WALRUS

A Low Latency, High Throughput Web Service Using Internet-wide Replication

David M. Shaw
dshaw@cs.jhu.edu

Advisor: Dr. Yair Amir
yairamir@cs.jhu.edu

Submitted as partial fulfillment of the requirements for the degree of Master of Science in Engineering from The Johns Hopkins University.

“The time has come,” the Walrus said,
   “To talk of many things:
Of shoes – and ships – and sealing-wax –
   Of cabbages – and kings –
And why the sea is boiling hot –
   And whether pigs have wings.”

Lewis Carroll, Through the Looking-Glass,
and What Alice Found There, 1872.

“Goo Goo Goo Joob”

John Lennon / Paul McCartney, I Am The Walrus, 1967.

Acknowledgments

I would like to thank my parents, Dr. Henry and Evelyn Shaw, for all their help over the long and winding road that I followed before arriving at Johns Hopkins. Whether I was designing special effects, puppeteering, filmmaking, or programming a computer, they never once stopped supporting me and encouraging me to excel. Their love and caring have enabled me to be where I am today.

I owe a debt to my advisor Dr. Yair Amir for all his help and support in getting my Masters degree. His insistence on proper design and his attitude of building it, rather than just planning it was a constant help while constructing the Center for Networking and Distributed Systems lab. I have learned a lot from him.

I am grateful for my sisters, Laura and Jessica. They continue to demonstrate for me professional excellence without losing their own special magic.

I must also mention Lloyd Taylor, formerly of the Applied Physics Lab, and Dr. Scott Smith of the Computer Science department. Both took a chance on that crazy film student with his phonetic text-to-cow translator, and gave me the opportunity to prove what else I could do.

Thanks also to Jonathan Stanton, who sat right behind me through a large part of my time at Hopkins. His willingness to stop working on whatever he was doing to debate endless points of operating system subtleties was always a joy for me.

Ben Kram and Ryan Brown kept reminding me that the real reason people use computers was for the sheer Inherent Coolness of it all. Rena Bunder would drag me off to the Paper Moon and never let me forget the wonderful wide world away from the keyboard. Pierre Joseph, Bill Bogstad, and Denis Gerasimov were not afraid to let the new kid borrow the occasional root password to learn with.

Finally, I absolutely must thank Paige Malerman. Her constant and untiring support, her cheerful freckled grin, and the editing miracles she wrought on the drafts of this document are all things I will be eternally grateful for. She is my friend, she is my companion, she is my love.

David M. Shaw
August, 1998

My work in the Center for Networking and Distributed Systems was supported by the Defense Advanced Research Projects Agency (DARPA) under grant F306029610293, by the NASA Center of Excellence in Space Data and Information Sciences under grant NAS5-32337-555505652, and by the Department of Computer Science, The Johns Hopkins University.

Abstract

The explosion of the content on the World Wide Web has created a situation where most Internet traffic is Web related. At the same time, practically all of the popular Web sites are still served from single locations. This combination of circumstances necessitates frequent long distance network data transfers (potentially repeatedly), which in turn result in a high response time for users, and are wasteful of the available network bandwidth. In addition, the frequent use of single locations to serve from creates a single point of failure between the Web site and its Internet provider.

This paper presents a new approach to Web replication, the Walrus system, which is intended to remedy these problems. Under Walrus, each of the Web server replicas may reside in a different part of the network, and the client's browser is automatically and transparently directed to the best server for this client.

"Best" is a relative term, dependent on where the client is located on the network, and which of the collection of replicated servers is most able to answer the client's request. Weighting is also a factor, and in the extreme case, server load may be more significant than network topology, as it is probably a better decision for the client to be sent to a more distant and less busy server than to one that is extremely overloaded but closer.

Implementing this architecture for popular Web sites will result in a better response time and a higher availability for these sites. Equally important, this architecture will potentially cut down a significant fraction of Internet traffic, freeing bandwidth for other uses.

Table of Contents

1. Introduction

The explosion of content on the World Wide Web has created a situation where the majority of Internet traffic is Web related. In fact, as of the beginning of 1995, Web traffic became the single largest load on the Internet [NSF-Traf95]. This explosive growth shows no sign of slowing down. In spite of the widespread acknowledgement of the problems related to this growth, practically all of the popular Web sites are still served from single locations, as per an earlier network model. This combination of circumstances necessitates frequent long distance network data transfers (potentially repeatedly), which in turn result in a high response time for users, and are wasteful of the available network bandwidth. In addition, this frequent use of single locations to serve from creates a single point of failure between the Web site and its Internet provider.

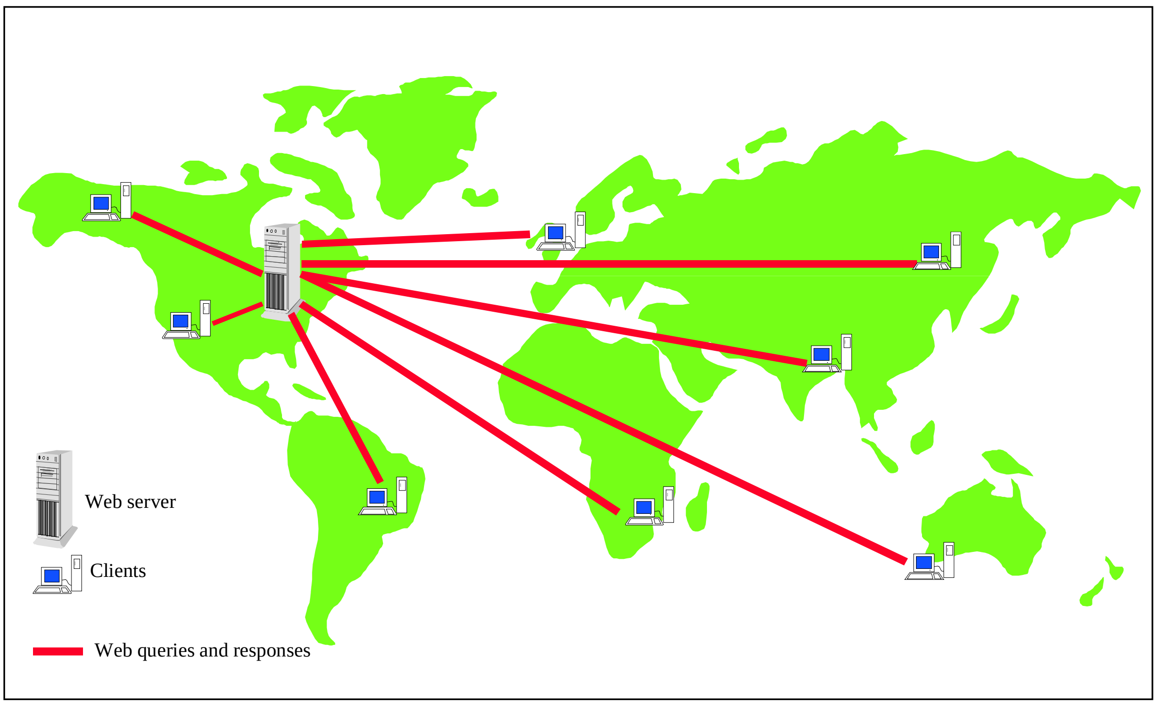

Figure 1.1: The Common Web Server Model.

