M. Engg. Project

Design and Implementation of System Architecture in a Huge and Secured Online Transactional System

by

Mohammad Delwar Hossain

বাংলাদেশ প্রকৌশল বিশ্ববিদ্যালয়
নং 105878
তাং 25/10/08
লাইব্রেরী, ঢাকা

Submitted to

Institute of Information and Communication Technology

In partial fulfillment of the requirements for the degree of

Master of Engineering in Information and Communication Technology

#105878#

Institute of Information and Communication Technology (IICT)

Bangladesh University of Engineering and Technology (BUET)

Dhaka 1000

May 2008

The Project entitled "**Design and Implementation of System Architecture in a Huge and Secured Online Transactional System**" submitted by Mohammad Delwar Hossain, Roll MF0231020, Session 2002-2003 has been accepted as satisfactory in partial fulfillments of the requirement for the degree of Master of Engineering (ICT) held on 31 May 2008.

Board of Examiners

1.	Dr. Md. Abul Kashem Mia Professor Institute of Information and Communication Technology BUET, Dhaka – 1000	Chairman (Supervisor)
2.	Dr. S. M. Lutful Kabir Professor & Director Institute of Information and Communication Technology BUET, Dhaka – 1000	Member
3.	Dr. Md. Liakot Ali Assistant Professor Institute of Information and Communication Technology BUET, Dhaka – 1000	Member

Candidate's Declaration

It is hereby declared that this project or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Delwar

Mohammad Delwar Hossain

Dedicated

to

my parents

Contents

Acknowledgements vii

Abstract 1

Chapter 1 2

Introduction 2

1.1. The Rise of Internet Service 2

1.2. Internet Service Properties 5

1.2.1. High Concurrency 5

1.2.2. Dynamic Content 6

1.2.3. Robustness to Load 7

1.3. Trends in the Internet Service Design Space 9

1.3.1. Rapid Evolution of Service Logic 9

1.3.2. General Purpose Hosting Platforms 10

1.4. Challenges in Designing Robust Internet Services 10

1.5. Project Objectives 11

1.6. Organization of the Project Paper 12

Chapter 2 13

Preliminaries 13

2.1. Web Serving Overview 13

2.2. Cluster Architecture 15

2.2.1. Layer 4 Switching with Layer 2 Forwarding 16

2.2.2. Layer 4 Switching with Layer 3 Forwarding 17

2.2.3. Layer 7 Switching 17

2.3. Server Selection / Request Distribution 17

2.3.1 DNS Round Robin 17

2.3.2 Weighted Round Robin (WRR) 18

2.3.3 Weighted Least Connections (WLC) 20

2.3.4 Locality-Aware Request Distribution (LARD) 20

2.3.5 Workload-Aware Request Distribution (WARD) 21

2.3.6 Parallel Pull-Based LRU (PPBL) 22

2.4. Advantages of Caching Dynamic Content 24

2.5. Communication Security Fundamentals 25

2.5.1. Security Objectives 25

2.5.2 Types of Attacks 26

Chapter 3 28

Design and Implementation 28

3.1. Cluster Architecture 30

3.2. Caching Dynamic Content 33

3.2.1. Document Object Model (DOM) Level 1 33

3.2.2. Solution in BIIS 35

3.3 Multi-tier Architecture 35

3.3.1 Three-tier Architecture 35

3.3.2 Solution in BIIS 37

3.4 Implementation of SSL 40

Chapter 4 41

Conclusion 41

4.1 Future Works 42

References 43

Acknowledgements

I would like to express my deep and sincere gratitude to my supervisor Dr. Md. Abul Kashem Mia, Professor, Institute of Information and Communication Technology (IICT), Bangladesh University of Engineering and Technology (BUET), Dhaka. I owe to him for his supervision, encouragement, personal guidance during the progress of my project. His in-depth knowledge in Software Engineering and his logical way of thinking have been very helpful for the successful completion of this work. I am deeply grateful to him for his co-operation.

I would like to thank the members of my project committee for their patience in understanding my work. I warmly thank Professor Dr. S. M. Lutful Kabir and Dr. Md. Liakot Ali for their valuable suggestions.

A number of people contributed their technical and analytical expertise to this project; they include Taslim Arif, Asif Adnan, Sumaya Jahangir, Anupam Bhattacharjee and others. However any error or omission associated with this project are strictly to the author.

I am thankful to all of my teachers, colleagues, and friends for their support during the completion of my project. Finally I owe my loving thanks to my parents, brothers, and sisters. Without their encouragement it would have been impossible for me to finish this work.

Above all, I am grateful to Almighty Allah who gave me the strength to finish this work.

Abstract

Today's Web applications need a platform that is fast, maintainable, stable and secure. This project work presents and implements an online transactional system architecture for handling massive and load conditioning demands, and concurrency of busy Internet secure services.

The bottleneck of Web content delivery is not serving static content but dynamic content which requires data processing. For such web applications scalability can be provided by a third party solution that caches application data and can maintain high concurrency when the concurrent user quantities are unpredictable. We propose to use a database cache for offloading the demand on a database-driven Web service. This project work presents the design of a database cache. In this project work we also study the security-scalability tradeoff, both formally and experimentally. We begin by providing a method for statically identifying segments of the database that can be encrypted without impacting scalability.

The system is designed and tested based on the present course registration system and grade submission system of BUET as a model university.

Chapter 1

Introduction

বাংলাদেশ প্রকৌশল বিশ্ববিদ্যালয়
নং 105978
তাং 25/10/08
লাইব্রেরী, ঢাকা

The bottleneck of Web content delivery is not serving static content but dynamic content which requires data processing [1]. The workload seen by Internet applications varies from time to time and often it varies in an unpredictable fashion [2, 3].

The web has evolved from being an archive of distributed, static information to an interactive, wide-area application service provider. This evolution requires new techniques for improving web performance, as measured by client-perceived latency. One approach to maintaining low latency is to scale these applications.

Web-based services that generate dynamic content do not receive the performance benefits of traditional static content caching.

This project presents architecture for handling the massive concurrency and load conditioning demands of busy Internet services. Existing programming models and operating system structures do not adequately meet the needs of complex, dynamic Internet servers, which must support extreme concurrency (on the order of tens of thousands of client connections) and experience load spikes that are orders of magnitude greater than the average. In this work, we show that the our design yields higher performance than traditional service designs, while exhibiting robustness to huge variations in load.

