behaviour of the web sites in the other test runs.

Table 5.21: Sites Load Balancing using HTTP Redirection - Behaviour in May 2002

Site load balancing in April 2000	**Behaviour in May 2002**
www.mail.com	URL mapped to multiple IPs and returned a 200 response

Table 5.22: Sites Load Balancing using HTTP Redirection - Behaviour in April 2000

Site load balancing in May 2002	**Behaviour in earlier test run**
www.passport.com	URL mapped to a single IP and returned a 200 response

5.5 Distributed Content

Finally we ran tests on the server list to identify the amount of distributed content we see on web pages. The server list consisted of servers that formed the multiple and single IP set. The two sets are chosen because the 302 response set only return the HTTP header with the 302 response with no body and hence no embedded content is present. By distributed content we mean embedded objects that are part of the HTTP page and that need to be retrieved to completely render the page. The analysis is needed to understand the number of objects the client have to retrieve from the server other than the server it initially made the HTTP request for the web page.

In gathering the statistics, we measured the average and median of the number of embedded objects retrieved from servers other than the base server, the total number of objects needed to be retrieved and the number of non-base servers used to get the embedded objects. The average of the total number of embedded objects used on sites increased, while the number of embedded objects retrieved from non-base servers doubled. In terms of the number of non-base servers used there was a shift towards using at least one external server to serve the objects . The shift is validated by the distribution in the results in Tables 5.25 and 5.26.

Table 5.23: Embedded Object Distribution - Statistics on 105 Sites: April 2000

Measured Parameters	**Average**	**Median**
Embedded objects retrieved from non-base servers	4.55	0
Total number of embedded objects	14.84	12
Number of non-base servers used	0.94	0

To further understand the distribution of objects on servers we extracted the

Table 5.24: Embedded Object Distribution - Statistics on 100 Sites: May 2002

Measured Parameters	**Average**	**Median**
Embedded objects retrieved from non-base servers	9.47	2
Total number of embedded objects	20.27	18.5
Number of non-base servers used	1.38	1

distribution of the number of servers used to serve embedded objects by sites. From the results we find that in the first run, as shown in Table 5.25, about half the sites did not use an external server and served the embedded objects from the base server. However there is a shift with more number of sites requiring clients to go to at least one other server to retrieve the embedded objects to render the web page. Tables 5.25 and 5.26 show the percentage of servers going to one, two or three servers other than server used to get the base HTML page.

Table 5.25: Embedded Object Distribution- Non-base Servers: April 2000

Number of servers other than base server to get embedded objects	**Percentage of total test sites displaying content distribution**
0	51
1	17
2	21
3	8
4	3
Total percentage	100

We saw more of content distribution. We identified the data points for analysis by extracting all the embedded images and plotting a graph of the number of objects needed to be retrieved from non-base servers to the total number of objects for the web page.

Table 5.26: Embedded Object Distribution- Non-base Servers: May 2002

Number of servers other than base server to get embedded objects	**Percentage of the test sites displaying content distribution**
0	33
1	30
2	21
3	12
4	1
5	2
9	1
Total Percentage	100

The two graphs in Figure 5.1 and Figure 5.2 show the embedded object distribution of the two runs of our tests.

Figure 5.1 and Figure 5.2 indicate a large number of embedded objects present at sites. The graph will have no objects above the diagonal splitting the graph because the objects retrieved from different servers will never exceed the total number of objects. A data point on the diagonal indicates that all the embedded objects had to be retrieved from a server other than the server the connection was made to. A data point on the x-axis indicates that all objects are retrieved from the base server.