Most existing Web replication architectures involve a cluster of servers that reside at the same site. These architectures improve performance by sharing the load between the different replicas, and improve availability by having more than one server to fulfill data requests. However, replicated servers alone cannot address the performance and availability problems embedded in the network itself.

The Walrus (Wide Area Load Re-balancing User-transparent System) system resolves these problems via a Web replication system where each of the replicas may reside in a different part of the network. The client Web browser automatically and transparently contacts the best replica, taking into account:

- Network topology – which replica is “closest” to the client, network-wise.
- Server availability – which servers are currently active.
- Server load – which server is currently able to return the most rapid response.

Popular Web sites that implement this system will result in better response times and higher availability of data from these sites. Equally important, this system will potentially cut down a significant fraction of Internet traffic, freeing bandwidth for other uses.

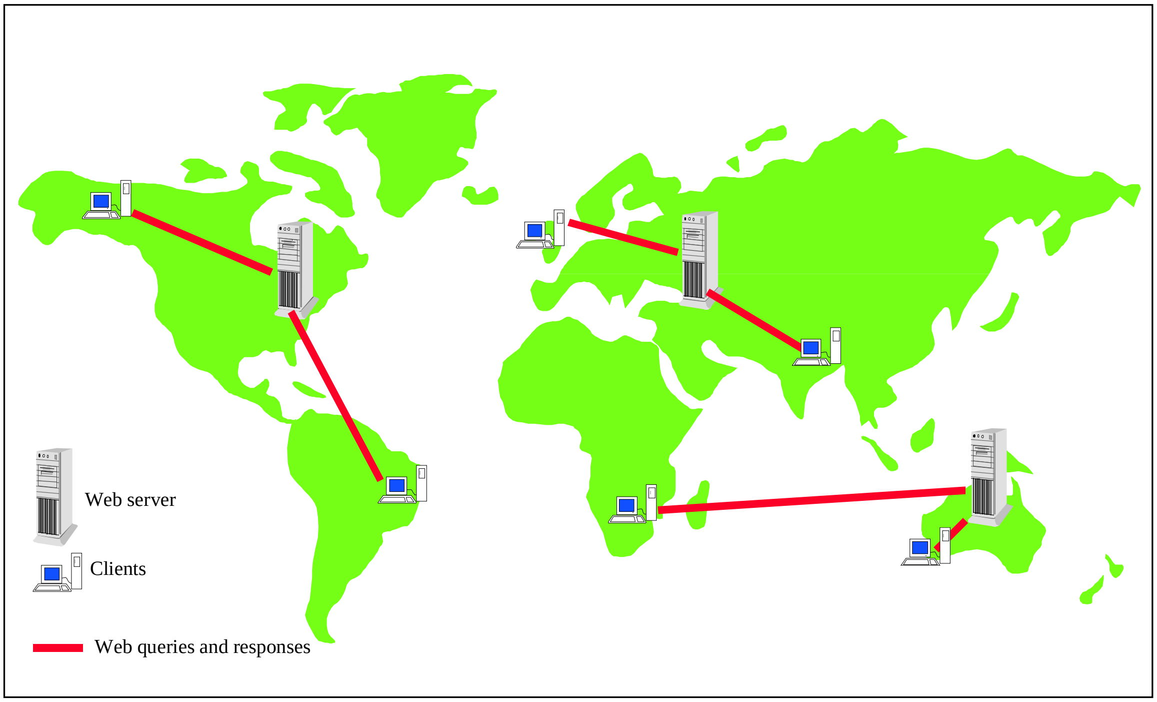

Figure 1.2: The Walrus System Model.

2. Related Work

The two most common methods used today to alleviate problems related to slow Web server responsiveness are caching and replication.

2.1 Caching

Systems that employ caching use one main Web server to store data. On demand by the client browser, a caching process makes copies of requested data closer to the client. The cache then serves subsequent requests for the same data, which obviates the need to re-access the main Web server. Data items have a "Time To Live" (TTL), which allows items to "expire" from the cache. This prevents the cache from serving stale data.

As it is most frequently implemented on the Internet today, Web caching uses a special proxy on either the client or the server side. This proxy acts as the go-between for the browser and the server. Client-side caching is when a client makes all requests for data through the proxy. The proxy then makes the actual request and caches the response for other clients (who are presumably at the same site as the original client). Many browsers, including Netscape's Navigator and Communicator and Microsoft's Internet Explorer, support this capability internally and perform client-side caching on a per-user basis. This per-user caching is independent of any other caching done at the server or client side. Some caches used on a per-site basis include the Squid [Squid], Harvest [CDNSW95], and Apache [Apache] caches.

The Squid caching system is a hierarchical cache where the parent caches are intended to reside close to the main transit points on the Internet. If a child cache does not have an object, the request is passed up to the parent, who fetches the object from the master Web server, caches it itself, and sends it to the child. The child, in turn, caches it itself and sends it to the client that requested it (which could either be a child of the child cache or a browser). Squid also supports the notion of "sibling" caches to spread out the load on the child caches. The Squid cache is based on the cache from the Harvest project. The Harvest cache is one part of a set of tools to gather, organize, replicate, and cache information on the Internet.

The Apache cache is a simple proxy-based cache. All browsers at a site may use it as a proxy, and it retrieves documents for the browsers and caches them. As the Apache cache is integrated into the popular Apache Web server, it is a commonly used solution for single sites that do not need to participate in a large caching hierarchy.

In server-side caching, one or more caching servers act as front ends to the main server. When a user requests a document, the caching server attempts to serve the document from its own cache, or from that of another caching server, before resorting to retrieving the document from the main Web server. Frequently accessed documents will therefore be quickly and widely cached, and thus the load for serving them will not fall onto the main Web server. The server and client-side caching methods do not conflict with one another, and they may be used singly or together as needed.

An intrinsic limitation of caching arises when the client sends information along with a request to the Web server for more than a static page (e.g., CGI scripts, or other similar server-side programming). Since the Web server's response to the request is dependent on the parameters sent along with the request, the Web server's response cannot be cached on either a caching server or client machine.

Freshness of data also limits the efficiency of caching. Given a particular document, data may be found in any or all of numerous caches between the client and the server. This creates a consistency problem as there is no to way guarantee the validity of any particular cache. No cache can guarantee complete freshness of data without a built in invalidation mechanism, updating invalid cache entries, or waiting for the expiration of invalid cached data. If this lack of knowledge of the consistency between the cache data and that of the main Web server is unacceptable, it quickly defeats the purpose of a caching system, as every request will need to be served, however indirectly, by the main Web server.

Historically, caching has been a poor response to the load problem because the cache management abilities of HTTP 1.0 [RFC-1945] (and earlier versions) were poor.

This has been rectified in HTTP 1.1 [RFC-2068], and Web document authors, as well as Web server managers, now have the ability to exercise considerable control over the caching (or non-caching) of their documents. However, HTTP 1.1 does not yet have universal support.

2.2 Replication

Web replication, sometimes referred to as mirroring, seeks to bridge this gap by duplicating the Web server, including its data and any ancillary abilities. A user can access any one of the replicas, as any of them will provide a valid – and presumably identical – response. Replication addresses some of the problems created with non-cacheable server requests, including those that require client-server interaction such as CGI scripts, database lookups, and server-generated pages.

