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

Yair Amir and David Shaw

Department of Computer Science
The Johns Hopkins University
{yairamir, dshaw}@cs.jhu.edu

Abstract[1]

Today, most of the popular web sites are served from single locations. This basic Web client-server model is easy to deploy and maintain and thus is very successful. It suffers, however, from several efficiency and availability problems. This paper presents Walrus, a low-latency, high-throughput Web service that addresses some of these problems. Under Walrus, a single logical Web server can be replicated to several clusters of identical servers where each cluster resides in a different part of the Internet. An important aspect of Walrus is its ability to transparently direct the web browser to the best replica without any changes to the web server, web client, and network infrastructure. "Best" is a relative term, dependent on where the client is located on the network, the load on each replica, and more. Walrus deploys an elegant algorithm that balances these considerations.

1. Introduction

Over the last few years, the Internet was transformed from a communication tool used mainly by academics to a major information source, penetrating every home and business. It is changing the way we communicate, learn, publish, purchase, and more, affecting many aspects of our lives. The technology enabling this revolution is the combination of the mature communication infrastructure and the relatively new World Wide Web. As early as the beginning of 1995, Web traffic became the single largest load on the Internet [8]. The explosive growth of Web content applications and sites shows no sign of slowing down.

The basic Web client-server model presented in Figure 1.1 proved to be extremely successful. Some of the reasons for this success seem obvious: It is extremely easy to deploy and maintain a web server. Once the web site is deployed, it can be immediately accessed from anywhere on the Internet. Control of the content of the site is completely retained by the owner of the site and changes take effect immediately (subject to caching).

The basic model suffers from several weaknesses:

- For clients of the web site:
  - ⇒ High latency is experienced when the server becomes the bottleneck.
  - ⇒ High latency is experienced when the client is distant (network-wise) from the server.
  - ⇒ No service is available if the server is down.
  - ⇒ No service is available if connectivity is lost to the part of the network in which the server resides.
- For the Internet as a whole:
  - ⇒ The network becomes congested because large quantities of data are sent repeatedly over the wide area network for clients that reside in the same locality.

In spite of the widespread acknowledgement of the weaknesses of the basic client-server model, practically all of the popular Web sites are still served from single locations[2]. Most of these sites deploy replication architectures that involve a cluster of servers residing at the same physical location. These architectures improve the performance of the logical server by sharing the load between the different replicas. They also improve availability by having more than one server to fulfill data requests. However, replicated servers in the same location cannot address the performance and availability problems embedded in the network itself: They still suffer from a single point of failure in their connection to the Internet, and their service latency is still high for clients that are distant (network-wise). Logically, their model is still the one presented in Figure 1.1, incurring the same cost of network resources.

In light of the above, it seems logical to construct a popular web site according to the model presented

[1] This research is supported, in part, by the Defense Advanced Research Projects Agency (DARPA) under grant F306029610293 to Johns Hopkins University.

[2] As far as we know, the Netscape site is the only major site that recently started to implement wide area replication with automatic direction of clients, similar to Walrus' Director. In our opinion, Netscape's site setup can be optimized by following the recommendations in Section 7.

in Figure 1.2. This model of wide area cluster replication addresses all of the above weaknesses of the basic Web client-server model.

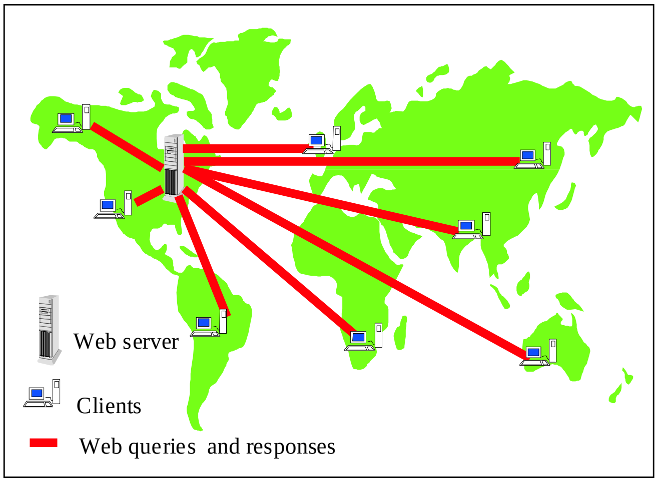

Figure 1.1 The Basic Web Server Model.