1.1. The Rise of Internet Service

The exponential growth of the Internet in the last few years has given rise to a vast range of new services being deployed on a global scale. No longer dominated by Web sites with static content, Internet services span a diverse range of categories

including stock trading, live media broadcasts, online shopping, instant messaging, peer-to-peer file sharing, and application hosting. In contrast to static content sites, this new class of dynamic services requires significant computational and I/O resources to process each request [4]. An increasingly complex array of systems are involved in delivering these services, including Web servers, caches, middle-tier application servers, databases, and legacy systems. At the same time, exponential growth of the Internet population is placing unprecedented demands upon the scalability and robustness of these services. Yahoo! receives over 1.2 billion page views daily [5], and AOL's Web caches service over 10 billion hits a day [6]. Internet services have become critical both for driving large businesses as well as for personal productivity. Global enterprises are increasingly dependent upon Internet-based applications for e-commerce, supply chain management, human resources, and financial accounting. Many individuals consider e-mail and Web access to be indispensable lifelines. This growing dependence upon Internet services underscores the importance of their availability, scalability, and ability to handle large loads. An extensive outage at E*Trade resulted in a class-action lawsuit against the online stock brokerage by angry customers [7]. Likewise, during a week-long outage of MSN Messenger [8], many users expressed a great amount of desperation and hostility from being disconnected from the instant-messaging service. As more people begin to rely upon the Internet for managing financial accounts, paying bills, and potentially even voting in elections, it is increasingly important that these services perform well and are robust to changes in load.

This challenge is exacerbated by the bushiness of load experienced by Internet services. Popular services are subjected to huge variations in demand, with bursts coinciding with the times that the service has the most value. The well-documented "Slashdot Effect" shows that it is not uncommon to experience more than 100-fold increases in demand when a site becomes popular [9]. The events of September 11, 2001 provided a poignant reminder of the inability of Internet services to scale:

virtually every Internet news site was completely unavailable for several hours due to unprecedented demand following the terrorist attacks on New York and Washington. CNN.com experienced a two-and-a-half hour outage with load exceeding 20 times the expected peak [10]. Although the site team managed to grow the server farm by a factor of 5 by borrowing machines from other sites, even this was not sufficient to deliver adequate service during the load spike. CNN.com came back online only after replacing the front page with a text-only summary in order to reduce load [11]. Apart from these so-called flash crowds, sites are also subject to denial-of-service attacks that can knock a service out of commission. Such attacks are increasingly sophisticated, often being launched simultaneously from thousands of sites across the Internet. Denial-of-service attacks have had a major impact on the performance of sites such as Buy.com, Yahoo!, and whitehouse.gov [12], and several companies have been formed to help combat this problem [13]. The number of concurrent sessions and hits per day to Internet sites translates into a large number of I/O and network requests, placing enormous demands on underlying resources. Unfortunately, traditional operating system designs and common models of concurrency do not provide graceful management of load. Commodity operating systems focus on providing maximal transparency by giving each process the abstraction of a virtual machine with its own CPU, memory, disk, and network. Processes and threads are traditionally used for concurrency, but these mechanisms entail high overheads in terms of memory footprint and context switch time. Although simplifying the programming model, transparent resource virtualization prevents applications from making informed decisions, which are vital to managing excessive load.

1.2. Internet Service Properties

BIIS (BUET Institutional Information System) is a web based application for registration, marks entry and many others functionality, which means it is total educational information system.

The Internet presents a system problem of unprecedented scale: that of supporting thousands of users demanding access to services that must be responsive, robust, and always available. This work is motivated by three fundamental properties of Internet services: (1) massive concurrency demands, (2) an increasing trend towards complex and dynamic content, and (3) a need to be extremely robust to load. In this section we detail each of these properties in turn.

1.2.1. High Concurrency

The growth in popularity and functionality of Internet services has been astounding. While the Web itself is growing in size, with recent estimates anywhere between 4 billion and 5 billion unique documents, the number of users on the Web is also growing at a staggering rate. In 2004, Nielsen//NetRatings estimates that there are over 750 million Internet users worldwide, and a recent study found that there are over 280.5 million Internet users in the United States alone.

As a result, Internet applications must support unprecedented concurrency demands, and these demands will only increase over time. According to Web traffic analysis companies (including Compete.com, comScore, Alexa Internet, and Nielsen Ratings), the domain yahoo.com attracted at least 1.575 billion visitors annually by 2008 according to a Compete.com study. The global network of Yahoo! websites receives 3.4 billion page views per day on average as of October 2007. It is the second most visited website in the U.S.

1.2.2. Dynamic Content

The early days of the Web were dominated by the delivery of static content, mainly in the form of HTML pages and images. More recently, dynamic, on-the-fly content generation, which requires significant amounts of computation and I/O to generate, has become more widespread. A typical dynamic Internet service involves a range of systems, including Web servers, middle tier application servers, and back-end databases, to process each request. The processing for each request might include encryption and decryption (e.g., if the SSL protocol is being used); server side scripting using frameworks such as Java 2 Enterprise Edition (J2EE) [14] or PHP [15]; database access; or access to legacy systems such as mainframes (still commonly used for airline ticketing). In contrast to static Web pages, dynamic services require greater resources, and the resource demands for a given service are difficult to predict. Moreover, the content generated by a dynamic Internet service is often not amenable to caching, so each request demands a large amount of server resources.

The canonical example of a highly dynamic Internet service is a large "mega-site" such as Google [16], which provides a wide range of services, including a search engine, real-time chat, stock quotes, driving directions, and access to online retailers. In addition to consumer-oriented sites, specialized business-to-business applications, ranging from payroll and accounting to site hosting, are becoming prevalent. The trend towards dynamic content is also reflected in industry standard benchmarks for measuring Web server performance includes a significant dynamic content-generation component as part of the standard workload.

Another aspect of the increased demand on Internet services is that the size of Web content is growing, with streaming media, MP3 audio files, videos, and large software downloads becoming increasingly common. The latest VeriSign Domain Name Industry Brief reports that in the first quarter of 2007 worldwide, there were a total of 10.7 million new registrations. Apart from placing new demands on

networks, this growth in content requires that Internet services be responsible for dedicating large numbers of resources for storing and serving vast amounts of data.

1.2.3. Robustness to Load

Demand for Internet services can be extremely bursty, with the peak load being orders of magnitude greater than the average load. Given the vast user population on the Internet, virtually any site can be suddenly inundated with a surge of requests that far exceed its ability to deliver service. The media has reported on numerous examples of popular Internet sites being brought down by excessive load. We have already discussed the effects of the September 11 attacks on news sites. In September 1998, the U.S. Government released Ken Starr's report on President Clinton on the Internet, causing seven government sites to be swamped with requests from over 20 million users [17]. During the February 1999 Internet broadcast of a Victoria's Secret Fashion show, over 1.5 million users logged in simultaneously to view the broadcast, causing the site to melt down under the overload. According to Keynote systems, a Web site monitoring firm, only 2 percent of requests to the site succeeded during the live show [18].