We see an increase in the embedded objects on sites between the two test runs. The number of servers on the diagonal of Figure 5.2 over that on Figure 5.1 indicates that an increasing number of web sites require clients to retrieve most of the embedded objects from a server other than the base server. The instances are of sites which have image servers to serve the client with embedded content. Although the number of embedded objects are large, the number of servers the client has to go to is quite small. In the April 2000 run, we tested 104 servers

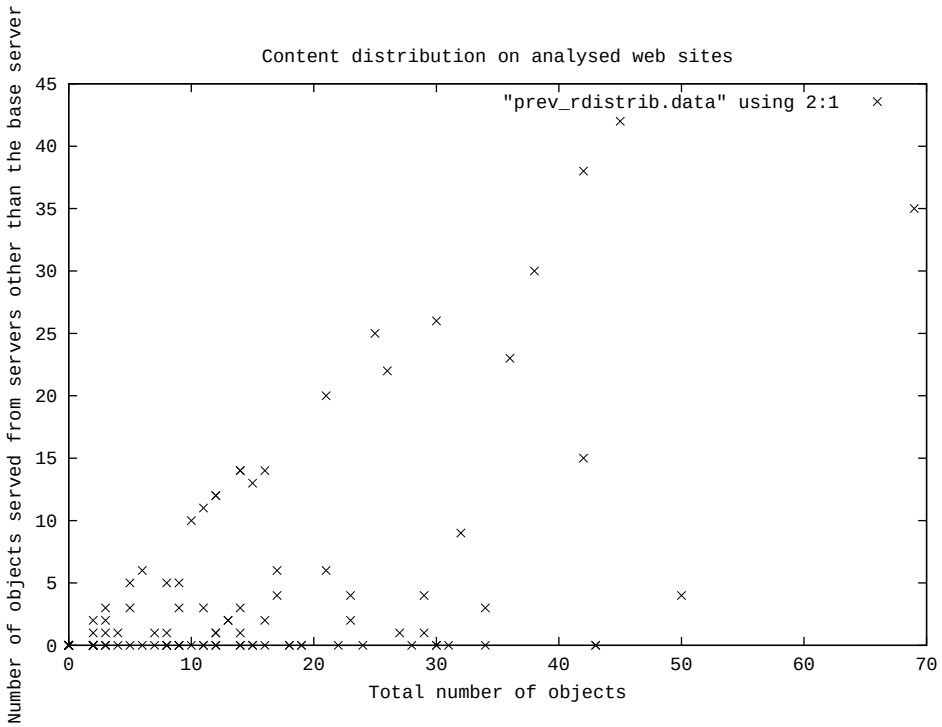

Figure 5.1: Relative Distribution of Embedded Objects on Web Sites: April 2000

constituting the single and multiple IP list. Out of the 104 servers, 51 showed content distribution by going to a server different from the base server to retrieve content. Similarly in the second run out of a total of 100 sites tested we found 64 sites doing content distribution. Figure 5.2 displays the relation of objects retrieved from distributed servers to the the total content needed to be retrieved. The large amount of distribution is clearly negative for the persistence of HTTP/1.1, although it helps in reducing load on servers. With studies done to show the benefits of pipelined HTTP connections it would be worthwhile for sites doing this form of load balancing to rethink the strategy.