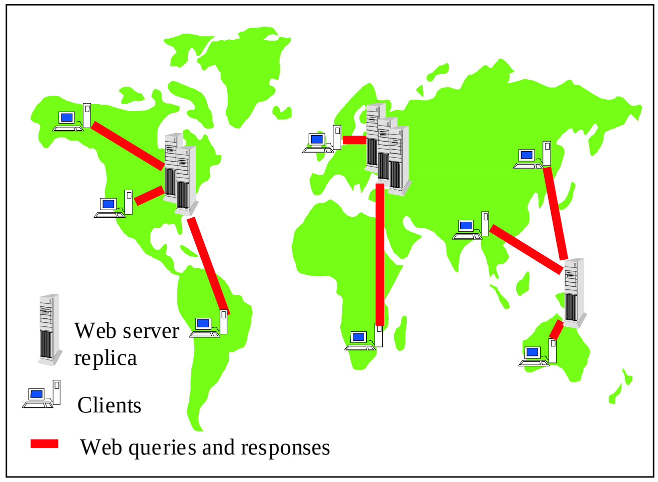

Figure 1.2 The Walrus Web Server Model

In our opinion, there are two reasons why this model is not widely used:

- It requires a complex setup and maintenance.
- It may not be clear to Web administrators how to ensure that most clients are directed to the nearest replica (network-wise) automatically.
- It requires Web administrators to obtain physical sites at different places.

Walrus (Wide Area Load Re-balancing User-transparent System) addresses the two first difficulties. It implements a Web replication system where each cluster of 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.

The key feature of the Walrus system is that it functions without requiring modification to the network infrastructure. In particular, it does not require *any* changes to the name server (DNS), the Web server, and the Web client (browser) software.

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

Related Work

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

Caching

Systems that employ caching use one main Web server or cluster 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 proxy on either the client or the server side. 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. Most current browsers support this capability internally and perform client-side caching on a per-user basis. Some caches used on a per-site basis include the Squid [15], Harvest [4], and Apache [2] 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.

In server-side caching, one or more caching servers act as front ends to the main 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 interacts with the Web server (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.

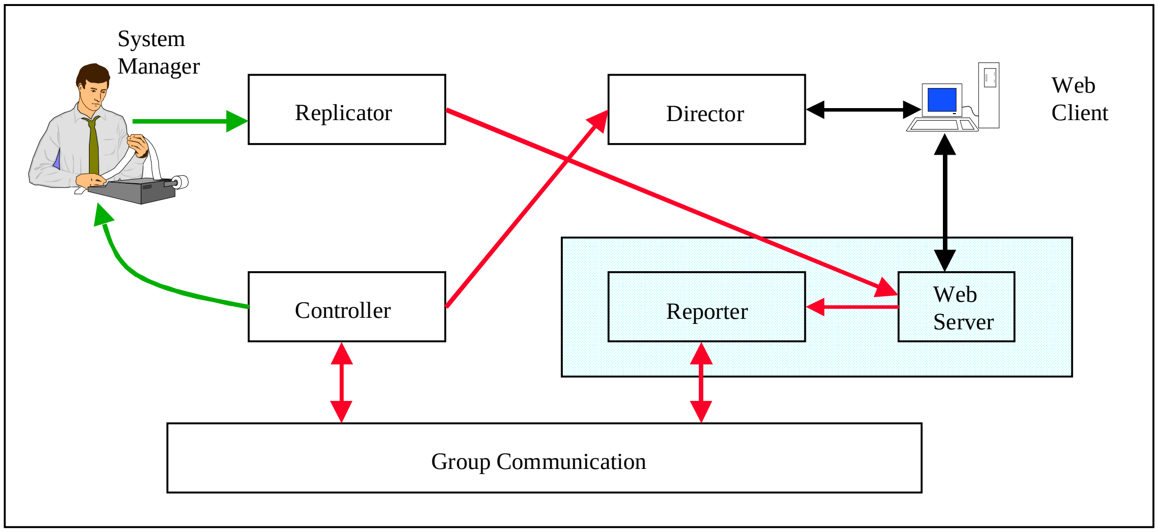

Figure 2.1: The Walrus System Architecture.

Freshness of Data is another limitation of caching. 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.

Replication

Web replication duplicates 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.

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 [5].