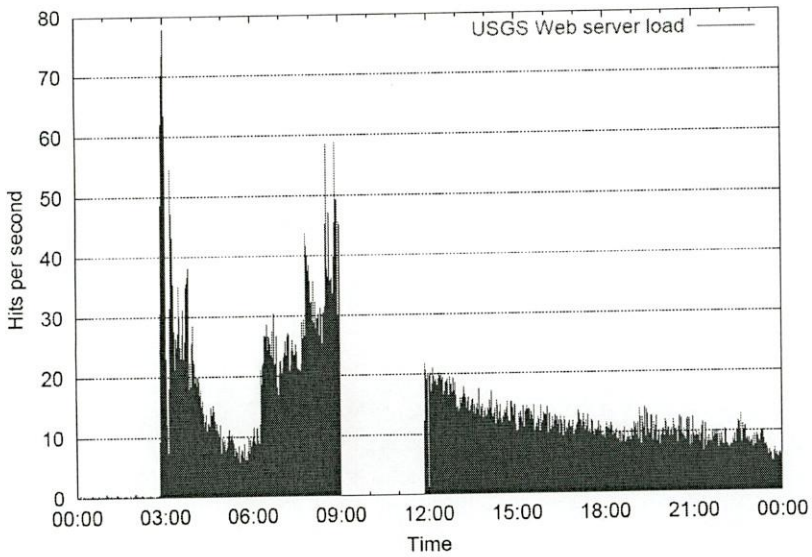

Figure 1: The effect of sudden load on a Web server [9].

Figure 1 shows a graph of the Web server logs from the USGS Pasadena Field Office Web site after an earthquake registering 7.1 on the Richter scale hit Southern

California on October 16, 1999. The load on the site increased almost 3 orders of magnitude over a period of just 10 minutes. Before the earthquake, the site was receiving about 5 hits per minute on average. The gap between 9am and 12pm is a result of the server's log disk filling up. The initial burst at 3am occurred just after the earthquake; the second burst at 9am when people in the area began to wake up the next morning. (Web server log data courtesy of Stan Schwarz, USGS)

As a more concrete example, Figure 1 shows the load on the U.S. Geological Survey Pasadena Field Office Web site after a large earthquake hit Southern California in October 1999. The load on the site increased almost 3 orders of magnitude over a period of just 10 minutes, causing the Web server's network link to saturate and its disk log to fill up [9]. Note that the figure 1 shows only the number of requests that were successfully logged by the server; due to the overload, it is likely that an even larger number of requests were present but not recorded. During the load spike, the system administrator who was responsible for the Web site was unable to login to the system remotely to clear up the disk log, and had to physically reboot the machine in order to regain control.

The most common approach to dealing with heavy load is to overprovision resources. In the case of a Web site, the administrators simply buy enough Web server machines to handle the peak load that the site could experience, and load balance across them. However, over provisioning is infeasible when the ratio of peak to average load is very high; it is not practical to purchase 100 or 1000 times the number of machines needed to support the average load case. This approach also neglects the cost issues which arise when scaling a site to a large "farm" of machines; the cost of managing a large cluster of machines is no doubt much higher than the multiplicative cost of the machines themselves.

Given that we cannot expect most Internet services to scale to peak demand, it is critical that services are designed to be well-conditioned to load. That is, when the

demand on a service exceeds its capacity, a service should not over commit its resources and degrade in a way that all clients suffer. Rather, the service should be aware of overload conditions and attempt to adapt to them, either by degrading the quality of service delivered to clients, or by (predictably) shedding load, such as by giving users some indication that the service is saturated. It is far better for an overloaded service to inform users of the overload than to silently drop requests.

Nevertheless, replication is a key aspect of service scalability, and is commonly employed using both scalable clusters to obtain replication within a service site [19], as well as in the wide area, as with content-distribution networks [20, 21, 22]. Despite such scaling, we expect the individual nodes within a scalable system to experience large load spikes. Our goal in this project is to complement the use of replication by addressing the problem of load management within an individual node. Therefore we focus on developing a general framework for authoring highly concurrent and well-conditioned service instances that can potentially be deployed in a replicated system.

1.3. Trends in the Internet Service Design Space

The systems challenge associated with robust Internet service design is magnified by two trends that increase the generality of services: rapid evolution of service logic and the drive for general-purpose platforms for hosting services.

1.3.1. Rapid Evolution of Service Logic

Internet services experience a high rate of change and innovation, driven by the constantly evolving Internet marketplace. Popular sites are constantly adding new functionality to stay competitive, a trend which is supported by the increasing ease of Internet service development using simplified Web scripting languages such as PHP [15] and Java servlets [23]. It is excessively time-consuming to carefully engineer

and tune each new service component to deal with real-world loads, and as a result, deployment of new service functionality often takes precedence over robustness.

1.3.2. General Purpose Hosting Platforms

Internet services are increasingly hosted on general-purpose facilities, rather than on platforms that are specifically engineered for a particular service. A number of companies are now providing managed Internet hosting services, which are based in replicated Internet data centers with carefully managed hardware and software platforms. Accordingly, the Internet services market is dominated by generic frameworks, often based upon scalable workstation clusters running commodity operating systems, using a standard software platform such as Java 2 Enterprise Edition (J2EE) [14]. In this environment, service authors have little control over the underlying medium upon which the service will be deployed and run.

As these trends continue, we envision that a rich array of novel services will be authored and pushed into the infrastructure where they may become successful enough to scale to millions of users. Several investigations are addressing the high-level aspects of service authorship, including naming, lookup, composition, and versioning [24, 25, 26, 27, 28]. Our focus is on the performance and robustness aspect of the problem: achieving well-conditioned performance on a wide range of services subject to huge variations in load, while preserving ease of authorship. We argue that the right way to approach this problem is through a generic service platform that manages load in a manner that is cleanly separated from the service-specific logic, while giving services an indication of load and the ability to participate in load-management decisions.

1.4. Challenges in Designing Robust Internet Services

Much work has focused on performance and robustness for specific internet services. However, with internet services becoming increasingly dynamic and

flexible, the engineering burden required to make services robust to heavy demands becomes excessive. Few tools exists that aid the development of highly concurrent, well-conditioned services; our goal is to reduce this complexity by providing general-purpose mechanisms that aid software developers in obtaining these properties.

An additional hurdle to the construction of Internet services is that there is little in the way of a systematic approach to building these applications, and reasoning about their performance or behavior under load. Designing Internet services generally involves a great deal of trial-and-error on top of imperfect OS and language interfaces. As a result, applications can be highly fragile—any change to the application code or the underlying system can result in performance problems, or worse, total meltdown.