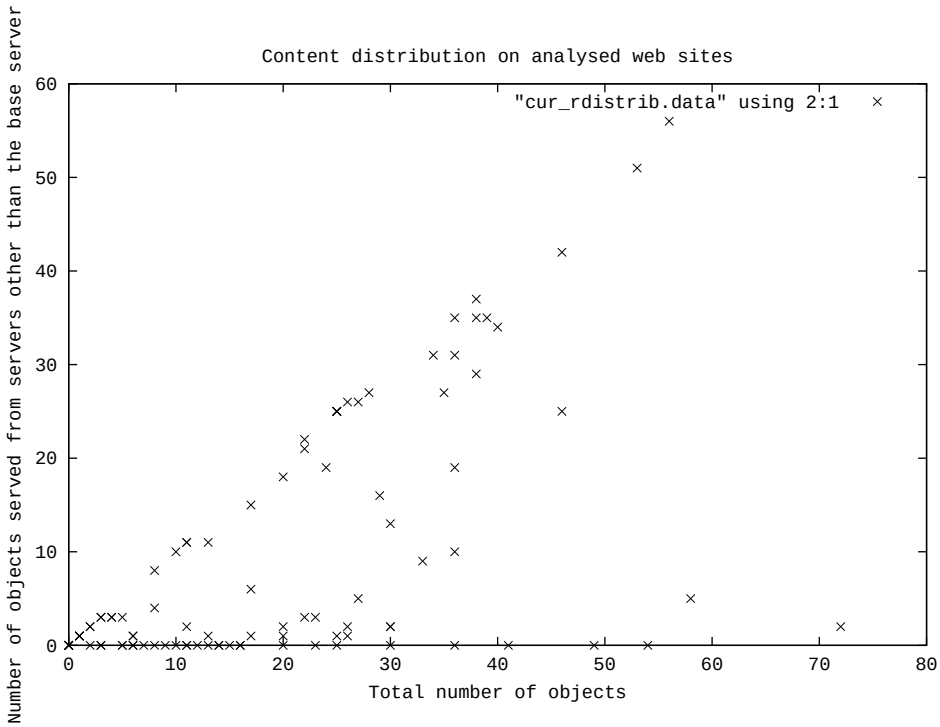

Figure 5.2: Relative Distribution of Embedded Objects on Web Sites: May 2002

5.6 Summary

In this chapter, we discussed the results from our tests explained in the previous chapter. We run the preliminary tests to classify the sites into three groups. We found that sites doing 302 redirections increased, while the number of sites having a single IP reduce. However the increase did not result in an increase in instances of sites using 302 redirection for load balancing.

In our tests with the Single IP set and Multiple IP set we found cases where ETag and LModT time changes with the content remaining same. Caches would invalidate content from such sites and hence it has a negative effect. The negative effects is an offshoot of content replication among multiple servers. The results also indicate transparent load balancing in the case of sites with a single IP address. We found sites showing changes in server software and content location,

again indicating transparent switching in the single IP set. Finally we investigated if sites use embedded and traversal links to rotate servers serving the links but could not find any conclusive results indicating load balancing. We analysed content distribution in our set of sites and the results indicated about half of the sites using content distribution.

Chapter 6

Summary and Conclusion

At the start of our research into distributed and replicated web content, we were aware of active research happening in the area of load balancing. Being web users ourselves the delay at popular web sites was apparent. Our preliminary tests indicated the use of multiple servers by web sites. With increasing web traffic it becomes apparent that the incidence of such architectures would increase. We realized the importance of content replication and distribution as an area in research and although research was being done and techniques were suggested there was a lack of real data on what was being done at web sites. With our previous research background and interest in caching and the HTTP protocols, the interplay between these techniques and load balancing caught our interest. All three techniques aimed at improving performance for web servers and yet no study was done to study the effects of one techniques over the other.

In our approach we decided to test sites most likely to use replicated and distributed content to reduce load. Our aims were to identify sites that do load balancing, to understand the implementations at such sites and the issues with such implementations on clients and in relation to other techniques like HTTP and

caching. Early tests showed that a large number of sites in our set mapped to single IP addresses. Sites using application layer switching were easy to detect since the client is involved in resolving the URL and in making the HTTP connections in the DNS based and HTTP based redirection schemes. Sites using a virtual IP and switching clients transparently required closer scrutiny of the responses to identify them as doing load balancing. Further we were interested in understanding if there are any issues with replication or distribution, on the HTTP/1.1 protocol and caching.

To test web sites, our test setup involved making requests to servers, collecting the responses and analysing them. We ran client programs against the web servers and automated scripts to parse and retrieve relevant content from the responses. The responses were checked for changes in server software, date, content-location and other such tags, which change based on the server the content is retrieved from. We looked for changes in ETag and LModT, which are unique cache validators with content remaining same. We looked for sites doing 302-based redirection for load balancing. Finally we understand the extent of distribution of content on servers. We were able to identify sites that do application layer load balancing like DNS and HTTP based redirection quite easily. Sites that had URLs mapped to a single IP address were more challenging and needed to be identified by looking at responses. A small set was identified by looking at the server software, content-location, entity tag and last modified time identifier tags in the HTTP header. Cases where entity tags and last modified time change with content remaining constant have a negative effect on caching. With replication of content the challenge in maintaining unique validators for contents increases. Having found such issues of negative caching it would be useful to study methods by which identifiers can be generated independent of the machine the content comes from.

