Chapter 1

Web Servers: Configuration and Design

1 Introduction

1.1 What is a web server?

1.1.1 Basics

At the most basic level, web server serves content to a Web browser. It receives a request for a web page such as: http://www.cs.brown.edu/courses/cs227/syllabus.html Which is called a URL (Uniform Resource Locator) and maps it to a local file on the host server. The file syllabus.html is on the host file system. The server then loads this file from disk and serves it out across the network to the user's Web browser. The browser and server communicates using Hypertext Transfer Protocol (HTTP). This work-flow is shown in the figure below.

1.1.2 (Web service architecture diagram)

1.1.3 Web services (URL protocols)

serving files In this section, we are going to give a brief overview of some of the web services that are available to the market at present.

- FTP (File Transfer Protocol) is a way to upload and download files on the Internet. Typically a site on the Internet stores a number of files (they could be application executables, graphics, or audio clips, for example), and runs an FTP server application that waits for transfer requests. To download a file to your own system, you run an FTP client application that connects to the FTP server, and request a file from a particular directory or folder. Files can be uploaded to the FTP server, if appropriate access is granted. FTP differentiates between text files (usually Ascii), and binary files (such as images and application executables), so care must be taken in specifying the appropriate type of transfer. When an Internet site makes files available to the general public, this is called 'anonymous' FTP. A password does not need to be supplied, although the user e-mail address is typically requested. Some sites have confidential files or directories, and an FTP login and password is needed to download or upload.

- Gopher is an Internet application protocol in which hierarchically organized file structures are maintained on servers that they are part of an overall information structure. Gopher provided a way to bring text files from all over the world to a viewer on your computer.

streaming (gather some references on streaming, direct readers elsewhere)

- HTTP (Hyper Text Transport Protocol) is the underlying client/server protocol used by the web, which is a huge client server system. For a more detail description, please refer to Section 2, HTTP Server Basic.

- RTSP The Real-Time Streaming Protocol allows to control multimedia streams delivered, for example, via RTP. Control includes absolute positioning within the media stream, recording and possibly device control.

1.1.4 why concentrate on HTTP servers?

In this chapter, references on web servers are going to be referring to HTTP servers. HTTP servers are simple to understand and it is the predominant protocol on the web. Therefore, by understanding HTTP, one can have an excellent overview on web servers.

1.1.5 compare and contrast: file servers vs. HTTP servers

HTTP servers are very similar to file servers.

file servers are read/write

HTTP servers are read-only

1.2 Advantages of HTTP servers

HTTP servers provide a standard, cross-platform means of delivering information and multimedia content on the web.

1.2.1 standard, cross-platform, multimedia content delivery system

1.2.2 enables the web as a universal information system

1.2.3 everybody else is doing it

1.3 What makes running a web server hard?

1.3.1 maximize performance given available resources

1.3.2 tune for varying definitions of "performance"

1.3.3 end-to-end security concerns

1.4 Contents overview

In next section, we are going to start off by explain what is involved in a http server when a user sends in a request. We are also going to explain the most common three types of request-response chains. Then, the general software architecture of a web server will be discussed, examples of current web servers (apache, oracle, etc) will be given to illustrate the workings inside a web server. In the fourth section, we are going to explore what has been done to optimize a web server. Keeping in mind that each user will define their performance goals differently, in general, user wants the web page to display correctly and quickly and web architects would like to achieve this goal in the cheapest way. In this section, we will look at some of the common errors that occurs and what has been done to optimize performance.

In order to explain fully how a web server works internally, we will consider the simplest server, and the workings within a single machine. Having explained the basics of a single server, we will build our explanation for clustered servers and big iron upon it and present the advantages and disadvantages for each of them. we will first explain how a single server works, and then build clustered servers based on the knowledge from a single basic server. In the sixth section, we are going to explore securities issues within the web servers. As uses for the web sky rockets, security for information that is sent across the web is of extreme concern. In this section, we are going to present the different measures that have been taken to ensure security.

The reader should be well equipped with a general overview of how web servers work, and all the theories behind how a web server functions. In the seventh section, we will look at some of the real life application of the theories that we have already discussed. Through case studies of some of the major web servers on the market, we will look into real life design choices that companies have to make based on tradeoffs that we have already discussed in previous sections. Finally, we will talk about how this chapter connects to other chapters in the book and there will also be pointers to advanced topics at the end of the chapter.