1.5. Project Objectives

This project work proposes a new design framework for highly concurrent server applications, called the huge and secured online transactional system (HSOTS). HSOTS combines the different system architecture and programming models to manage the concurrency, I/O, scheduling, and resource management needs of Internet services.

This project work describes the design, architecture, and implementation of a HSOTS –based Internet services platform. This platform provides efficient, scalable I/O interfaces as well as several resource-control mechanisms, including load balancing, caching and multi-tier system. The objectives of the project work are

- To design a system which can handle complex and massive transactions that involve a lot of database interactions and have both public and private data.
- To design the system so that it can be used in online mode.
- To implement links between database and website for handling massive data.

- To ensure the stability of the system while handling the massive online data transactions.
- To provide interfaces with different credentials for different kind of input data and output request.
- To ensure security of the overall system from any sort of hacking

1.6. Organization of the Project Paper

This project work develops a system architecture that simplifies the development of highly concurrent and well-secured services. Dynamic resource control is used to ensure that the service stays its ideal operating regime despite large fluctuations in load. This project paper is organized as follows.

This project paper has total four chapters. Contents of the chapters are as follows. Chapter 1 describes general discussion on different terms involving web or multi-tier application. Chapter2 contains preliminaries of different clustering technology, caching technology and security. Chapter3 discusses design and implantation details of our system. Finally we conclude our work with recommendation for future works in Chapter 4.

Chapter 2

Preliminaries

2.1. Web Serving Overview

URLs are the most user-visible component of how the World Wide Web (Web) works. When a user wants to retrieve a Web page, all the necessary information for contacting the right Web server and making the appropriate request for the desired page is contained in an URL that the user can type in the address bar of the browser. URLs are composed by several parts as seen bellow

http://www.example.com:80/index.html

protocol://host:port/filePath

- **Protocol:** The protocol specifies which application layer (OSI layer 7) protocol the browser will use to communicate with the server. In the case of Web connections, the protocol will normally be **http** for traditional clear-text requests or **https** for secure encrypted requests. These two are the standard Web protocols, but current browsers usually understand others like **ftp** or **mailto**, even if they are not strictly part of the Web.

- **Host:** The host name will specify which Web server a request is directed to. It is a human understandable name that will be translated to an IP address by the DNS naming system used in the Internet. Once the browser obtains the IP address that the host name belongs to using DNS, it can directly address the Web server host.

- **Port:** The port component of the URL is an optional part required by the underlying network protocol layer, TCP. Even if the Web client will always use a port number when connecting to the server, the port is optionally in the URL: as different protocols have well-known ports that are chosen by default if no

specific port number is given. For example, HTTP uses 80 as its well-known port number.

- **File path:** It contains the path to a Web page or file in the Web servers file system, expressed in a form relative the document root of the Web server, not to the operating system root of the server's file system. The format for the path uses Unix conventions for separators. With the popularity of dynamic sites and Web applications, the path frequently doesn't specify the location of a physical file that exists on the servers file-system, instead it serves more as a Remote Procedure Call that the Web server will use to decide which part of the Web application logic to execute.

The initial URL that the Web browser user types in an address bar will eventually be mapped to a set of blocks in a file-system that a Web server process will read or a set of instructions that will generate a dynamic response.

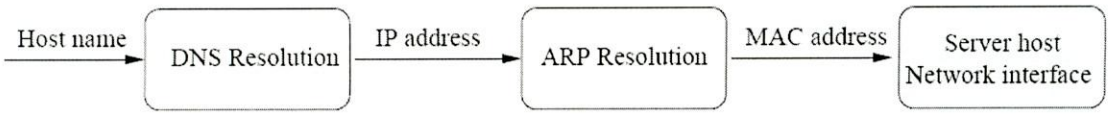

Figure 2.1: Web server address resolution steps

There are various steps in this process assuming that the final server is connected to an Ethernet network (see Figure 2.1)

- **DNS Resolution:** The Domain Name System (DNS) is the standard mechanism on the Internet for resolving a given host or domain name to an unique IP address. The IP address identifies one physical machine connected to the Internet.

- **IP Routing:** When the Web browser obtains the IP address from the name resolution subsystem, it sends an HTTP request to that IP address, by sending a

set of IP packets with the IP address of the host. These IP packets go through potentially several hops between Internet routers until they reach the sub-network that is configured as including the host with the IP address.

- **ARP Resolution:** When the request IP packets reach the last router before the host, the router will resolve the IP address of the host to an Ethernet MAC address. A MAC address is an unique identifier of the network card installed on the host. The mapping between IP and MAC addresses translation allows the Ethernet switch to send the request packets to the right network interface connected to it.

Once the request IP packets arrive to the Web server host, they get reassembled to build a HTTP request, and the server maps the URL to a file in the local file-system. The Web server will use then the standard operating system facilities to read the disk blocks needed for obtaining the contents of the file.

This process has several steps in which a resolution of a mapping between different naming and addressing schemes is performed: from host name to IP address, from IP address to MAC address and from requested URL to file-system block. These points in the process of handling a HTTP request are where Web clustering techniques can be introduced. For example, DNS round-robin will resolve a given host name to one of several IP addresses available, or a Web server can cache files in main memory instead of loading them from the file-system for each request. Several examples of cluster architectures are discussed in the next section.

2.2. Cluster Architecture

We will use a classification of different Web server clustering technologies that has been proposed explicitly by Steve Goddard and Trevor Schroeder [29, 30], but is common as well in the research literature. Most of the Web servers clustering technologies are transparent to the client browsers; the browsers are at no time aware

of the existence of clustering on the server side. However, not all clustering technologies are transparent to the web server software. While still being transparent to the clients, some clustering solutions are visible to the Web servers, and need specialized software at different levels in the server system.

2.2.1. Layer 4 Switching with Layer 2 Forwarding

In L4/2 clustering, the dispatcher and the servers all share a common cluster network address (referred as "Virtual IP address" by some vendors). All incoming traffic for the cluster address is routed to a dispatcher, using static ARP entries, routing rules, or some other mechanism. All the servers have a primary unique IP address, but share the cluster address with the rest of the servers and the dispatcher, which has the cluster address as its main IP address. Figure 2.2 shows an architecture of an L4/2 clustering.