There is a large amount of distribution of embedded content that form the web page at sites. Although the servers needed to retrieve the embedded content is a small set the distribution of content works against the design of persistence in HTTP/1.1 protocol. With studies showing the benefits of pipelined HTTP/1.1 web sites must try to understand the benefits and disadvantages of distributing content on the HTTP/1.1 protocol performance. The degree of distribution increased between the two runs of tests.

Our tests did not show any load balancing done by sites by rotating the servers used to serve embedded and traversal links on web pages. Although a more distributed location of our test clients, might have shown better results. Sites not sending several tags in the HTTP header hampered the investigation. A future step would be to standardize on tags that must be sent by servers.

The approach used to investigate replicated and distributed content at web sites and the test methodology can be used as a model for future investigations of load balancing or of distributed and replicated web content. Running of tests with a gap of two years allowed us to see several sites resolving issues with caching that we saw in our first test run. There has been an increase in distributed content over the period and a need to understand the effect of the content distribution and the provision of persistent connections and pipelining of requests in HTTP/1.1 has become pertinent. Several sites that displayed changes in software, IP and other tags change in the second run. There is however no clear indication within our test set that load balancing at sites are increasing.

Following are the some of the probable directions a researcher can follow up based on our work. More studies can be done placing clients in geographically distributed locations and making requests to web sites. Since servers use techniques to gauge the geographical proximity of clients we might see different behaviour at load

balanced sites. Specifically there might be cases where sites using servers like Akamai distribute content such that they are closer to the location of the client. The interplay of caching with content replication and distribution was explained. Tests indicated issues with caching content from multiple servers. The challenges in providing clients with consistent cache identifiers in a multiple server environment increases over a single server environment. Web servers cannot tie identifiers to the any unique server parameter and the tags used to validate caches must be a function of the content being served. A useful study would be to further understand how effective caching can be done on such sites and in maximizing the clients cache hits.

Large number of web sites distribute content and with the benefits of HTTP/1.1 persistent connections and pipelining of requests, a study is needed to understand the impact of both methods. A new architecture based on the study is needed to take advantage of bothe techniques.

In conclusion we hope our study filled a gap in the understanding of techniques used by sites to balance load and replicate and distribute content. We believe the results highlights several issues in current implementation of load balancing, which need to be addressed and fixed. Future work in terms of further investigation and better architectures can be done based on the findings and issues we faced. We hope our approach and test methodology provide a model for studying sites doing content replication and distribution.

Appendix A

Server List

Following is the list of servers used during the test runs in April 2000 and May 2002. The table indicates the classification of each site in April 2000 and the change in May 2002

Table A.1: Server List: Classification for April 2000 and May 2002 test run

Site name	**Classification in April 2000**	**Classification in May 2002**
www.yahoo.com	Multiple IP set	Multiple IP set
www.four11.com	Single IP set	Multiple IP set
www.microsoft.com	Multiple IP set	Multiple IP set
www.msn.com	Multiple IP set	Multiple IP set
www.linkexchange.com	302 Response set	302 Response set
www.aol.com	Multiple IP set	Multiple IP set
home.netscape.com	Multiple IP set	Multiple IP set
www.go2net.com	302 Response set	302 Response set
www.lycos.com	Multiple IP set	Single IP set
www.hotmail.com	302 and Multiple IP set	302 and Multiple IP set
www.whowhere.com	302 and Multiple IP set	302 and Multiple IP set
www.excite.com	Single IP set	Single IP set
www.mckinley.com	Single IP set	Single IP set
www.city.net	302 Response set	Single IP set
www.webcrawler.com	Single IP set	Single IP set
www.altavista.digital.com	Multiple IP set	Single IP set