2 HTTP Server Basics

HTTP is a HTTP is a "request-response" type protocol that follows the following path: [table here] TCP is a connection-based, end-to-end Internet network protocol, and HTTP is an application-level protocol that works on top of it.

There are three general request-response chains in which HTTP operates upon.

2.0.1 Basic Client to server http operation

The first is when a user agent makes a request directly to the origin server. In this scenario, the user agent makes a connection directly to the origin server and sends its request. The server will be listening for incoming connections and start a new thread or process to serve the new request. Once the request has been processed, the server sends the response back over the connection.

[diagram here]

2.0.2 Http operation involves proxy/cache agent

The second request-response chain involves a proxy or cache agent as an intermediary. In this scenario, the user agent makes its request to the proxy instead of to the origin server (See diagram below). The proxy then makes the request to the origin server on behalf of the client. The server replies to the proxy, and then the proxy relays this to the user agent, thus fulfilling the request. This type operation is mostly seen in firewall environments where the local LAN is isolated from the Internet. An alternate on this procedure is for the intermediate agent to also serve as a caching agent.

[diagram here]

2.0.3 Client to server via tunnel HTTP operation

When making a request through the cache agent, the cache agent tries to serve the response from its internal cache of resources. The cache itself saves any response it receives, if the response is a cachable one. This shortens the request-response chain, improves response time, and reduces network load. Most proxy agents are also caching agents. The third and final scenario is one involving an intermediate agent, acting as a tunnel. A tunnel blindly funnels requests and responses between two HTTP applications. As shown in the diagram below, it is providing a path for the user agent to the server.

[diagram here]

A tunnel is different from a proxy in how it operates. A tunnel is simply a mechanism via which the user agent sends requests and receives responses from an origin server. The tunnel itself does nothing to the requests, unlike a proxy which may rewrite certain headers or require authentication from the user before providing services. A tunnel would be used most often to route HTTP traffic over a non-TCP/IP link. Past the above three basic request-response chains, anyone can put together any combination of intermediate agents. It is entirely reasonable for a user agent to send a request to a proxy, which sends it through a tunnel which reaches another proxy, and finally makes it to the origin server. Through all of this, the basic idea still maintains the request-response paradigm, although it may make many contortions along the way.

3 Web Server Software Architecture

3.1 Apache as a running example

3.2 Oracle as a web server?

3.3 Other server software should be mentioned

3.4 Web server features

3.4.1 server-based caching

3.4.2 algorithms

URL lookup

scheduling/prioritizing requests

thread pools

3.4.3 reimplementing OS and filesystem

disk partitioning

object location on disk

3.5 Integrated dynamic generation (mainly reference dyncon chapter)

4 Configuring for Performance

4.1 Performance goals

With the exponential growth of the Internet, measures need to be taken to ensure that web viewers, even as their number increase, continue to have a pleasant experience. A web site must load quickly and correctly. Studies have shown that a page has approximately seven seconds to load enough content to interest the user before she becomes impatient and clicks her mouse on the back button. Even more annoying to users and, more importantly, embarrassing to the content provider is when the site doesn't load at all. Several measures such as writing smaller web pages and using proxy caches can be taken to achieve this goal outside the web server, but in this chapter, we will focus on the making those optimizations at the start, within the web server. A web server that will fare well with a growing user base must maintain several key performance levels.

As each web surfer views a page on the web, he makes a request to the web server for that page. Similar to retrieving a coat from a dry cleaner, the client sends out a ticket, asking for a specific piece of data, and that data is returned. Sometimes, each page may require several requests, because the page is composed of several parts, graphics, for example. As the number of simultaneous viewers increases, and as web pages become more complex with many embedded pictures, sounds, and animations, the number of requests a web server has to fulfill simultaneously becomes very high. Thus our first performance goal is to maximize the number of requests per second a server can handle.

Likewise, as the number of simultaneous page views increases and web pages become more complex with intricate layouts and special effects, the amount of data that needs to be pushed out to the web surfer every second increases as well. Therefore a good, scalable web server will maximize throughput.

Another characteristic of a web server that will affect its performance is latency. Latency is the amount of time between the client's sending out the request and receiving the data. This is also known as roundtrip time. While usually latency can be attributed to bottlenecks outside the web server – for example, a slow internet connection – as the number of requests per second increases, the amount of time it takes to fulfill each request becomes important. Therefore a good web server will minimize the time needed to read a file off disk or generate one in real-time to send it to the client.