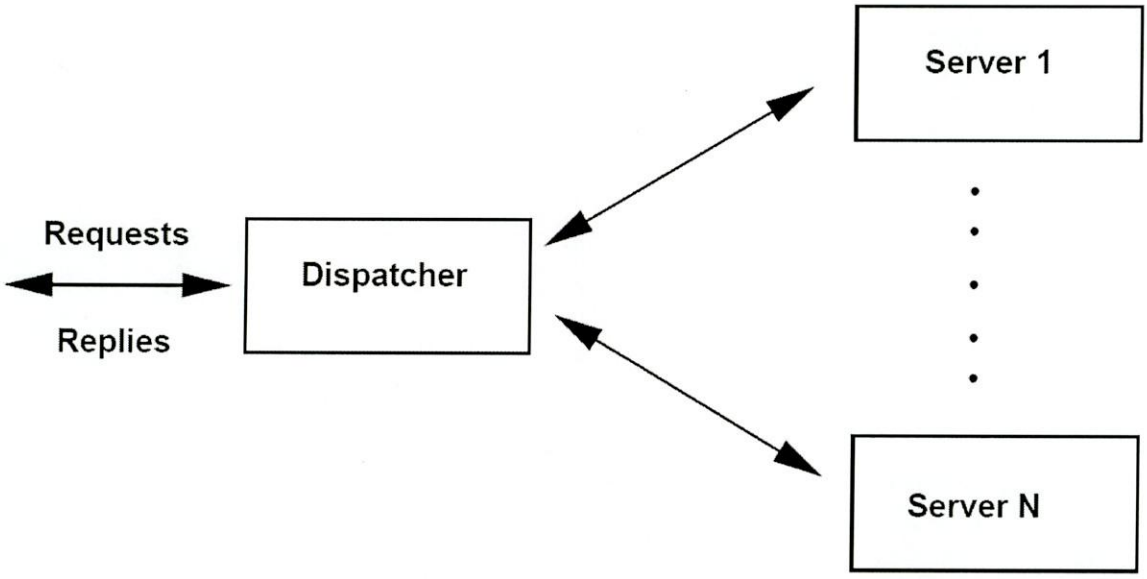

Figure 2.2: Layer 4/2 cluster architecture [31]

2.2.2. Layer 4 Switching with Layer 3 Forwarding

It is also known as "Load Sharing Using Network Address Translation (LSNAT)", and it is an Internet standard detailed in RFC 2391 [31]. Figure 2.3 shows an architecture of an L4/3 clustering. Examples of commercial implementations are Cisco's LocalDirector 400 series [32],

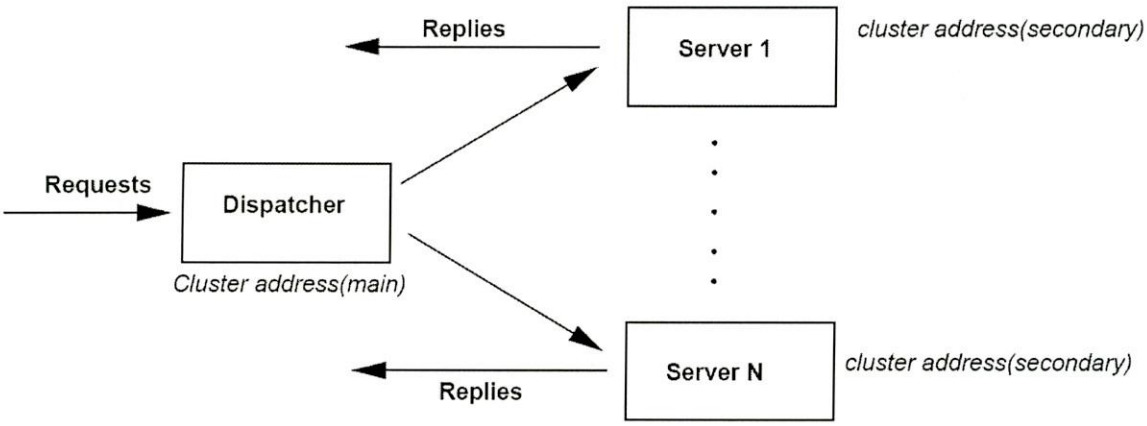

Figure 2.3: Layer 4/3 cluster architecture [31].

2.2.3. Layer 7 Switching

Also known as content-based switching or content-aware request distribution, it operates at the application layer (L7) of the OSI protocol stack. Commercial implementations that have this functionality are Nortel's Alteon ContentDirector [33], Foundry Networks' ServerIron family [34] and Cisco's CSS 11000 switches [32].

2.3. Server Selection / Request Distribution

2.3.1 DNS Round Robin

In a Server selection system that uses DNS, the DNS server will resolve the name of the service or domain to a different IP address using different algorithms such as load average of a simple round-robin. This mechanism is specified in RFC

1974 [35]. DNS is the naming system used in the Internet; as such it supports massive scalability by making use of hierarchical organization of naming information and caching at several levels of the naming system.

DNS-based server selection [36] techniques base their effectiveness on setting a very low (down to 0) value for the TTL (Time To Live) field of their responses, minimizing caching of the name resolutions, which allows the implementation of round-robin or other load balancing algorithms, as well as more complex solutions that try to resolve to IP addresses that will give better service based on the physical location of the client. If clients contact the DNS server for each request, the DNS server that is in front of a web cluster can control the back-end server to which each request gets sent.

The problem with this approach is that all the elements involved in resolving a name to an IP address, from the browser to the root DNS servers, use caching for improving performance and response times to users. Between DNS servers, the validity of a piece of information is given by a TTL field that specifies how long that information can be cached before refreshing it from the original or "authoritative" source, and web browsers are known to cache resolution information for about 15 minutes. This caching at several layers is what makes DNS so scalable.

It has been shown in that DNS-based server selection schemes actually have a negative impact on client-perceived web access latency. Without caching, the client should always need to contact the DNS server before each request, increasing name resolution overhead by up to two orders of magnitude. This problem is aggravated by the delays caused by DNS queries needed for embedded objects in HTML pages such as images.

2.3.2 Weighted Round Robin (WRR)

Weighted Round Robin (WRR) is the most common distribution algorithm found in commercial web load-balancing solutions, and although commercial

vendors frequently support other more complex server selection schemes such as URL-aware selection, WRR and its variants are still very popular and widely used, thanks to their simplicity and scalability.

In this server selection scheme the front-enback-end dispatcher is usually a Layer 4 switch that distributes request to back-end servers using only load-balancing requirements. As its name implies, it is a variant of Round-Robin that instead of "blindly" dispatching requests iteratively to the available back-end servers, has a weighting introduced for each server, normally calculated as a function of the CPU, disk and/or memory use. The weight of each server in the load-balancing algorithm is reevaluated periodically.

The main benefits of WRR are that it obtains low idle times in the back-end servers and achieves a very well balanced load distribution. However, the main problem of WRR is that it shows a high cache miss ratio. In web servers, being able to cache the working set (the set of web objects that the server accepts requests for) is critical, as the throughput of the current disk devices is much lower than the expected throughput of network servers.

For web server clusters that have big working sets this supposes a problem, as each individual node main memory cache has to fit the entire working set, which is frequently impossible. When the node caches are not able to hold all their working set, the throughput is low, as the cluster is not taking advantage of its aggregated cache size. WRR scales well to high-traffic sites but don't provide greatly improved response rates when the working set bigger than the node caches.