Table A.2: Server List: Classification for April 2000 and May 2002 test run

Site name	**Classification in April 2000**	**Classification in May 2002**
www.compaq.com	Single IP set	Single IP set
www.tandem.com	Single IP set	Single IP set
www.go.com	Single IP set	Single IP set
www.xoom.com	302 Response set	Multiple IP set
www2.bluemountain.com	Single IP set	302 Response set
www.mtv.com	Single IP set	Multiple IP set
www.bbc.co.uk	Single IP set	Single IP set
www.amazon.com	302 Response set	302 Response set
www.quote.com	Single IP set	302 Response set
www.msnbc.com	302 and Multiple IP set	302 Response set
www.cnet.com	Multiple IP set	Multiple IP set
www.search.com	Multiple IP set	Single IP set
www.news.com	Multiple IP set	Single IP set
www.download.com	302 and Multiple IP set	Single IP set
www.spedia.net	Single IP set	Multiple IP set
www.snowball.com	Single IP set	Single IP set
www.burstmedia.com	Single IP set	Single IP set
www.usa.net	Single IP set	Single IP set
www.entrypoint.com	Single IP set	Single IP set
www.cnn.com	Multiple IP set	Multiple IP set
www.real.com	Single IP set	Single IP set
www.sony.com	Single IP set	Single IP set
www.station.sony.com	Single IP set	302 Response set
www.scea.sony.com	Single IP set	Single IP set
www.sportsline.com	Multiple IP set	Multiple IP set
www.ebay.com	Single IP set	Multiple IP set
www.theglobe.com	Single IP set	Single IP set
www.homestead.com	Single IP set	Single IP set
www.macromedia.com	Single IP set	Single IP set
www.webjump.com	Single IP set	Single IP set

Table A.3: Server List: Classification for April 2000 and May 2002 test run

Site name	**Classification in April 2000**	**Classification in May 2002**
www.nettaxi.com	Single IP set	Single IP set
www.snap.com	Single IP set	302 Response set
www.tucows.com	Single IP set	Single IP set
www.freethemes.com	Single IP set	Single IP set
www.looksmart.com	Single IP set	Single IP set
www.beseen.com	Single IP set	Single IP set
www.fool.com	Single IP set	Single IP set
www.fortunecity.com	302 Response set	302 Response set
www.ugo.net	Single IP set	302 Response set
www.gamepen.com	302 Response set	Single IP set
www.game-revolution.com	Single IP set	Single IP set
www.gamedemo.com	Single IP set	Single IP set
www.ecircles.com	302 Response set	302 Response set
www.etrade.com	Single IP set	302 Response set
www.chek.com	Single IP set	Single IP set
www.collegeclub.com	Single IP set	302 Response set
www.cjb.net	Single IP set	Single IP set
www.hp.com	Multiple IP set	Multiple IP set
www.zdnet.com	Single IP set	Single IP set
www.hotfiles.com	302 Response set	302 Response set
www.pathfinder.com	302 Response set	302 Response set
www.freevote.com	Single IP set	Single IP set
www.onelist.com	Single IP set	302 Response set
www.apple.com	Single IP set	Single IP set
www.shareplay.com	Single IP set	Single IP set
www.datais.com	Single IP set	Single IP set
www.talkcity.com	Single IP set	Single IP set
www.nasdaq.com	Single IP set	Multiple IP set
www.dell.com	Multiple IP set	Multiple IP set
www.nasa.gov	Single IP set	Single IP set
mars.sgi.com	Single IP set	Single IP set
mpfwww.jpl.nasa.gov	Single IP set	Single IP set
www.eudoramail.com	Single IP set	Single IP set
www.mapquest.com	Single IP set	Multiple IP set
www.simplenet.com	302 Response set	Single IP set
www.weather.com	Multiple IP set	Multiple IP set