And, of course, errors happen. When a server is under heavy load, some requests may just have to be dropped – the request may actually never reach the server, or the server may be simply so busy that it can't keep up.

And obviously web servers do go down, whether it be because of power failure, a hard drive crashing, or buggy software. A good web server will try to minimize the likelihood of crashing and, in that unfortunate situation, minimize the amount of time needed to come back up.

Lastly, and most importantly, to a certain extent we could simply buy the fastest server if money were not an issue. However, cost is a big factor in most budgets, so while not a performance factor, a good web server will yield the best performance with a given financial investment.

4.2 Limited resources

Limited resources derive from cost constraints. The speed of the internet connection, the speed and size of disk and memory, and the speed of the processor are all determined by how much money is spent on the web server infrastructure. But to better understand the various techniques of optimizing web server performance given these limited resources, we need to explore what these resources are and how they are significant.

The first, and probably most significant, limited resource is network bandwidth. In most web server configurations, this is the first constraint that will affect throughput. Typically, when one refers to network bandwidth, he means the amount of data the Internet connection can transport to and from the server in a given time. However, a network often overlooked is the communication between servers. For example, every time you check your credit card bill online, the web server probably has to talk to a mainframe over a local network to retrieve that information before the server can send you the page.

Another financially constricted resource is disk storage, which is limited in two ways: bandwidth and latency. Disk bandwidth, which is similar to network bandwidth, is usually increased using certain configurations of RAID (Redundant Array of Inexpensive Disks). RAID striping increases bandwidth by having multiple disks read and write different pieces of data, so for example (see figure) in a two disk RAID setup, the first disk would handle all odd numbered bytes and the second disk would handle all even numbered bytes. However, striping also increases the chance of disk failure, since if one drive dies, the data on the other drive is incomplete and thus useless. RAID mirroring simply writes the same data on multiple drives. While not increasing bandwidth, mirroring ensures that if there is a disk failure, there will be a backup. Modern RAID techniques us a complex combination of striping and mirroring to increase disk bandwidth while maintaining reliability, but as a result the price path to high bandwidth is not a linear one. Disk latency, the time needed for a disk request to be filled, on the other hand, is for the most part limited on technology. Some techniques such as caching, have lessened disk latency on the average, but in general it is a fixed factor that has not decreased much in recent history. (footnote: in the last 10 years, disk latency has decreased by less than 50 percent.)

Sometimes a panacea for having a slow disk is to access the data from memory instead. Random Access Memory (RAM) is extremely fast, but extremely expensive, so the amount of data you can store in RAM, whether it be parts of the web page precluding the need to access a slow disk ever time or information about each client, is limited even more severely.

And of course, at the heart of every web server is its processor. Typically the number of instructions that a server can perform each second, usually measured in MIPS (Millions of Instructions per Second) is not a great factor in its performance, but now that web sites are becoming dynamic, requiring the server to process complicated data before it is sent as a web page processor speed is starting to be important. IPS can be increased by using a faster processor, which is limited by the current technology available, or by using more than one processor under SMP (Symmetric Multi-Processing). But as modern RAID doesn't increase disk bandwidth linearly with the number of disks, SMP doesn't increase IPS linearly with the number of processors, and some platforms handle it better than others.

Lastly, there are some more obscure resource limits that will affect web server performance. Tasks are done simultaneously with threads, such that each task is given its own thread, so a client requesting a page from the server will be given a thread. However, the number of threads the operating system can handle is fixed (and the web server is often configured to max out at a number of threads much lower than this). Additionally, TCP limits the number of simultaneous connections to XXXX. So no matter how fast or big your web server may be, eventually it will no longer be able to handle more simultaneous requests. At this point, queuing occurs, where requests are put off to be processed later. Queues, however, are also limited in size to the amount of memory available to store the queue.