There are two commonly used forms of replication. Primary Backup (a example of which is detailed in [16]), where one "master" server is simply duplicated to form replicas, and Active Replication (an example of which is detailed in [1]), 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.

2. The Walrus System Architecture

The Walrus system is designed to operate in the standard Internet environment without requiring any modifications to the infrastructure. Specifically:

- Walrus does not require changes to the Web server.
- Walrus does not require changes to the Web browser.
- Walrus does not require changes to the DNS, operating system, or Internet Provider Service (ISP) setup.

Walrus employs the 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 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 [9] compliant operating system.

The Walrus system architecture is presented in Figure 2.1. The architecture contains four components: The Replicator, the Reporter, the Director, and the Controller. For communication between its various components, the Walrus system uses the Spread cross-platform wide-area group communication toolkit [14].

As depicted in Figure 2.2, 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 geographical area also contains one or more Web servers collectively known as a ***cluster***. This cluster serves as the ***home*** cluster for the local area. Each ***area*** points to one ***cluster*** (not necessarily its home cluster).

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.

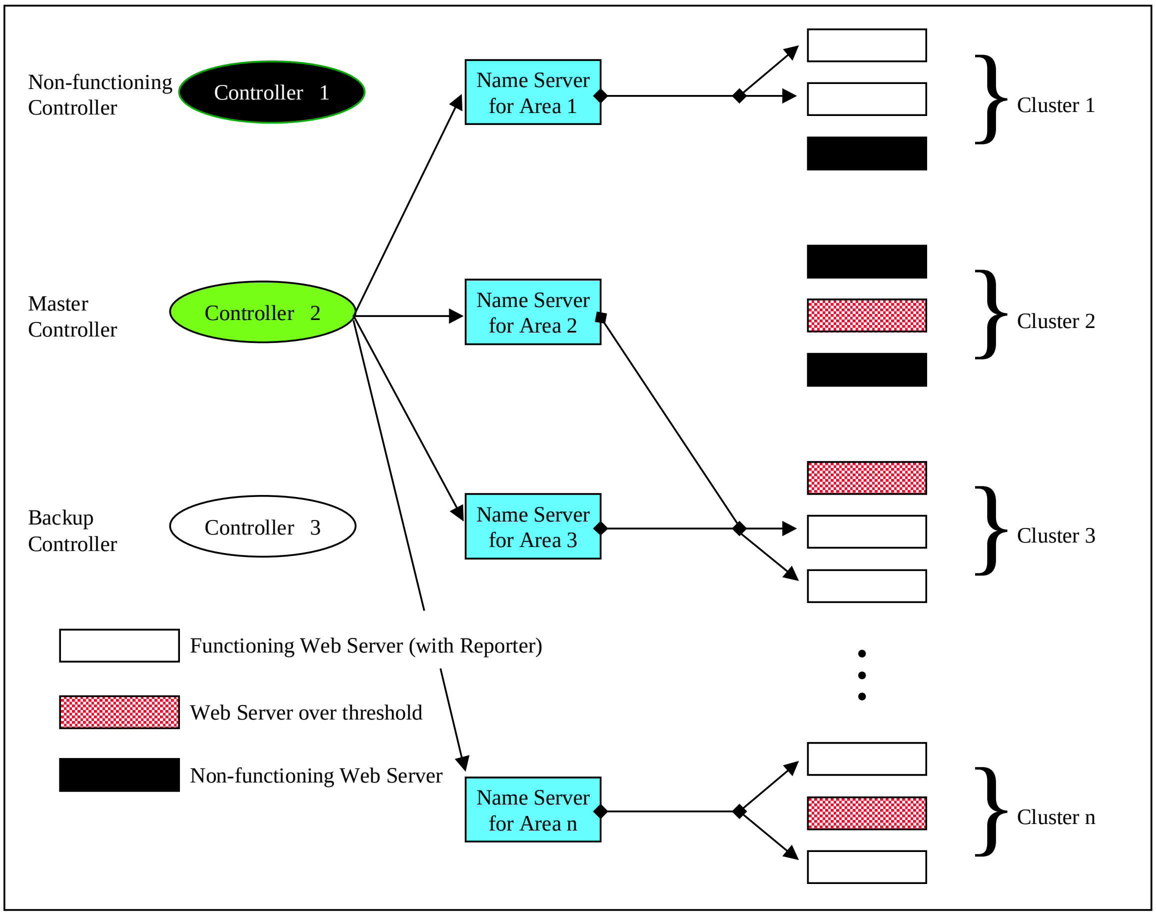

Figure 2.2: The Walrus System in Action.