Table A.4: Server List: Classification for April 2000 and May 2002 test run

Site name	**Classification in April 2000**	**Classification in May 2002**
www.about.com	Single IP set	Single IP set
www.nih.gov	Single IP set	Single IP set
www.recycler.com	Single IP set	Single IP set
www.realtor.com	Single IP set	302 Response set
www.classifieds2000.com	Single IP set	Single IP set
www.tminterzines.com	Single IP set	Single IP set
www.ircache.net	Single IP set	Single IP set
www.wwf.com	Multiple IP set	Single IP set
www.100free.com	Single IP set	Single IP set
www.infospace.com	Single IP set	Single IP set
www.web1000.com	Single IP set	Single IP set
www.sun.com	Single IP set	Single IP set
java.sun.com	Multiple IP set	Single IP set
www.dash.com	Single IP set	Single IP set
www.usatoday.com	Single IP set	Single IP set
www.developer.com	Single IP set	Single IP set
www.cisco.com	Single IP set	Single IP set
www.internet.com	Single IP set	Single IP set
www.mail.com	302 Response set	Single IP set
www.bloomberg.com	Single IP set	Single IP set
www.mp302 Response set.com	Single IP set	Single IP set
www.ibm.com	Multiple IP set	Multiple IP set
www.passport.com	Single IP set	302 Response set
www.dai.net	Single IP set	Single IP set
www.noaa.gov	Single IP set	Single IP set
www.schwab.com	Single IP set	Single IP set
www.winamp.com	Single IP set	Single IP set
www.fidelity.com	302 Response set	302 Response set
www.uproar.com	302 Response set	302 Response set
www.mplayer.com	Single IP set	Single IP set
www.gamespot.com	Multiple IP set	Single IP set
www.monster.com	Multiple IP set	Multiple IP set

Table A.5: Server List: Classification for April 2000 and May 2002 test run

Site name	**Classification in April 2000**	**Classification in May 2002**
www.Headbone.com	Single IP set	Single IP set
www.women.com	Single IP set	Single IP set
www.cdworld.com	Single IP set	Single IP set
www.infosel.com	Single IP set	Single IP set
www.scu.edu	Single IP set	Single IP set
www.berkeley.edu	Single IP set	Single IP set
www.nai.com	Single IP set	Multiple IP set
www.uolmail.com	Multiple IP set	302 and Multiple IP set
www.fbcgi.com	Single IP set	Single IP set

Bibliography

[1] Internet Weather Report at http://www.mids.org.

[2] C. E. Wills and M. Mikhailov, "Towards a better understanding of web resources and server responses for improved caching," in *Proceedings of the Eighth International World Wide Web Conference*, May 1999.

[3] V. Cardellini, M. Colajanni, and P. S. Yu, "Dynamic load balancing on web-server sytems," *IEEE Internet Computing*, May-June 1999.

[4] V. Cardellini, M. Colajanni, and P. Yu, "Scalable web server systems: Architectures, models and load balancing algorithms," in *Tutorials and Workshops, ACM SIGMETRICS*, July 2000.

[5] D. Mosedale, W. Foss, and R. McCool, "Lessons learned adminstering netscape's Internet site," *IEEE Internet Computing, Vol. 1, No. 2, March-April*, 1997.

[6] C. Yoshikawa, B. Chun, P. E. A. Vahdat, T. Anderson, and D. Culler, "Using smart clients to build scalable services," in *Proceedings of Usenix annual technical conference*, April 1997.

[7] R. Vingralek, Y. Breitbart, M. Sayal, and P. Scheuermann, "Web++: A system for fast and reliable web service," in *Proceedings of Usenix annual technical conference*, June 1999.

[8] M. Colajanni, P. S. Yu, and D. M. Dias, "Analysis of task assignment policies in scalable distributed web server systems," *IEEE Transaction on Parallel and Distributed Systems, Vol. 9, No. 6*, June 1998.

[9] www.cisco.com.

[10] Ciscos Local Director "http://www.cisco.com/warp/public/751/lodir/index.html".

[11] E. Anderson, D. Paterson, and E. Brewer, "The magicrouter, an application of fast packet interposing," in *Second Symposium on Operating Systems Design*, May 1996.

[12] O.P.Damani, P.E.Chung, Y. Huang, C.Kintala, and Y. M. Wang, "One-ip: Technique for hosting a service on a cluster of machines," *J. Computer Networks and ISDN systems*, vol. 29, September 1997.

[13] R. F. et al., "HTTP/1.1," tech. rep., W3C, June 1999. RFC.

[14] V. Cardellini, M. Colajanni, and P. Yu, "Redirection algorithms for load sharing in distributed web server systems," in *Proceedings 19th IEEE Int'l Conference on Distributed Computing*, May 1999.

[15] D. Andersen, T. Yang, V. Holmedahl, and O. H. Ibarra, "Sweb: Towards a scalable world wide web server on multicomputers," in *Proceedings of the 10th International Parallel Processing Symposium*, April 1996.