Different distribution schemes based on Layer 7 switching try to overcome this problem by distributing requests to nodes that are more likely to already have the requested object in the cache, effectively partitioning the working set and making it possible for individual nodes to keep their respective partitions in cache. One example of this design is Locality-Aware Request Distribution [37].

2.3.3 Weighted Least Connections (WLC)

Weighted Least Connections (WLC), is a common alternative to WRR as a request distribution algorithm. It is a weighted variant of Least Connections (LC), which bases its distribution decisions on the number of open TCP connections each back-end server has open. The rationale behind this idea is that if the server has several open connections, it means that it is busy servicing requests, and the more open connections it has, the less new requests it should receive.

The weighted version of LL, WLC, goes one step further and includes a weight in the distribution algorithm. This weight is calculated and updated the same way as with WRR, using variables such as disk, CPU or network resources usage. This algorithm is available for several load-balancing solutions, in software Linux Virtual Server [38] is an example and in hardware Cisco Catalyst web switches [32] provide it as well.

2.3.4 Locality-Aware Request Distribution (LARD)

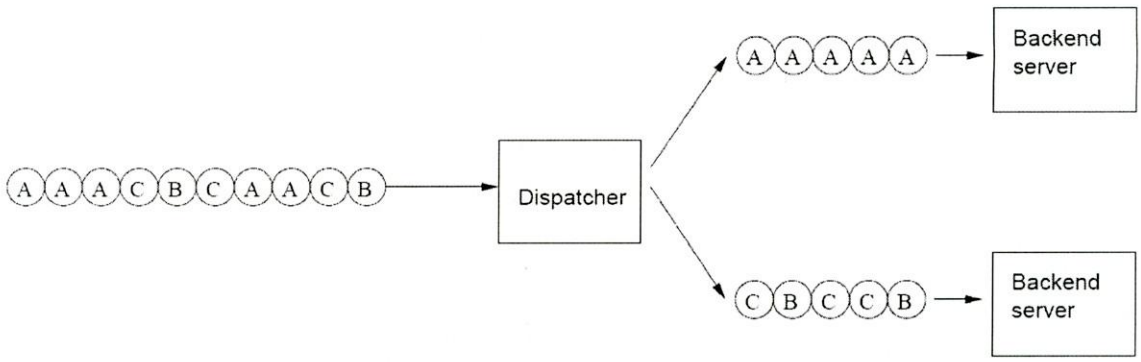

Figure 2.4: LARD operation [37]

Locality-aware request distribution (LARD) [37] is a form of content-based request distribution described in. It uses Layer 7/HTTP information. The main advantages of content-based request distribution compared to schemes like WRR or WLC are:

- Better throughput thanks to a lower number of main memory cache misses in the back-end servers.

- Ability to partition the working set over the different back-end servers, having more available disk space.
- Ability to use specialized back-end nodes for services like multimedia streaming or dynamic web applications.

LARD [37] focuses on obtaining the first of the advantages cited above, improved cache hit rates. In figure 2.4 the operation of a basic locality-aware dispatcher is shown, by distributing requests in this way, there is the risk that the load on the back-end servers becomes unbalanced, resulting in worse performance than with simpler round-robin algorithms.

Thus, LARD [37] has to provide a solution that simultaneously achieves load balancing and high cache hit rates on the back-end servers.

2.3.5 Workload-Aware Request Distribution (WARD)

Workload-Aware Request Distribution (WARD) is a content-aware request distribution strategy that intends to improve some shortcomings of the scalable LARD architecture. The basic idea behind WARD is that "by mapping a small set of most frequent files to be served by multiple numbers of nodes, we can improve both locality of accesses and the cluster performance significantly" [39].

In a cluster using LARD of n nodes, only 1/n of the requests will be handled locally forwarding the other (n-1)/n requests, a relatively expensive operation that can introduce a forwarding overhead that limits the performance and scalability of the cluster.

Additionally, Web server workload studies have shown that 90% of the web requests are for only 2-10% of all the accessed documents, constituting less than 5% of the total working set. WARD aims to minimize forwarding overhead from handling the most frequent files while keeping the benefits of the optimized use of the overall cluster main memory caches that LARD provides.

WARD chooses a set of most frequently accessed documents, called core that will be served by all the cluster nodes and partitions the rest of the files to be served by the backend servers. It uses an algorithm called ward-analysis [39] that identifies the optimal core for a given workload and system parameters such as the number of nodes, main memory cache size, TCP hand-off overheard and disk access overhead. The core is proposed to be computed once in a day, so that the cluster will always use the values obtained from previous day's workload.

This basic form of WARD is very similar to the scalable LARD architecture, and it would use the LARD distribution algorithm for the files that are not in the core. The other change proposed in WARD is to decentralize the dispatcher, by using a pre-computed partitioning of the working set based on the workload of the previous days. The scalable

LARD architecture introduces the idea of a co-located distributor that resides in every backend server, but still keeps a centralized dispatcher that every node has to contact to decide which backend server will handle the request. With a decentralized dispatcher, the overhead of a unique dispatcher is eliminated and better scalability is achieved. This approach will work well assuming that the core is reasonably stable, if workload stability is not possible, the simpler "dynamic" centralized dispatcher will be a better solution most of the time.

Simulation tests [39] show that WARD provides performance improvements of up to 60% increased throughput compared to round-robin strategies and up to 50% compared to locality-based strategies such as LARD. The use of a distributed dispatcher together with previous day's workload information improves throughput furthermore for typical stable workloads.

2.3.6 Parallel Pull-Based LRU (PPBL)

Most web cluster architectures use a push approach to request distribution, the dispatcher decides which backend server should handle a given request based on

layer 3, 4 or 7 information and then sends or "pushes" the request to the chosen server. WARD [39] introduces a distributed dispatcher, but it does so at the expense of having static dispatching information that is updated with previous day's information instead of more dynamic algorithms like LARD [37]. The PPBL prototype web proxy uses a Distributed Shared Memory (DSM) based cluster interconnected by a Scalable Coherent Interface (SCI) network. It uses certain features of the DSM architecture to obtain better performance, and it is uncertain if this design would be able to be generalized to be appropriate for non-DSM clusters.

Parallel Pull-Based LRU [40] is an architecture that aims to distribute the dispatcher while still having the advantages of a dynamic content-based request distribution algorithm. As its name implies, it uses a "pull" approach to request distribution, where backend server nodes don't just receive requests but take part in a parallelized request distribution algorithm. This algorithm uses the basic idea that the backend servers, pull the requests that they should handle from an incoming request queue that resides in the front-end node.