4.3 Benchmarking and load testing

4.3.1 GStone

4.3.2 SPECweb99

4.3.3 TPC-W

4.3.4 WebBench

4.3.5 WebStone

4.3.6 Mercury Interactive

4.4 Post-mortem analysis

4.4.1 log analysis

4.4.2 trace analysis

4.5 Choosing a configuration

4.5.1 performance studies

4.5.2 experimental analysis

4.5.3 symptoms of overload

5 Tradeoffs Between a Single Server, Clustered Servers, and Big Iron

5.1 Introduction

The ideal web server would have unlimited resources – an infinitely fast network, boundless and instantaneous disks and memory, and a processor that could whisk through an infinite loop in six seconds. However, in the real world such a machine doesn't exist. Even with infinite financial resources, technology and other external bounds force us to seek other alternatives.

And yet still, there is no ideal solution for maximizing performance with limited resources. Rather there are many good solutions, but each solution depends on the different needs of the web site.

The basic web server is just one machine – it has one processor, a sizeable but limited amount of memory, one or more individual disks, and a single network card. Simple, efficient, and best of all inexpensive, this server might be ideal, if your audience never exceeded 100 at a time, or fewer if you needed to make dynamic content. So seeing the limelight on the horizon, two paths exist for improving the web server's capabilities: expanding outward and expanding inward.

5.2 Price/performance tradeoffs

5.2.1 clustering

Similar to the ideology of RAID and SMP, clustering utilizes many inexpensive, basic (or nearly so) web servers so that they appear to the outside world as one entity. Like members within a team, they all do their part towards a common goal, but there are also several methods in which the team of servers is organized.

The simplest form of clustering is mirroring, in which several identical servers perform the same functions. A special network router distributes client requests evenly to the servers. This might actually be an ideal solution, since one could simply add capacity by adding another server. However, the solution only works well in a situation where the web pages are static. Every time content changed, the change would need to propagate through every mirrored server. If change happens frequently, which is becoming much more prevalent in dynamic websites, the overhead of maintaining synchronicity may undo the gains for mirroring. Additionally, if the amount of content is large, and most of it is accessed infrequently (online bank statements, for example), it may be impractical and wasteful to keep a separate copy on every mirrored server.

Another method of clustering is data partitioning. If the website can be separated into self-contained pieces, each piece can reside on an inexpensive server. This solution is ideal as long as user interaction would never affect the state of another server. For example, a large website might place message boards on a different server or one server might handle bank records for one letter of the alphabet. However, this method requires that estimates be made about the usage distribution before the servers are set up. It may be the case that estimates were wrong, and creating a separate message board server did very little to alleviate load from the main server.

Finally, task factoring is like a modern office environment where there is an intricate web of cooperation between workers and managers. There may be some mirrored servers to handle heavily used, static data, and some data partitioned servers to supply customized interfaces, a server to handle login authentication and direct the user to his customized interface, and a large database server. In most intricate web server situations, a customized cluster with task factoring will be implemented to best suit that situation.

Clustering is often the implementation of choice because it is an inexpensive solution, which is oftentimes scalable after implementation. Similar to RAID, clustering is more cost effective than simply investing in a faster server, and on a large scale it is the only solution as the current technology limits how much power can be placed in one server. But again like RAID, a task factoring implementation increases the probability of failure, since if one critical server goes down, the entire site may cease to function, and a mirroring solution neutralizes that weakness by providing support fail over. Since mirrored nodes are all identical, if one goes down, the others can pick up its share. But there are severe disadvantages that may make clustering not suit some situations. A highly interactive web community, for example, would be difficult to implement using a cluster because clusters have difficulty maintaining a common state between all machines. In a web community, information is constantly being updated which permeates all parts of the web site. Therefore each clustered server would need to obtain that information all the time. This poses problems in synchronicity and bandwidth. Even if a dedicated database server is used, won't it too eventually become overwhelmed? What if two clustered servers need to read and write the same data at the same time? If so, how do they communicate this?

But for clusters to operate efficiently, each server node needs to do an equal share of the work through a process of load balancing. Under a mirroring scenario, software-monitoring tools can be used to more efficiently distribute load – if one machine seams to be overly busy, more requests will be shifted towards the less busy servers. Under a data and task partitioned scheme, however, there is no such simple solution. Before any server's duties are finalized and implemented, their load must be estimated to determine if dedicating a machine to that data or task is worthwhile. After the website is up and running, it may be found that the estimates were wrong or as time progresses needs change, and load is not balanced. If redundancy is used, software tools might be able to reconfigure a node so that it then performs a task or handles a segment of data that was before overwhelmed. But while this may be a software solution to the problem, intricate and optimized server setups will have dedicated network lines between appropriate key servers, and reconfiguring a physical network is much more difficult than reconfiguring software.