Moreover, replication correctly handles situations such as advertising, which require that the system keep an accurate count of server requests. Under these types of conditions, caching must be disabled since the individual caches may not always be under the Web server’s control (as in the case where caching is performed by the browser), so there is no way to count the number of actual hits [Goldberg].

There are two commonly used forms of replication. Primary Backup (a example of which is detailed in [TM96]), where one “master” server is simply duplicated to form replicas, and Active Replication (an example of which is detailed in [Amir95]), where if any of the replicas are modified, the changes will propagate back to all of the replicas. There are no “master” servers in Active Replication.

A problem that arises in both caching and replication is that there is no immediate way for the client to determine and select the “best” server when requesting data. This paper proposes a method to address this problem. This replication-based method which transparently routes the client to the most ideal server is applicable to both replicated and cached servers, as the benefits for connecting a client to the “best” replica are similar, though not identical, to those for connecting the client to the best cache.

3. The Walrus System Architecture

The Walrus system is designed in four separate pieces: The Replicator, the Reporter, the Controller, and the Director. The fact that these components are separate ensures future expandability of any of the pieces with minimal effect on the remaining three.

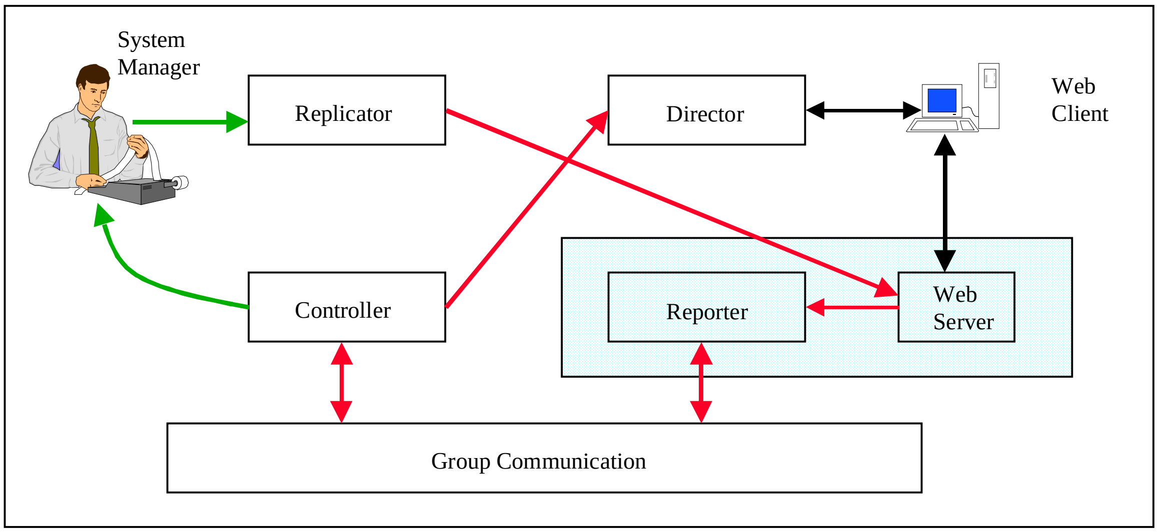

Figure 3.1: The Walrus System Architecture.

For communication between its various segments, the Walrus system uses the Spread Multicast Toolkit [Spread]. Spread is a cross-platform reliable multicast and group communication system.

3.1 Design Goals

Walrus is designed to be platform independent. With that independence in mind, Walrus was implemented in C, with special attention paid to ensure platform independence, and thus the basic source should be usable on any POSIX [POSIX-1003.1] compliant operating system.

A major design goal of the Walrus system was that it would function without requiring any modification of the network infrastructure. Walrus uses readily available building blocks in the form of standard unmodified name (DNS) and Web servers.

In addition, Walrus employs the most common "differing fiefdom" model of the Internet today. It does not require any degree of trust between differing sites to work, nor does it require that different sites run the same Web server, name server, or operating system software.

Walrus Terminology

Each geographical area that is to be controlled by Walrus has one name server. In the Walrus system, this is known as an ***area***. Each area has 1 or more Web servers assigned to it, collectively known as a ***cluster***. Thus every area is assigned to a cluster, and every cluster is located near an area. If an area is assigned to its nearby cluster, then that cluster is ***home***.

A ***threshold*** is a value, above which a server is deemed to be overloaded and not to be used for new clients except in case of dire need. Naturally, in a choice between no service at all, or overloaded service, Walrus will elect to use the slower – yet still functional – overloaded server.

A cluster is ***assigned*** to an area if the name server for that area points to one or more of the Web servers within this cluster.

3.2 The Replicator

To ensure that all of the different Web servers return equivalent responses, they must be kept in a consistent state. The Replicator manages this task by reconciling changes among all of the replicated servers. Details of the Replicator implementation used in Walrus are described in Section 4.

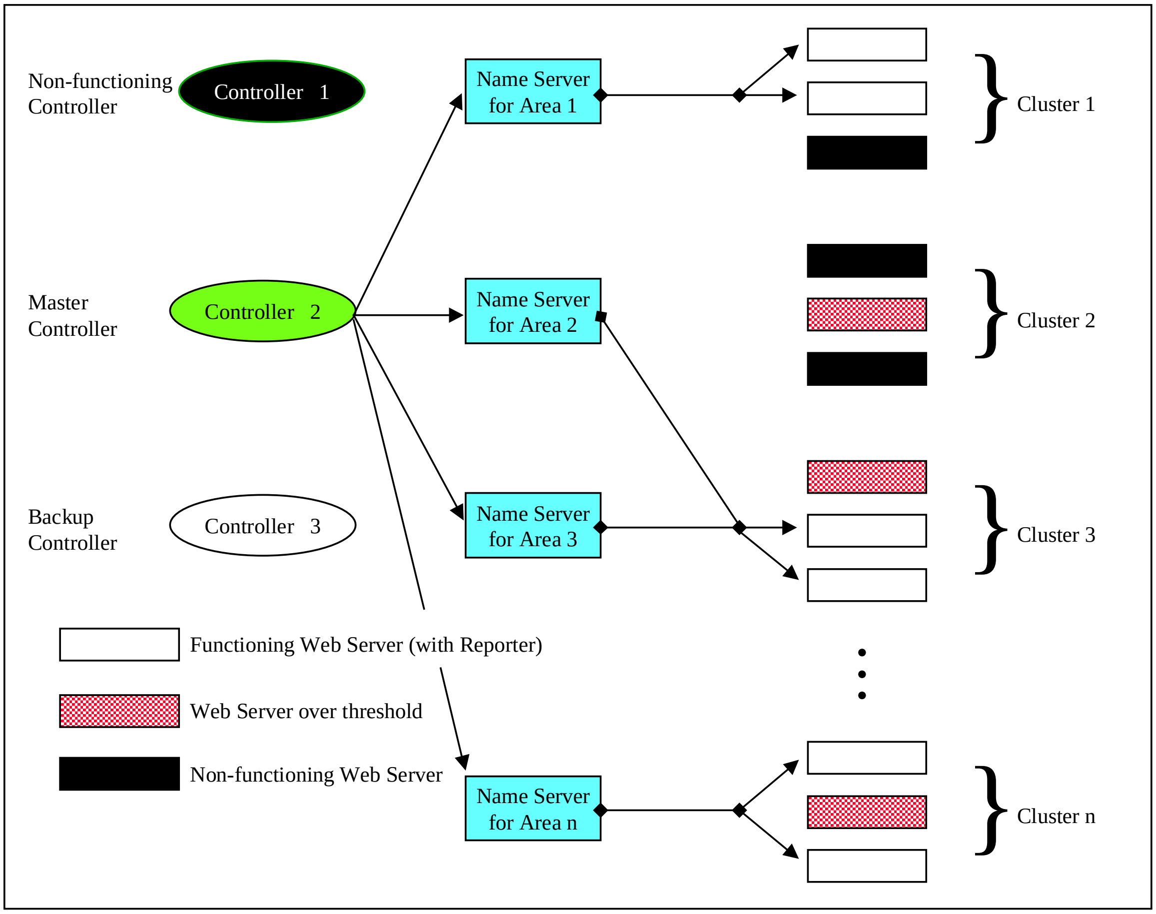

Figure 3.2: The Walrus System in Action.