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.

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.

To direct the actions of the 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 mutual exclusion handling). There may be any number of backup Controllers.

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.

The ***Director*** defines the method used by the Controller to cause the client browser to go to the appropriate server. 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. Walrus employs the standard name service of the Internet [10, 11] as its Director.

Walrus uses the group communication capabilities of the ***Spread*** wide-area group communication toolkit [14] 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 usage.

Spread employs a process group paradigm. 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.

Spread also provides membership services. 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 that are not used by Walrus. For a complete description see [7, 14].

Figure 2.2 presents the Walrus system in action. In this scenario, there are three controllers in the system but one of them is not functioning. Therefore, controller 2 is managing the system. All of the areas are now pointing to their home clusters except area 2 Cluster 2 currently contains two non-functioning Web servers and one overloaded Web server. Therefore, it is better to point area 2 to some other cluster, in this scenario, Cluster 3 was selected (based on network proximity and load). Each name server points to all of the functioning (with load less than the threshold) Web servers in the cluster assigned to them.

3. The Replicator – Content Replication

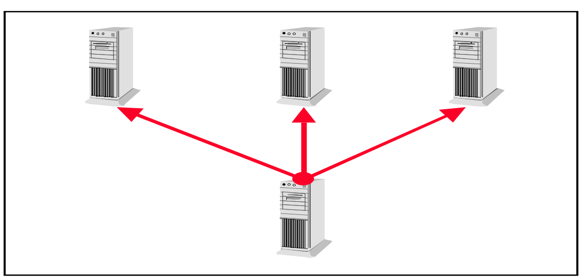

Figure 3.1: The Rdist Method of Replication.

The Walrus system uses a primary-backup with primary-push replication scheme (see Figure 3.1). The current implementation of the Replicator contains software that drives *rdist*, a commonly used UNIX-based primary-backup file synchronization package. *rdist* creates or updates a 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.

It is important to note, however, that any of the commonly used primary-backup [16], active replication [1], or lazy replication [6] techniques may be adequate.

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 (secure shell) 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.

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. For these cases, the ***Replicator*** will have to be implemented using a more sophisticated scheme that will ensure the consistency of the database. For example, the active replication technique described in [1] will work well over the wide area network, utilizing the ***Spread*** group communication toolkit [14]. Although such a technique is not detailed in this paper, it nicely fits the Walrus architecture.

4. The Reporter – Status Gathering

The Reporter is a process that resides in each of the Web server machines. It monitors the operation of the operating system and the web server system on that machine. The Reporter gathers statistics once every unit of time 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 cheap.

The Reporter connects with the Spread group communication subsystem and joins a group named "Reporters" together with all the currently active and connected Reporters in the system. Upon receipt of a request message to the "Reporters" group from a Controller, the Reporter reports its status by multicasting a message on the "Controllers" group. The Reporter continuously monitors its operating system and Web server while waiting for Controllers requests.

Using a group communication system simplifies the task of returning this data without the Reporter knowing which Controllers are active, where they reside, and which of them is the master. The Reporters need just multicast the data to the "Controllers" group (See section 6 for details).

The Reporter is comprised of three pieces. The major piece contains the code to communicate with the rest of Walrus and is platform independent. However, 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. The available Walrus distribution supports various Unix platforms and the Apache Web server.

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 gathered statistics are as follows:

- 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 operating system code.
  - ⇒ The portion of time the CPU was idle.
- Memory statistics:
  - ⇒ Free memory.
  - ⇒ Amount of memory paged out. 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.
  - ⇒ Amount of free virtual memory space. 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.
  - ⇒ The number of blocked processes (waiting for an I/O operation to complete).

Information about the Web server

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

- 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, a physical limitation of the server hardware or software, or both.

Special Web server status

The Reporter may return three special replies to notify the Controller that it must address special circumstances:

- TEMPFAIL – the reporter could not gather statistics. The controller should 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.
- PANIC - similar to BYPASS, but requires notification to a human being. For example, a Web server process failure would result in a PANIC.

5. The Director – Transparent Directing of Clients to Servers

The ***Director*** is actually the standard distributed domain name system (DNS) [10, 11] used on the Internet without any changes. The Walrus system uses the DNS Round Trip Times method we proposed in [3] to transparently direct the user to the correct server. A brief recap of this method follows. A more complete description of this method and how it is used in Walrus is given in [13].