standards needed here

5.2.2 big iron

The other method for expanding a web server is to invest in a more capable single server with more disk bandwidth, more memory, and more processing power. Disk bandwidth, which is similar to network bandwidth, is usually increased using certain configurations of RAID (Redundant Array of Inexpensive Disks). RAID striping increases bandwidth by having multiple disks read and write different pieces of data, so for example (see figure) in a two disk RAID setup, the first disk would handle all odd numbered bytes and the second disk would handle all even numbered bytes. However, striping also increases the chance of disk failure, since if one drive dies, the data on the other drive is incomplete and thus useless. RAID mirroring simply writes the same data on multiple drives. While not increasing bandwidth, mirroring ensures that if there is a disk failure, there will be a backup. Modern RAID techniques us a complex combination of striping and mirroring to increase disk bandwidth while maintaining reliability, but as a result the price path to high bandwidth is not a linear one.

Sometimes a panacea for having a slow disk is to access the data from memory instead. Random Access Memory (RAM) is extremely fast so having more memory would allow more pages to be to be fetched quickly from RAM instead of slowly from the disk drive. Also, since data is stored about each connected client, more memory allows for more connected clients.

And now that web sites are becoming more dynamic, requiring the server to process complicated data before it is sent as a web page, processor speed is starting to be important. Currently available technology limits the speed a single processor can be, so like RAID, multiple processors can be used in the same machine to increase the number of instructions that can be processed per second.

The advantage of big iron over clustering is that load balancing is usually not much of an issue. Load still needs to be balanced between processors, but this is usually something well handled by the operating system. Also, since there is only one machine accessing one disk, big iron systems are well suited for highly dynamic websites in which constantly changing data permeates every page.

Unfortunately, however, big iron machines are not at all as cost effective as clustering. (figure: cost curve of price vs. performance.) Also, after the machine is up and running, it is virtually unscalable. Additional processors typically cannot be added. Disk space can be added, but not bandwidth. And memory can be added, but that would require bringing the system down for the upgrade. For a always-on server, the future must be planned for, so an overkill system is initially invested in, and the extra power goes unused as the workload is anticipated. Lastly, because there is only one machine, it is more likely than a mirroring cluster to fail. However, modern big-iron machines have built-in internal redundancies such that if a processor, disk, or power supply fails, the others will pick up the slack.

6 Security

6.1 Why bother?

Web server security is extremely important. As more and more users uses the web to transmit information, we must ensure that sensitive information is handled in the appropriate way. For example, most people would not be too happy if their credit card information falls in the hands of an undesirable third part. Also, other people should not be able to tap into a network.

There are two different levels of security for a Web server. On one level is the security of the data stream itself so that it may not be viewed or modified by a malicious third party. On another level is the security of the content itself – the authentication and authorization of people to view and change that content. URLs that begin with "https" are handled using SSL (now referred to as Transport Level Security – TLS) algorithms. These algorithms basically work by setting up a secure, encrypted link between a Web browser and a Web server. SSL protects either: the data being posted to the Web server or the retrieval of some confidential data from the Web server. An example of a user posting confidential data to a Web server can be found in a typical Web store application. In such an application, the user is usually given a choice of presenting his or her credit card information to the order form. Although the Web server may not echo the user's credit card information again to the Web browser, the actual transmission of this information must to be treated as confidential. There is also the issue of protecting content on the Web server that is being served. For example, an auction site may want to protect the bids a user is receiving for an item so that only the individual who posted the item sees all the bids. In this case, it is not enough to simply encrypt the traffic being sent. The Web server must also be able to identify the user and the data she has access to. These two processes are referred to as authentication and authorization, respectively. Web servers usually support authentication using a technique called basic authorization. In this technique, a Web server sends a special header to the user's Web browser asking for a username/password combination. This results in the Web browser popping up a log-in window. Web servers are usually less sophisticated with regard to authorizing the viewing of data. Most Web servers merely allow the restriction of the reading of directories and files within a directory by group or user. More sophisticated options for determining whether a user is authorized to view files (such as time of day) must usually be programmed into a Web application. Below, methods of web security will be described in more detail.

subsectionAuthorization and encryption

In initial editions of web servers, authorization was implemented using an simple text file in the server directory called .htaccess, with the username and password required to access that directory. The web server would look for this file, and if present, prompt the web surfer for her username and password. If the surfer enters something valid in the list, the data that she requested is delivered. (who remembers the login information, client or server?) As websites became more complex and user bases grew, the simple list based authentication process was replaced with databases and integration to the operating system security (see. Windows active directory.)