3.3 Information Gathering – The Reporter

Every Web server has a background process (or “daemon”) running on it known as a ***Reporter***. The Reporter monitors the status of the local Web server and reports it to the rest of the system so decisions on how to balance the system can be made. Details of the Reporter implementation used in Walrus are described in Section 5.

3.4 Decision Algorithm – The Controller

In the Walrus system, there are one or more programs known collectively as ***Controllers***. These programs use the information the Reporters send to control which parts of which cluster are in use, and which parts should (for reason of load or otherwise) be taken out of use. The location of the Controllers does not have to be related to the different areas.

To prevent multiple incarnations of the Controller from interfering with each other, only one Controller may be active at any one time (see Section 6.3 for the mutual exclusion protocol used). There may be any number of non-active Controllers which will function as backups against the failure of the main Controller.

As a practical matter, optimum service occurs when there is at least one Controller in each area. This would ensure that if a portion of the network becomes isolated, every area is still guaranteed to have a Controller. For convenience and efficiency in performing updates, this Controller can run on the same physical machine that provides name service for that area. Details of the Controller implementation used in Walrus are described in Section 6.

3.5 The Director

The ***Director*** defines the method used by the Controller to cause the client browser to go to the appropriate server. There are many possible methods for directing the client to the correct server. However, to ensure the system can be used by the widest range of users, it is crucial that any method chosen must be completely transparent – the client must not be forced to do anything special to benefit from the system. To allow for the extremely wide range of software and hardware in common use on the Internet today, the Director cannot in any way be system or browser dependent. Details of the Director implementation used in Walrus are described in Section 7.

3.6 The Communications System – The Spread Group Communication Toolkit

Walrus uses the group communication capabilities of the Spread Multicast and Group Communication Toolkit to communicate between the Controllers and the Reporters. Spread is a system that provides multicast and group communications support to applications across local and wide area networks. It uses a simple but powerful API to simplify group communication issues

Spread employs a group paradigm for its communication framework. Processes can join or leave any number of groups at will. A process may multicast a message to any group or set of groups, and all of the processes that are members of at least one of these groups will receive the message. Spread conforms to open group semantics where a process does not have to be a member of a group to send to it.

Walrus uses several of Spread's capabilities. The Controller sends a message to a special group that only the Reporters join to request their current status. All operational Reporters receive this message. The Reporters, in turn, send their status to a special Controller group. All operational Controllers will receive this message.

In Spread, messages can be sent with different levels of reliability, ranging from "best effort" to FIFO, to several levels of guaranteed deliveries. Messages can also be sent with a defined order or unordered, and this ordering holds across different group members participating in the communications.

Spread provides membership services to all processes. When a process joins a group, all members of that group will receive notification of the new membership list. This is useful to determine what other processes are able to see messages that were sent to that group. If a process fails, or is rendered unreachable via a network partition, this fact will be detected by Spread, and all other processes that share a group with the failed process will be notified.

Spread provides several additional capabilities which are not used by Walrus. For a complete description see [Spread, MAMA94].

4. Replication

In the Walrus system, the selection of one particular Web replication scheme over another to maintain the consistency of the Web servers is unimportant. Any of the commonly used primary-backup [TM96], active replication [Amir95], or lazy replication [LLSG92] techniques may be adequate to use either independently or working together with other schemes. Currently, Walrus is distributed with scripts to drive “rdist”, a commonly used UNIX-based primary backup file synchronization package, which creates or updates a simple copy of all files from one server to another. Some efficiency is gained by only copying those files that are out of date or missing on the recipient server. For non-UNIX platforms, comparable replication systems are available.

Under UNIX, when the rdist application is run on the master Web server, it uses rsh (remote shell) or a similar program such as ssh, to execute a copy of rdist on the remote server. These two copies of rdist then compare update times on all of the files in question. If any file is out of date, the master rdist sends it to the client, which then replaces the out of date copy of the file on its side.

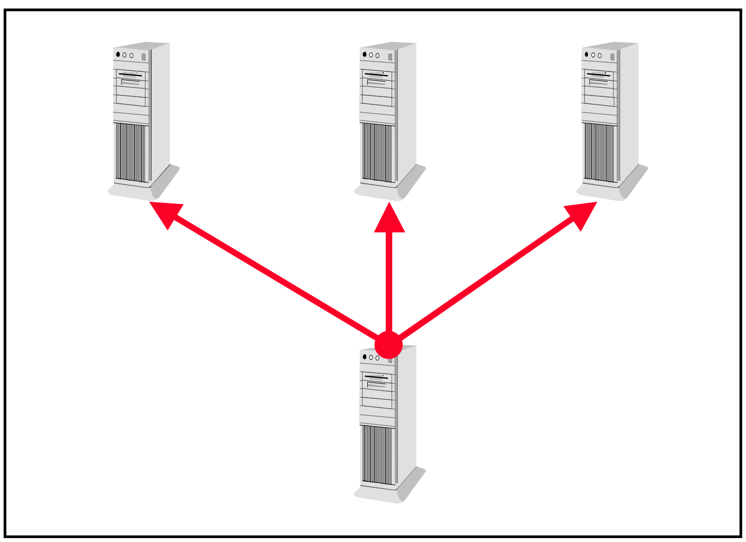

Figure 4.1: The Rdist Method of Replication.

Generally speaking, primary backup is the most appropriate replication system for the Web. Given that the most common document type served on the Web is static text, and the next most common document type is a server-side script that does not reference other data sources, the duplication of one master server is usually the best solution to the problem of keeping all of the servers synchronized.

It is important to point out that in some cases, particularly where there is a database application as a backend to the replicated Web server, this database must also be kept consistent. An active replication scheme is likely to work better in this particular situation, as it allows for modification from any of the replicated servers rather than from one primary "master" server. However, such replication is outside the scope of the Walrus system.

5. Information Gathering – The Reporter

The Reporter is comprised of three pieces. The major piece containing the code to communicate with the rest of Walrus is designed to be platform independent, while out of necessity, the other two pieces (code to read the current status from the running system, and code to retrieve statistics from the Web server process) must be written specifically for each platform and Web server respectively.