The algorithm works as follows: each backend node selects a pending URL from the incoming queue and checks if it is in its own URL list. If the matching entry is found, the backend modifies the incoming queue to signal that it is handling the request and sends the reply directly to the client. The front-end can delete the entry from the incoming queue once a backend node has decided to handle it. If the entry is not found by any of the backend servers, the front-end will choose the less loaded node to handle it. The load information is kept in the DSM and periodically updated by each backend. If the backend that has found the request in its local URL list is overloaded, it can choose a less loaded in the entry's server list or if there is none, it can forward it directly to the less loaded node in the cluster keeping all the information about the URL local, accessible to the other node through the DSM.

A first implementation that used the above design proved not to be very scalable. Throughput doesn't improve anywhere near linearly by adding nodes to the cluster,

and the synchronization locks required by the request distribution algorithm prevent the cluster from performing better. A second implementation is proposed by the authors that will limit the number of locks that are necessary for each request. This implementation uses new data structures and an special SCI mapping that performs a fast atomic fetch and increment on each access that can be used by the backend nodes to signal when they have finished evaluating if a given request is for them. Good scalability is achieved by making use of these low level capabilities of the middleware; a cluster of 7 nodes has a 580% better throughput than a single node server.

PPBL is a very interesting design that takes a new approach to request distribution, both because it uses a pull based approach whereas most other cluster architectures use a push design as well as because it makes use of high-speed SCI networks and DSM middleware to achieve better scalability. It would be interesting to evaluate if the ideas behind PPBL are applicable outside of the specific implementation technologies used by the authors, as they had to resort to low-level optimizations to achieve reasonable scalability.

2.4. Advantages of Caching Dynamic Content

The solution using data binding technique is simple and effective to use. It also does the caching (binding) in the browser, which also is preferred.

On the other hand looking at the restrictions imposed on a solution, using Java or ActiveX should be avoided and since a DSO can only be implemented as a Java applet or ActiveX component, it is simply not adequate. This method has also the severe disadvantage of being limited to Microsoft and the Internet Explorer.

Using the XML DOM will be as effective and yet browser independent. As for the downside, since the DOM is so new, at the time of this writing, neither Netscape Navigator nor Internet Explorer support the XML DOM Level 1 or have a very

limited support. Although it is a matter of time before both will, there is no guarantee that both will follow the DOM Level 1 recommendation from the W3C. A comparison can be made regarding the recommendation that the W3C made for DHTML, where despite these recommendations Netscape and Microsoft have very different implementations.

The XSL recommendation is not ready either at this time, which so far has resulted in Microsoft adding own functionality to the XSL and its processor.

To conclude, given the requirements and restrictions imposed, using the XML DOM is the simpler and the better choice for the BIIS system and will be used for the prototype.

2.5. Communication Security Fundamentals

2.5.1. Security Objectives

When looking for a definition of security for a certain system, which shall serve as a base for a security policy or a system specification, we can distinguish three main perspectives.

The first perspective deals with the question how the attacker reaches the target information system: in person, in which case the countermeasures are collectively called physical security, or via the electronic network, in which case the countermeasures are network and system security. This paper is not concerned with physical security.

The second perspective looks at what fundamental types of threats the system is to be secured against, or in other words, the security objectives that is to be achieved. The eight security objectives explained in the following offer a framework for categorizing and comparing the security mechanisms of various systems:

- **Confidentiality:** The confidentiality objective refers to preventing disclosure of information to unauthorized persons or systems.
- **Integrity:** The integrity objective refers to preventing undetected modification of information by unauthorized persons or systems.
- **Availability:** Availability refers to ensuring that unauthorized persons or systems cannot deny access or use to authorize users.
- **Authentication:** Authentication is concerned with determination of the true identity of a system user and mapping of this identity to a system-internal principal (e.g., valid user account) by which this user is known to the system.
- **Authorization:** The authorization objective, also known as access control, is concerned with preventing access to the system by persons or systems without permission to do so.
- **Auditability:** Auditability is concerned with being able to reconstruct the complete history of the system behavior from historical records of all (relevant) actions executed on it.
- **Nonrepudiability:** The nonrepudiability objective refers to being able to provide irrefutable proof to a third party of who initiated a certain action in the system, even if this actor is not cooperating.
- **Third-party protection:** The third-party protection objective refers to averting damage done to third parties via the IT system,

2.5.2 Types of Attacks

An intentional violation of a security objective is called an attack. Attacks may either be initiated by persons outside the organization or by insiders. We distinguish between targeted and untargeted attacks. Targeted attacks intend to harm a specific communication system or type of system, e.g., for purposes of industrial espionage,

warfare, or terrorism. Untargeted attacks victimize any vulnerable system they discover. Targeted attacks are typically preceded by a phase of gathering information about the target, e.g., using online and offline available references, as well as dedicated tools for discovering vulnerable systems on a network. Some common types of attacks are the following.

- **DoS:** The goal of the attacker is to decrease the availability of the system for its intended purpose.
- **Eavesdropping:** The goal of the attacker is to violate the confidentiality of the communication, e.g., by sniffing packets on the LAN or by intercepting wireless transmissions.
- **Man-in-the-Middle:** In a man-in-the-middle attack, the attacker acts toward both end points of the communication as if the attacker were the expected, legitimate partner.
- **Breaking into a System:** A break-in usually involves the consecutive penetration of multiple subsystems and the step-wise elevation of the privileges of the attacker.
- **Virus:** A virus-based attack manipulates a legitimate user to bypass authentication and access control mechanisms in order to execute the malicious code injected by the attacker.
- **Trojan:** A Trojan is a virus where the malicious functionality is hidden behind functionality that is desired and used by the user.
- **Worm:** A worm is malicious code whose propagation mechanisms rely on automatic exploration and exploitation of vulnerabilities in the targeted system, without involvement of any user.

Chapter 3

Design and Implementation

In this chapter we propose a new server architecture, the huge and secured online transactional system (HSOTS), which is designed to enable high concurrency, load conditioning, and ease of engineering for Internet services. HSOTS decomposes an application into a multi-tier and load balancing system. HSOTS makes use of dynamic resource control to allow applications to adapt to changing load and resource conditions.

Web clusters have been traditionally implemented by "growing" single-node Web server architectures instead of being designed from scratch as a system that takes advantages of a clustering architecture, clustering has been more "bolt-on" Web servers than "built-in".

This legacy reflects on the design of commercial Web clustering and load balancing solutions. A very common deployment scenario for a commercial clustered Web server configuration is shown in Figure 3.1, where the dispatcher is the central point of contact for all the clients and it distributes requests to the backend Web servers using layer 4 switching with either layer 2 or 3 forwarding. There are some advantages to this design:

- Backend nodes are standard Web servers, keeping the same configuration and administration needs of a single-node Web server, minimizing the system administration overhead of moving from a single Web server to a cluster.
- The front-end dispatcher can be a dedicated device with embedded software that is very efficient at the request distribution and load balancing tasks it performs.