[16] V. Cardellini, M. Colajanni, and P. S. Yu, "Redirection algorithms for load sharing in distributed web-server systems," in *Proceedings of the 19th International Conference on Distribtuted Computing Systems*, June 1999.

[17] V. Cardellini, M. Colajanni, and P. S. Yu, "Geographic load balancing for scalable distributed web systems," in *Proceedings of the International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'00)*, August/September 2000.

[18] V. Cardellini and M. Colajanni, "A performance study of distributed architectures for the quality of web services," in *Proceedings of Hawaii International Conference on System Sciences*, January 2001.

[19] E. Zegura, M. Ammar, Z. Fei, and S. Bhattacharjee, "Application-layer anycasting: A server selction architecture and use in a replicate web service," *IEEE/ACM Transactions on Networking*, August 2000.

[20] J. Veizades, E. Guttman, C. Prkins, and S. Kaplan, "Service location protocol, RFC - 2165."

[21] F. Hao and E. Zegura, "Supporting server selection in differentiated networks," in *Proceedings of INFOCOM 2001*, April 2001.

[22] J. D. Guyton and M. F. Shwartz, "Locating nearby copies of replicated Internet servers," in *Proceedings of the ACM/SIGCOMM'95 Conference*, August 1995.

[23] T. Ogino, M. Kosaka, Y. Hariyama, K. Matsuda, and K. Sudo, "Study of efficient server selection method for widely distributed networks," in *Proceedings of the INET'00 Conference*, July 2000.

[24] M. Conti, E. Gregori, and F. Panzieri, "Load distribution among replicated web servers: A QOS based approach," in *Second Workshop on Internet Server Performance*, May 1999.

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

[26] L. Cherkasova and S. R. Ponnekanti, "Optimizing a content aware load balancing strategy for shared web hosting service," in *Proceedings of the International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'00)*, August/September 2000.

[27] L. Cherkasova, "Flex: Load balancing and management strategy for scalable web hosting service," in *Proceedings of the IEEE Symposium on Computers and Communication*, July 2000.

[28] C.-S. Yang and M.-Y. Luo, "Efficient support for content-based routing in web server clusters," in *Usenix symposium on Internet Technologies and Systems*, October 1999.

[29] K. Bharat and A. Z. Broder, "Mirror and mirror on the web: A study of host pairs with replicated content," in *Proceedings of the Eighth International World Wide Web Conference*, May 1999.

[30] S. Jamin, C. Jin, A. R. Kruc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the Internet," in *Proceedings of the IEEE Infocom 2001 Conference*, April 2001.

[31] A. Myers, P. Dinda, and H. Zhang, "Performance characteristics of mirror servers on the Internet," in *Proceedings of the IEEE Infocom'99 Conference*, March 1999.

[32] T. B. et al., "HTTP/1.0," tech. rep., W3C, May 1996. RFC.

[33] E. Cohen, H. Kaplan, and J. D. Oldham, "Policies for managing tcp connections under persistent http," in *Proceedings of the Eighth International World Wide Web Conference*, May 1999.

[34] H. Frystyk, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, "Network performance effects of HTTP/1.1, CSS1 and PNG," in *Proceedings of the ACM/SIGCOMM'97 Conference*, September 1997.

[35] B. Krishnamurthy and C. E. Wills, "Analyzing factors that influence end-to-end performance," in *Proceedings of the Ninth International World Wide Web Conference*, May 2000.

[36] M. Aron, P. Druschel, and W. Zwaenepoel, "Efficient support for P-HTTP in cluster based web servers," in *Proceedings of the 1999 USENIX Annual Technical Conference*, June 1999.

[37] A. Iyengar and J. Challenger, "Improving web server performance by caching dynamic data," in *In Unix Symposium on Internet Technologies and Systems*, December 1997.

[38] R. B. Bunt, D. L. Eger, G. M. Oster, and C. L. Williamson, "Achieving load balance and effective caching in clustered web servers," in *Proceedings of 4th Internation Web Caching Workshop*, March/April 1999.

[39] "www.100hot.com."

[40] "www.akamai.com."