After every time unit, the Reporter gathers statistics and stores them for later use. The time unit is a configurable value, and is usually system dependent. Generally that value is set at around 1 minute, but it can go higher for systems where the reading of statistics is expensive, or lower for systems where statistics gathering is cheaper.

5.1 Information about the System

The Reporter gathers statistics from the system while it is running. Since Walrus is intentionally designed to be platform independent, these statistics are at a high enough level to obscure platform-related differences. The statistics that the Reporter gathers are as follows:

General Information

- The time duration that the statistics represent (e.g., over the last five minutes).

CPU Statistics

- The portion of time the CPU spent on user code, including the Web server and the Reporter process itself.
- The portion of time the CPU spent on system code.
- The portion of time the CPU was idle.

Note that on UNIX systems the time spent on user code includes all time spent on “niced”, or lowered priority user code.

Memory Statistics

- Free memory

  Since every type of system seems to have a different definition of "free," free memory, as used here, is memory which is either available now, or would be made available if needed, such as memory being used as cache. In simple C-based terms, if a user process could `malloc()` (request in a block) this memory, and the memory resides completely in physical RAM, it qualifies as free memory.

- Amount of memory paged out

  Since there are many different virtual memory systems in use, most with different terminology for the same concepts, this value is defined as the amount of memory that has been forced out of physical memory and onto disk due to a physical memory shortfall. Note that some systems (e.g., BSD-derived UNIX systems) also employ the concept of "swapping," which is the removal of an entire process from physical memory, generally due to an extreme memory shortfall. This statistic includes, but is not limited to, all memory lost via swapping.

- Amount of free virtual memory space

  This is the amount of free space on the virtual memory device (usually a disk). This is a crucial value, as most systems will panic and fail when virtual memory space is exhausted.

Process Information

- The number of runnable processes

  Even though only one process can be running at any given moment (barring more than one processor), this is the number of processes that could theoretically be running as they are not sleeping or blocked for any reason. This statistic roughly compares to what is known in UNIX as the load average. In an architecture that does not use process-level accounting, this can just as easily be the number of runnable threads, or any other similar concept.

- The number of blocked processes

  These are processes or threads that are waiting for some sort of I/O – such as disk or network access – to complete. Presumably, once this I/O is complete, the processes would become runnable.

  Note that this number does not include "sleeping" processes. A sleeper is sleeping on its own merits and is not waiting for any system resources.

  In addition to the sleeping process, many BSD-derived UNIX systems also have an "idle" process. These are sleeping processes that have been sleeping for more than a certain period of time, and thus become better candidates for virtual memory activity if that becomes necessary. As an idle process is just a long-time sleeping process, and Walrus does not deal with sleeping processes, it also does not deal with idle processes.

5.2 Information About the Web Server

The Apache Web server [Apache] is distributed with an optional module known as "`mod_status`". This module returns the status of the local server.

The Reporter makes a connection to the Web server running on the local host, and retrieves the information which is then parsed and rendered into the form used by Walrus.

The items returned are:

- The number of Web requests ("hits") served.
- The number of busy servers – how many requests are being served at that moment. This can be thought of as the Web server's "load average."
- Maximum number of servers – this is a configuration command in the Web server that roughly corresponds to the question of how many requests can be served at once. This may be a software licensing issue, or a physical limitation of the server hardware or software, or both.

5.3 Additional Information

In addition to the items discussed above, the Reporter can return three special replies to notify the Controller that it must address special circumstances:

- TEMPFAIL

  This is returned if the Reporter could not gather the statistics for some reason. The proper action for the Controller to take under these circumstances is to continue to use the statistics gathered in the last round. This message is most commonly returned when the Reporter has just started up and the statistics have not yet stabilized.

- BYPASS

  This is returned by the Reporter to request that the Controller take this server out of consideration for this round. BYPASS occurs if the Reporter knows something crucial that cannot necessarily be told to the Controller via the usual statistics. For example, the BSD UNIX version of the Reporter will request BYPASS if there is any swapping activity. Since swapping only happens in case of an extreme memory shortfall, the Web server is in serious trouble and should not be used.

- PANIC

  This message is similar to BYPASS, and is treated the same way by the Controller with the difference that it also notifies a human being. PANIC is to be used only in case of extreme trouble that requires outside attention. For example, a Web server process failure would result in a PANIC.

5.4 Getting this Information to the Controllers

The Spread Multicast and Group Communication Toolkit [Spread] is used to return the collected statistics to the Controllers. Although Spread was originally incorporated into Walrus to benefit from Spread's group membership capabilities (see Section 3.6), using a group communication system also simplifies the task of returning this data without the Reporter knowing which Controller is currently the master. The Reporters need just send the data to a particular group, and the current master Controller – which is the only one that can receive the message, as per Section 6.3 – will receive the data.

6. Decision Algorithm – The Controller

To determine the Controller's action for each round of reports, Walrus uses a four phase algorithm. The goal of the algorithm is to ensure that all three of these design principles are followed in order of importance:

- Every area must have at least one functioning server.
- As much as possible, every area should use Web servers that are local to it.
- As much as possible, all of the other areas and Web servers should be load balanced.

6.1 The Decision Algorithm

Phase One: *Problem Notification*

For every server returning PANIC
    notify a human being.

Phase Two: *Locality Preference*

For every area whose home cluster has at least one functioning server below the threshold
    assign it to this home cluster.

Phase Three: *Ensure Availability*

For every area that is pointing to a cluster with no functioning servers
    assign it in to a functioning cluster in a round-robin, starting with the best cluster.

Phase Four: *Load Balancing*

For every area still unassigned
    assign it to an unassigned cluster, matching the worst area to the best cluster.

Figure 6.1: The Decision Algorithm.

Phase One: Problem Notification

Phase One of the algorithm checks to see if the Web servers are trying to notify the Controller of a problem by iterating over every responding server to see if any server has requested either BYPASS or PANIC. When either message is returned, Walrus treats the server as if it was dead and non-responsive. In the case of a PANIC, the Controller notifies a human being in charge of the server about the problem.

It might seem odd that a human being in charge is notified in the case of a PANIC, but not in the case of a dead server. The reason for this is that due to the nature of the Internet, a "dead" server may only be temporarily unreachable and not truly dead at all. Because no system can tell the difference between a dead server and an isolated or unreachable one, Walrus does not notify the controlling user in this case. In addition, given an unstable network connection, a server could appear to oscillate between dead and live many times per minute, which could cause many spurious alarms. A possible future modification of Walrus would be to implement user notification after a server is dead for a period of time.

Phase Two: Locality Preference

Whenever possible, Walrus assigns each cluster to its home area. This is to ensure that if at all possible, clusters will be bound to their geographically closest area. This assignment is considered possible if at least one of the servers in the cluster is below the threshold value defined earlier.

Phase Three: Ensure Availability

During this phase, Walrus ensures that every area is assigned to a functioning cluster. If there are any areas that are assigned to clusters with no functioning servers, they will be assigned to functioning clusters in a round robin, starting with the best current cluster. This happens even if the clusters were already assigned or are over threshold – the goal is to ensure that every area has at least one functioning server associated with it. As before, it is better to be overloaded than non-functional.