The DNS Round Trip Times method takes advantage of the fact that each local name server, when querying a remote name 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 the best server network-wise (not taking into account server load or availability). i.e. the client will contact the best name server for it to access network-wise.

Normally, all servers for a particular Internet domain 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". 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, where multiple name servers exist on a network, each serving an authoritative, but different IP address for a particular address, 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. As already established, the client's local name server tends to favor those remote name servers that are closest to it (and currently have a less congested access).

As previously stated in Section 2, for this system to work well, each name server controlling an ***area*** has to be positioned very close to the Web servers in this area (network-wise).

When the client contacts a specific authoritative name server Walrus will know which Web servers are best for this client network-wise.

In most cases the name server controlling an ***area*** will point to Web servers in that area. However, sometimes, due to availability and load, Walrus will update the name server to use servers from another area. This is achieved by using the dynamic update facility of DNS [12].

The chief limitation of this method is the time it takes for the client's local name server to settle on the quickest remote name server. To aid this process, a low time-to-live (TTL) value can be set on the DNS information to force it to be re-requested with greater frequency.

There are many other details and limitations with this method, and readers are encouraged to see [3, 13] for a more complete discussion.

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 Assignment Algorithm.

6. The Controller – Dynamic Assignment of Areas to Clusters

The Controller is the heart of the Walrus system, responsible for making the dynamic assignment of areas to clusters. For fault tolerance reasons, there exist several Controllers in the network. The Controllers use the Spread group communication subsystem to communicate with Reporters. All of the Controllers join the "Controllers" group. In addition, the currently connected Controllers use the Spread semantics to elect a master Controller (see Section 6.3). The master Controller is the only Controller actively making assignment decisions. The rest are waiting to take over if they are elected as the master Controller.

The Controllers work in rounds. Periodically, the master Controller multicasts a request message to the "Reporters" group. After gathering the results of the different Reporters (which are sent on the "Controllers" group), the master Controller computes the new best assignment of areas to clusters. If changes to the current assignment are required, the master Controller updates the Director using dynamic DNS updates, as explained in the previous section.

6.1 The Assignment Algorithm

The master Controller uses a four phase algorithm to determine the new best assignment for each round. 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 Web 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.

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 assignment 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.

It is important to note that although the DNS update is performed instantly, its effect is not similar to the effect of process migration in a multiprocessors architecture. Users that already resolved the Web site will continue to use their assigned server (until the entry expires in their DNS). Only new users will be directed according to the current Director setting. This creates a relatively gradual transition.

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.

All the Controllers join on the "Controllers" multicast group. When a new Controller comes on-line, an existing Controller is taken down or fails, or a network partition or re-merge occurs, the currently connected Controllers receive a membership message notification from Spread on the "Controllers" 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. Therefore, there is only one master Controller in each connected component of the network.

Note that when network partitions occur, it is possible to have two or more master Controllers functioning at the same time in (different partitions). However, since the Reporters and Controllers communicate only via Spread, a master Controller considers only Web servers with Reporters that reside in the same network component with itself, as Spread determines the component. From the semantics guaranteed by Spread [7], it is not possible to have two active components containing the same group member at the same time. Therefore, each Web server can be considered and assigned by no more than one master Controller.

7. Engineering Considerations

7.1. Experience

Some tests of the DNS Round Trip Times routing method were done for [3]. 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.

7.2. 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. 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.

7.3. 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 the master Controller fails, there can be any number of backup Controllers waiting to be promoted to the main Controller.

8. Conclusions

This paper presented 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 continually seeks 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 model 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 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. More information on the Walrus system can be found on http://www.cnds.jhu.edu.

References

[1] Y. Amir. 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.

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

[3] Y. Amir, A. Peterson and D. Shaw. Seamlessly Selecting the Best Copy from Internet-Wide Replicated Web Servers. The 12th International Symposium on Distributed Computing (DISC'98) (formerly WDAG), pages 22-33, Lecture Notes in Computer Science 1499, September 1998.

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

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

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

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

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

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

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

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

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

[13] D. Shaw. Walrus: A Low Latency, High Throughput Web Service Using Internet-wide Replication. Masters Thesis, Department of Computer Science, The Johns Hopkins University, Baltimore, Maryland, August 1998.

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

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

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