To prevent eaves dropping, data transported between the server and client became encrypted using SSL (Secure Socket Layer), and HTTPS became the standard for sending web objects securely using SSL. SSL is a form of public key encryption, in which each host generates two large random numbers, p and q, whose product is called n. then two number j and k are chosen such that they are relatively prime to p and q. j is then called the public key and k is called the private key. j is sent over the internet such that the other host can use it to encrypt data by raising it to the j power. This data is then sent back to the original host, which then raises it to the k power. The result is the original data modulo n.

6.2 Firewalling sensitive data

Oftentimes a web server will be located on the premises of the company that wants to host its content, which is based off of internal sensitive data. A bank, for example, may have an existing network with a mainframe, and they want users to access that data through a web server, which will exist on their network.

Obviously they would want users to be able to access the web server through the Internet, but not the mainframe. Additionally, the web server must be able to access the mainframe. Intuitively, transitivity would prevent all these conditions from being met. A firewall solves the security issue sitting between the web server and the mainframe to make sure that while the web server can communicate to the mainframe, no one else can. The firewall could also sit between the web server and the Internet, analyzing TCP packets to make sure that only requests are sent to the web server. Obviously this is only one application of firewalls, and they become more complicated, allowing others from within the firewall to access the internet, etc.

6.3 Virtual private networks (VPNs) and extranets

Sometimes, however, the web server is not located on the same network as the content. For example, a smaller bank might have a mainframe on their local network, but they might want to outsource the web hosting responsibility to someone else. In this situation, the two networks are only connected via the Internet. Again, the web server needs to access sensitive data on the mainframe to display to the web user. However, transmitting that data over the Internet permits the possibility of eaves dropping. VPNs (a.k.a. tunnels) and extranets provide a secure solution for this communication and transparently pretends that the web server's local network is the same as the bank's local network. VPNs typically require an additional machine on each end – the web server and the local network. Each tunnel server is actually an Internet router. When one of the routers encounters a TCP packet destined for a machine on the other network, it encrypts the entire packet, wraps it in a new Internet packet destined for the other router over the public Internet. Once the packet is received by the tunnel server/router, it is decrypted and sent on its way to the destination. This way no one needs to specifically configure anything but the tunnel servers to operate over the Internet. The web server and mainframe just believe that they are on the same private network.

7 Case Studies (primarily concerned with configuration choices)

7.1 E-bay

Online auctioneer eBay Inc. was launched in 1995 and has become the world's leading online trading community and most popular shopping site. Each day eBay users add nearly 600,000 items for sale – everything from automobiles to video games. At any given time the company has over 3 million active simultaneous auctions, accepting thousands of bids every minute. With that volume of merchandise for sale, tracking the data and bids puts a high demand on systems and storage.

But not only do eBay's systems need to be fast, they also need to be reliable. Since it's inception, the web site has experienced 52 hours of total failure, the longest downtime being 22 hours, which made the front page of the New York Times and caused its stock price to drop 25%. Site failures are embarrassing and turn away customers. After the June 1999 crash, many irate customers complained of losing thousands of dollars and even turned to lawsuits. The less irate simply migrated to rival auction sites Amazon and Yahoo. The crash was estimated to cost the company over $5 million in lost revenues.

EBay web server infrastructure uses a combination of all web server technologies – big iron, mirrored clustering, data partitioned clustering, and task divided clustering. The main, big-iron machine is a Sun Microsystems Starfire E10000 server, which is used for transactions against an Oracle database. This machine stores all auction data – product description, price, number of bids, duration, etc. Virtually every web server for the eBay website accesses this machine to produce auction web pages. The Starfire utilizes 64 symmetric multiprocessors, which are targeted to remain 50% idle to provide a buffer against heavier use. The server's storage consists of over 400 disk drives. With so many drives, hardware failures happen quite frequently, thus RAID level 5 is used to provide a combination of striping for speed and mirroring for reliability. 50-100 gigabytes of memory are used for caching and general database operation use.