Phase Four: Load Balancing

For every area that was not assigned in Phase Two or Phase Three, starting with the worst area, this phase attaches it to the best cluster that is still unassigned, assuming such a cluster exists and assuming it is worthwhile to do so. To make the assignment worthwhile, the candidate cluster must be performing significantly better that the original cluster in order to offset the DNS updating cost. This phase will improve the efficiency of the system by assigning the most used area to the least used cluster.

6.2 Oscillation and Conservatism

Rapid oscillation may occur in some load balancing environments when one cluster has a light load, so every area is assigned to it, driving up its load. Then every area is re-assigned to a different cluster due to the induced high load on the original cluster, and the process repeats itself.

The Load Balancing phase of the Walrus decision algorithm is deliberately conservative in its assignments to prevent this oscillation problem. At most, one area will be assigned to each cluster during Phase Four of each round.

6.3 Mutual Exclusion

The Controller is a crucial component of the Walrus system, and its failure would be problematic for the efficient functioning of Walrus. The incorporation of multiple Controllers into the system is used to avert this. To prevent one Controller from interfering with another, a feature of the Spread Multicast and Group Communication Toolkit is used. As mentioned in Section 3.6, all Controllers listen on a particular multicast group for the responses from the Reporters. When a new Controller comes on-line, or an existing Controller is taken down or fails, all Controllers will receive a membership message from Spread on the Controller group. As Spread maintains a consistent order in the membership list, all Controllers can immediately detect their status from their position in the list. The "master" Controller, which is the only one to pass its decisions onto the Director, is determined by being the first group member on this list.

7. The Director

The Walrus system uses the DNS Round Trip Times method we proposed in [APS98] to transparently direct the user to the correct server. A brief recap of this method, which uses the domain name system (DNS) [RFC-1034, RFC-1035] follows:

7.1 DNS Resolution

- Every server on the Internet has a fully qualified domain name (FQDN).

- FQDN are usually read from left to right, and each segment is rooted in the segment after it.

- Thus, "www.cnds.jhu.edu" is a machine known as "www" located in the Center for Networking and Distributed Systems (CNDS), which is part of the Johns Hopkins University (JHU), which is an educational (EDU) institution in the US.

- There is an assumed dot (".") at the end of each FQDN, which stands for the root domain at the top of the domain name tree.

- Barring any information cached on a local name server, a query for www.cnds.jhu.edu is sent to one of the root (".") name servers.

- This server, which due to the enormous size of the entirety of the domain name system cannot know every address, returns a list of domain name servers for "edu." Then the process repeats, with one of the "edu." servers returning a list of name servers for "jhu.edu." and one of the "jhu.edu." name servers returning a list of name servers for "cnds.jhu.edu." one of which then returns the requested IP address(es) for "www.cnds.jhu.edu."

- Unless specifically configured not to, all name servers aggressively cache the responses to any queries that pass through them.

- This caching is used to reduce the difficulty of the problem. If at any stage, a server has already cached he address being requested (or the address of a name server authoritative for that domain), it will return it directly, short-cutting the above process.

- If a query is made for a name with more than one address associated with it, all of these addresses are returned by the server.

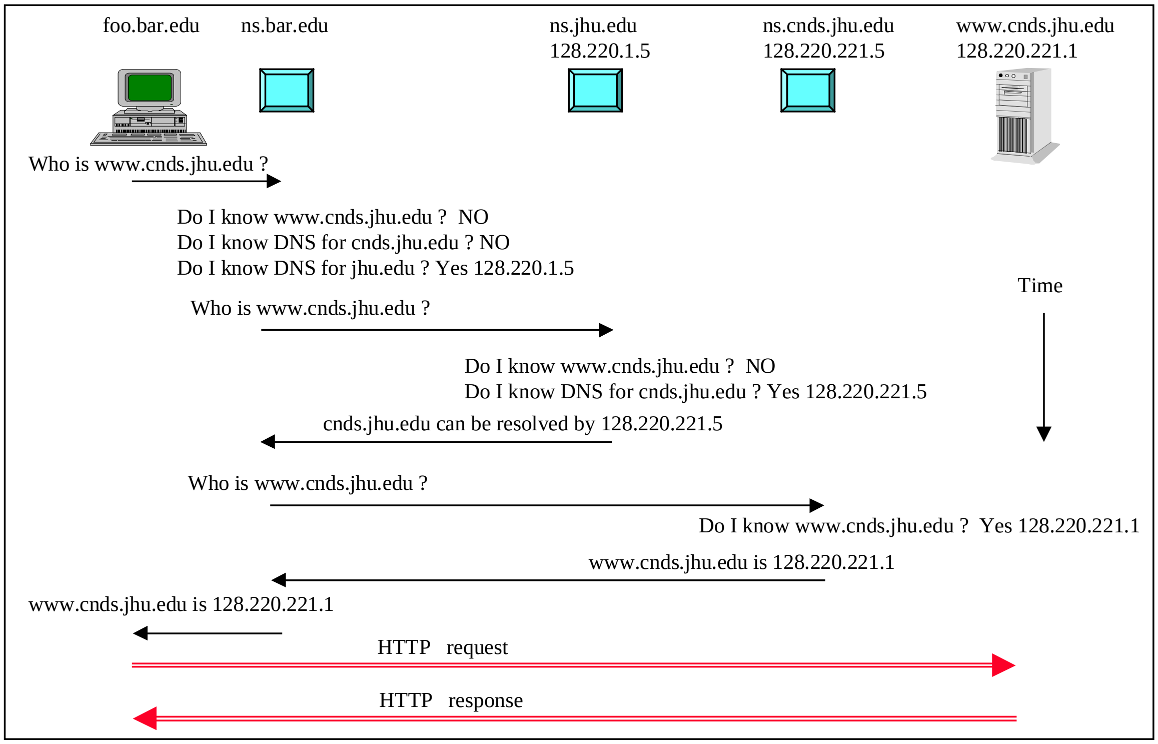

Figure 7.1: The DNS Resolution Process.

7.2 Using Multiple Authoritative Name Servers

The DNS Round Trip Times method takes advantage of the fact that each local server, when querying a remote server, tracks the round trip time (RTT) of packets to that server. This is done for optimization reasons, as over time the local server will favor those servers that respond the quickest. Given otherwise similar servers, this favoritism usually results in selecting a server that is closer network-wise.

This optimization can be leveraged for the Web server selection problem by changing one concept in the DNS. Normally, all servers for a particular domain (called a "zone" in DNS parlance) carry the same data, but this is not a requirement. From the perspective of a querying server, all that is significant is whether the remote server answers "authoritatively", as well as being in the chain of authority for the domain the querying server is interested in. An "authoritative" answer indicates that the data in question came from the name server's own configuration, rather than having been cached from another server.

Thus, under the example set above, where multiple name servers exist on a network, each serving an authoritative, but different IP address for www.cnds.jhu.edu, the selection of the Web server to respond to a client's query depends on which authoritative name server the client's local name server happens to ask. Since, as already established, the client's local name server tends to favor those remote name servers that are closest to it, by using the dynamic update facility of DNS [RFC-2136] to load the name server closest to a particular client with the IP addresses of Web servers that we wish that client to use, an excellent system for forcing a client to use a particular Web server emerges.

It can be pointed out that using DNS optimization gives a good idea of how close the chosen name server is to the client's name server, but that may say nothing about where the Web server is located relative to the client itself. It is assumed that when the system is set up, the chosen name server and the Web server will be placed in close network proximity. In addition, the local name server the client is querying is expected to be in close network proximity to the client, as is the usual setup on the Internet today.

7.3 Convergence

The RTT method DNS uses to determine the best name server for a particular zone is not necessarily optimal, but over time is a reasonable approximation of optimality. The time it takes to establish this reasonable approximation is the chief problem with this method. In order for the local name server to query the fastest responding authoritative name server, it needs to try them all at least once to determine which will return the fastest response. Thus, until all authoritative name servers for the remote zone are queried at least once, the local name server may query an authoritative name server that is very far away network-wise, and thus get the IP address of a Web server that is very far away from the client. The local name server is unlikely to repeat its error as this distant authoritative name server will likely have a larger RTT value than the other authoritative servers in the list, but for this one connection this DNS-based system will not be effective. It should be emphasized that in no case is this "harm" greater than the contacting of a single-location server that is located in the remote area.

Due to the necessity for the client's local name server to cycle through and establish timings for all of the authoritative name servers for the zone in question, the Web server must receive a fairly large quantity of hits from a relatively concentrated area (generally one institution such as an ISP or a university, which would have a set of local name servers) for this method to work. Thus it is fairly unlikely to be an efficient Director for less busy servers over the short term. It will, however, work in the long term, once the local name servers learn the distances involved via contacting the remote authoritative name servers for the zone in question.

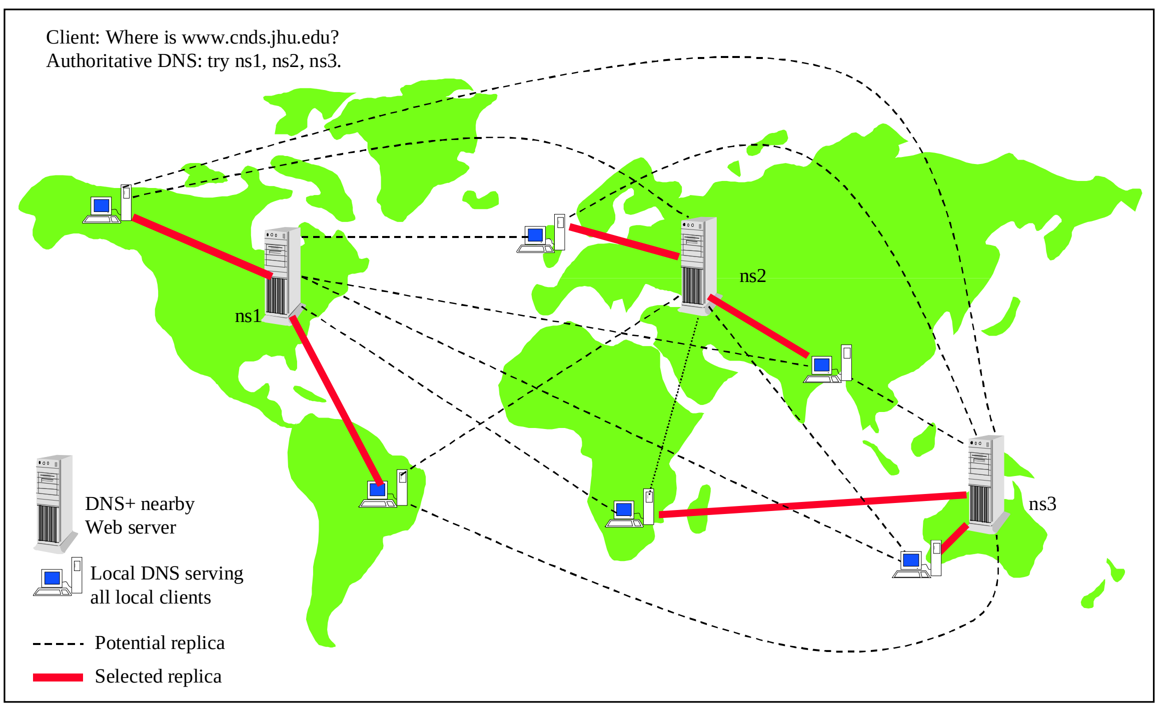

Figure 7.2: The DNS Round Trip Times Method in Action.

Figure 7.2 illustrates the DNS Round Trip Times method. The server www.cnds.jhu.edu is replicated in three places, and has an accompanying name server in each place. Clients that want to query this Web server will ask their local name server to resolve the name. The local name server will eventually try all three of the name servers that are authoritative for www.cnds.jhu.edu. Ultimately, the local name server learns which remote name server is closest to it and concentrates on the Web server that this name server points to.

7.4 Using Time To Live Values to Speed Convergence

Another concept from DNS can be used to help the convergence process along. In DNS, each name-to-address mapping can have a variable time-to-live (TTL) value. This value specifies a time, after which the address returned should be regarded as invalid, and should not be used again. Setting a lower than usual TTL on the Web server's address record can force the name to be re-requested with greater frequency. Of course, this can be a disadvantage as well, as each additional request causes extra traffic and occasionally a perceptible delay to the client, as the name is re-resolved. A period of around 5 minutes is recommended as the limit for this value, as it is short enough to cause a frequent refreshing of the data, but yet is not so short as to cause excessive network traffic to re-resolve the name.

Until convergence takes place, one connection (defined as one or more "hits" to the Web server within a certain TTL, from a client using a particular local name server) has a 1/n chance of getting the "correct" server, where n is defined as the number of name servers serving that zone. Therefore, given the characteristics of the zone, one can estimate the optimal number of name servers for that zone. For example, a very popular site that gets continual connections from around the world such as Yahoo or CNN could support many more name servers than a site that gets a slower stream of connections.

The larger number of connections that a busy site will receive can minimize the delay of being forced to try all of the name servers for a zone before optimizing on the best one. However, beyond a certain level of traffic (at least one connection per TTL value), a site with fewer authoritative name servers will converge faster as the local name server will have fewer servers to check.

The TTL value may also be tuned to give the best results in the particular environment where it is used. Note that the TTL time period is timed relative to the local name server, and not to the client or the Web server – if several clients sharing a local name server all request a document from the same Web server within the time period, that still counts as one "connection."

8. Additional Considerations

8.1 Experience

Some tests of the DNS Round Trip Times routing method were done for [APS98]. This experience produced the following recommendations:

- The time it takes for the DNS system to converge on the best authoritative name server increases linearly with the number of name servers. At the same time, the law of diminishing returns dictates that the added benefit derived from each additional name server decreases as more name servers are added. Since the system is not effective until it converges, using a smaller number of areas (each of which requires at least one authoritative name server) will provide the best overall performance.

- The name server that delegates authority to the multiple authoritative name servers must not simultaneously be used as a local name server for users. Where such misuse occurs, that name server would soon learn which replica was best for its own users, cache the answer, and then return that answer to any remote name server. This distorting effect will only last until the TTL of the Web server's address expires, but can happen any time that name server is used by a local client to look up the address of the replicated server.

8.2 Deployment Tradeoffs in Walrus

There are some efficiency tradeoffs learned while implementing Walrus:

DNS Setup

To maximize the efficiency of using the DNS Round Trip Times method as a Director under Walrus, the following setup is suggested:

- Every area name server is authoritative for its own zone (e.g., area1.cnds.jhu.edu, area2.cnds.jhu.edu, and so on), as well as authoritative for the zone containing the address that the Walrus system is balancing for (e.g., "www.cnds.jhu.edu").