A second Starfire and seven four-processor Sun Enterprise ES450s query the entire database periodically to create a local search index. These eight servers are task and data partitioned such that they locally host the search index and perform only searches, of which there are many kinds – one for each server. After a search is performed, the result will transfer the user to the appropriate page on the appropriate server. Serving as the web front end – what a surfers web browser will directly communicate with – are about 60 mirror clustered Compaq PC servers, running Windows NT and a Microsoft Internet Information server. Each machine serves the dynamic auction content using ISAPI, compiled C++ based web scripting which interfaces with the main Starfire database to generate auction pages.

EBay chose the web server infrastructure they did, because it best fit their needs. The biggest bottleneck, however is the giant Starfire database. This is a big iron machine that unfortunately can only improve by becoming bigger. However, it already uses the maximum number of processors that the machine allows, so eventually other alternatives will need to be discovered. For example, Storage Area Networking (SAN) may be a solution in which multiple machines can actually share the same disk array.

7.2 HotMail

Hotmail is the world largest web-based email provider. It's infrastructure is quite simple compared to eBays and much more easily scalable. Similar to eBays infrastructure, HotMail's front end is hosted by a mirror cluster of inexpensive web servers. These web servers interface to a small database that keeps track of user information. Behind these is a farm of thousands of data partitioned basic web servers. Each user of Hotmail has his mailbox on one of these servers. When he logs into Hotmail, the database will direct him to the appropriate server based on his login information. Once there, the mailbox server acts autonomously for reading, composing, and sending mail. To receive mail, the database server is queried to identify which mailbox server to forward each piece of mail. Unlike eBay's gargantuan database server, Hotmail's is relatively simple, containing only a map from usernames to mailbox servers. Thus the database server, which essentially only performs simple searches, is easily mirror clustered as the need arises. The mailbox servers do not need to maintain a common state, so there is no communication between them. The mirrored database servers, however, need to keep contact in case user information is changed. Aside from this complication, Hotmail can easily be expanded by adding more inexpensive servers.

8 Related Topics

8.1 Performance optimization is application-specific

As we have seen with the case studies, different web server applications will warrant different means for optimization. Dynamic content generation and customization are more processor intensive and utilize a connection to a database. Search engines are more internal bandwidth hungry and can only usually be optimized through big iron.

8.2 Interaction with *proxy caches*

While optimization on the server end does allow more users to view the website simultaneously, eventually resources such as limited internet bandwidth forms will preclude anymore useful optimization on the server infrastructure end. Proxy caching is a solution where web data is remembered one a cache server closer to the user, such that many users can access this server instead of accessing the main web server. A prime example is during the Election 2000 fiasco which crashed the web servers of many news companies, but CNN remained unscathed because they utilized Akamai's proxy cache. But with the use of proxy caching, several dilemmas occur: how does the web server ensure that the proxy cache contains up-to-date data, when proxy caches are many and anonymous? Likewise, if proxy caches serve pages in lieu of the web server, how will page view statistics be affected?

9 Summary/Conclusion

9.1 Web services are important

The web despite its so-far exponential growth rate, is only beginning to blossom. In just a few year, half of America is on the internet, and the average American signs online XX times per week. Almost anything you want to do can be done on the web from paying your credit card bill to searching for a house to making new friends. Let's face it, the web is here and it's growing.

9.2 Hardware and software

True the biggest, fastest web server could be used, but cost is a factor that encourages engineers to optimize website performance with limited resources. And different web sites need a different regimen of hardware and software to be efficient. A large database web server probably needs to go the route of big, fast, and expensive, since single server is more efficient than maintaining state across several machines. Additionally, the software of choice will probably be of a UNIX flavor, as the Microsoft NT alternative does not function efficiently on large, multiprocessor machines, because of thread pool limitations. Likewise, a setup not requiring consistent state – that which can be easily task and data partitioned or simply mirrored – would probably be best served by a cluster, since the performance gain for many servers is relatively inexpensive. For clustering, NT is a much more viable alternative.

9.3 Security

As the web becomes more popular, more sensitive information is being delivered to users. But this it is imperative that this data not fall into the wrong hands. HTTPS is a standard for which web data can be sent securely encrypted to and from the user. Firewalls provide a solution to block off unauthorized access to corporate mainframes. And Virtual Private Networking provides a means to build extranets, where off-site servers can communicate with private mainframes, pretending to be on the same private network.