- When a cluster is assigned to an area, it is assigned to a name under the area name (e.g., "www.area-1.cnds.jhu.edu", "www.area-2.cnds.jhu.edu", etc.).

- Each area name server has an alias (called a CNAME) pointing the balanced address to the local name ("www.cnds.jhu.edu" to "www.area-*n*.cnds.jhu.edu").

- This extra level of redirection minimizes network traffic when updates occur, since otherwise the update data would have to reach every one of our authoritative name servers on the net. If every area is authoritative for a different zone, the update only needs to reach one name server.

Area Setup

As discussed in Section 8.1, there are some inefficiencies when there are a large number of Walrus areas in use. We recommend that Walrus areas are only placed in relatively widespread geographical areas. This will limit the slow convergence time from becoming excessive.

Cluster Setup

Similar to the limits on areas, there are also practical limits on how many servers can reside in each cluster. The reason for this particular limitation is that some DNS resolvers in use today cannot properly deal with a DNS reply from the name server that contains too many addresses. Most DNS replies are UDP-based. If a reply cannot fit in one UDP datagram, the client's resolver is supposed to retry the query using TCP. Unfortunately this takes extra time, and worse, some resolvers in use today will not do this, and merely fail. Therefore, we recommend testing to ensure that all server addresses for each cluster can fit within a single DNS reply datagram. There are many tools (for example, "dig" on UNIX platforms) that can perform this test. A good starting point would be a maximum of between 20 and 30 Web servers in each cluster.

If it is absolutely necessary to have more Web servers than will fit into one DNS reply datagram, the collection of Web servers should be split, and a different area added to cover the other servers.

8.3 Failure Modes

Walrus is designed to automatically handle system failures. If a Web server fails, but the machine it is running on remains functioning, the Reporter will return PANIC to the Controller, causing the System Administrator to be notified. If the whole machine fails, then it will not respond and thus is automatically excluded from consideration by Walrus, and will not be used until it recovers.

A name server failure is also handled automatically and transparently by the name server system – in the case of a non-responsive name server, other servers for that zone are queried in RTT-order until an answer is received. In other words, the users that would have queried that server will automatically and transparently fall back to the next-best server to query.

If a Controller fails, there can be any number of backup Controllers waiting to be promoted to the primary Controllers.

9. Conclusions

This thesis presents Walrus, a new method for utilizing multiple Web servers across a geographically distributed area, and automatically arranging them to guarantee service, efficiency, and load balancing. Walrus will continually seek to improve the speed and reliability of the Web servers that are presented to the public, but the public does not need to do anything special to benefit. Indeed, the public does not even have to know that the Walrus system is being employed to benefit from it.

The Walrus system is a significant improvement over the current method of using one server (or cluster of servers) to cover the entire planet. Even in its worst case scenario, Walrus will perform equally as well as the best case of the one server method.

There are many possibilities for future growth of the Walrus system. For example, the underlying DNS-based method used to connect the user to the correct server is not inextricably tied into the Walrus system – if necessary, a different method can be used. [APS98] discussed and compared a number of Directors which could be used in the Walrus system. One possibility is if all of the Web server replicas reside within one network, then the shared IP address scheme proposed in that paper would be a good possibility.

The shared IP address scheme works only on a single autonomous network, and cannot cross network boundaries. Using “closest exit routing,” where the destination network is presumed to know best how to find a host within that network, packets for that host are handed to its network as soon as possible. The network then uses the Shortest Path First algorithm to reach the nearest server answering to the IP address in question. To get around possible problems with routing instabilities, [APS98] recommends sharing the IP address of several authoritative name servers, which will each return a unique address for a Web server. As name servers commonly use the connectionless UDP (User Datagram Protocol) for communication, a possible routing change cannot break communication in the manner that a TCP (Transmission Control Protocol) based connection can. This is similar to the DNS Round Trip Times method, but does not require any convergence time, as the choice of which name server is closest is determined by the network routing.

Another possible enhancement would be the incorporation of the level of consistency for the Web server as well as any database backend in the data returned by the Reporter. This will allow automatic downgrading of any out-of-date server, which will result in the more up to date servers being used with greater frequency.

Last but not least, it should be noted that there is no particular requirement that Walrus be used for the World Wide Web only. In fact, the same method is directly usable by any name-based service, which includes email, news, and most other network services.

References

[Amir95] Amir, Y.: Replication Using Group Communication over a Partitioned Network, Ph.D. Thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Israel (1995). Available at http://www.cs.jhu.edu/~yairamir/

[Apache] The Apache HTTP Server Project. http://www.apache.org/

[APS98] Amir, Y, Peterson, A, Shaw, D.: Seamlessly Selecting the Best Copy from Internet-Wide Replicated Web Servers. The 12th International Symposium on Distributed Computing (DISC'98) (formerly WDAG), Andros, Greece, September 1998, to appear.

[CDNSW95] Chankhunthod, A., Danzig, P. B., Neerdaels, C., Schwartz, M. F., Worrell, K. J.: A Hierarchical Internet Object Cache. Technical Report 95-611, Computer Science Department, University of Southern California, Los Angeles, California, (1995).

[Goldberg] Goldberg, J.: On Interpreting Access Statistics. http://www.cranfield.ac.uk/docs/stats/

[LLSG92] Ladin, R., Liskov, B., Shrira, L., Ghemawat, S.: Providing Availability Using Lazy Replication. ACM Transactions on Computer Systems, 10(4), pages 360-391, 1992.

[MAMA94] Moser, L., Amir, Y., Melliar-Smith, P., Agarwal, D.: Extended Virtual Synchrony. The 14th IEEE International Conference on Distributed Computing Systems (IC-DCS), pages 56-65, June 1994.

[NSF-Traf95] NSFNET Backbone Traffic Distribution by Service report. ftp://ftp.merit.edu/nsfnet/statistics/1995/nsf-9503.ports.gz

[POSIX-1003.1] IEEE Standard Portable Operating System Interface for Computer Environments. IEEE Std1003.1-1988

[RFC-1034] Mockapetris, P.: RFC-1034: Domain Names - Concepts and Facilities. (1987).

[RFC-1035] Mockapetris, P.: RFC-1035: Domain Names - Implementation and Specification. (1987).

[RFC-1945] Berners-Lee, T., Fielding, R., Frystyk, H.: RFC-1945: Hypertext Transfer Protocol - HTTP/1.0. (1996).

[RFC-2068] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T.: RFC-2068: Hypertext Transfer Protocol - HTTP/1.1. (1997).

[RFC-2136] Vixie, P. (ed), Thompson, S., Rekhter, Y., Bound, J.: RFC-2136: Dynamic Updates in the Domain Name System (DNS UPDATE). (1997).

[Spread] The Spread Multicast and Group Communication Toolkit. http://www.cnds.jhu.edu/projects/commedia/spread/

[Squid] Squid Internet Object Cache. http://squid.nlanr.net/

[TM96] Trigdell, A., Mackerras, P.: The Rsync Algorithm. Technical Report TR-CS-96-05, The Australian National University. Available at ftp://samba.anu.edu.au/pub/rsync/