- Every component in the cluster is transparent of the others, it is a loosely-coupled architecture where each component has no or minimal knowledge of how the others work, providing both fault tolerance and simpler administration and deployment, as well as allowing heterogeneous systems to work together.

The architecture in Figure 3.1 is simple and provides some degree of scalability. However, goals of higher scalability and better use of resources available on the backend nodes require different solutions. Most of the commercial state-of-the-art Web switch/dispatcher products have the optional functionality of taking in account application layer (OSI layer 7) information, such as the requested URL or the current session.

Application layer data awareness (layer 7 switching) is important for achieving better request distribution , but commercial dispatchers pay a big performance price when doing it as they have to be both transparent to the clients as well as to the backend Web servers.

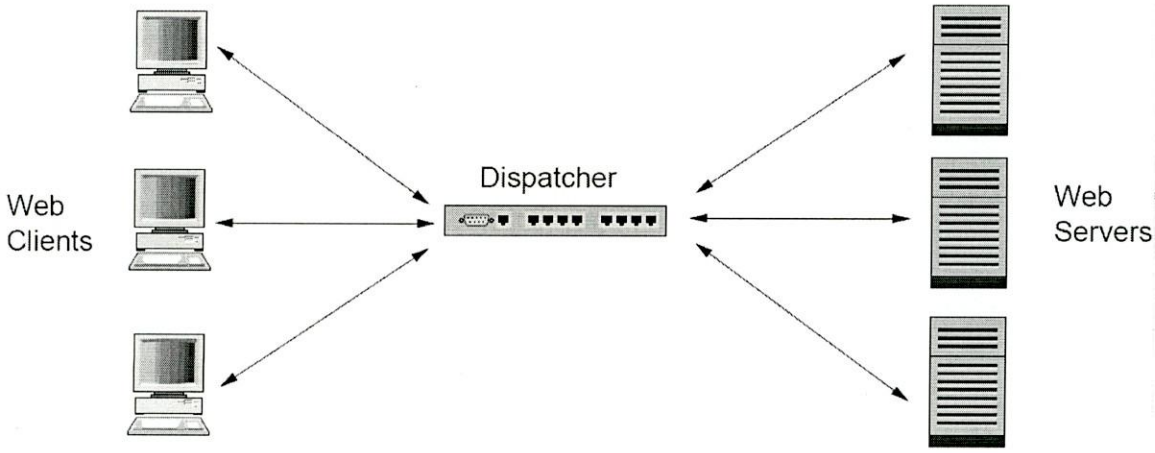

Figure 3.1: Common Web cluster deployment scenario.

The fact that these clustering solutions are highly transparent between their own components (dispatchers and backend nodes) as well as to the outside world (Web browsers) makes it difficult to implement highly scalable clustered Web servers with them.

In this section the design and implementation of a clustered Web server that uses commodity off-the-shelf (COTS) hardware and distributed software components is described. Designing all the components from scratch allows more flexibility for using state-of-the art algorithms and controlling resources globally in the cluster, achieving better load balancing and efficiency, which should lead to good scalability. This Web cluster could be considered as a Web server with an integrated content-aware layer 7 switch and global cache management.

All the clustering software discussed in this project work has been implemented in Java. It is not very common in the research literature to use Java as the platform for implementing prototype Web servers or proxies. However, there are popular commercial application server products implemented in Java that prove that it is a viable platform for high-performance network servers. Furthermore, with the release of Java Standard

Edition version 1.5, non-blocking I/O support is made available to Java developers both under Unix as well as Win32 operating systems.

This chapter is organized as follows: first the overall architecture of the clustered Web server is described; next each of the components' (front-end/dispatcher, node manager and cache manager) design is discussed, giving details of their different implementation approaches and how they interact with the rest of the cluster components.

3.1. Cluster Architecture

Three main components form the clustered Web server discussed in this project work:

Front-End: The front-end is the process that will be the single point of contact for all the clients that want to issue an HTTP request to the server and receive a response from it. Every byte of data that the server receives or sends has to go

through it, so it is the most performance-sensitive component of the server. It has three main responsibilities:

- Wait for connections on a TCP port and accept requests from Web clients, keeping track of the open client connections.
- Obtain the requested URLs and decide which backend node will handle them.
- Receive and send back the responses generated by the backend nodes to the appropriate clients.

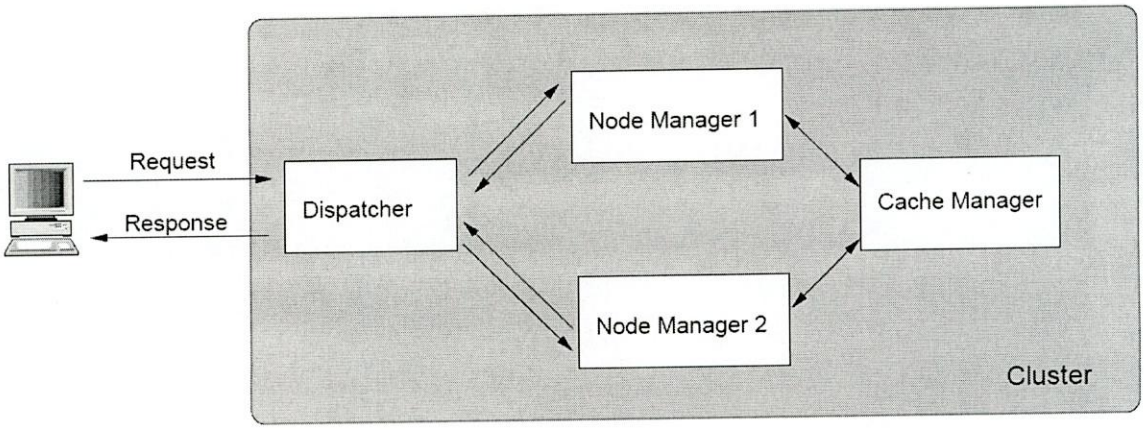

Figure 3.2: Logical view of the components in a running cluster.

Cache Manager: The cache manager is the component that holds cluster-wide global information about the state of the backend nodes' main memory cache. Its main responsibility is to keep a mapping of files and the nodes that have them them in their cache, so that when a backend node has a request for a file that is not in its local cache it can retrieve it directly from another node's main memory instead of accessing the file-system. The reason for this design is that with the current network interconnection technologies it should be faster for a node to retrieve some data from another node's main memory across the network than reading it from the file-system.

Node Manager: The node manager is the main "worker" process in the cluster. Each backend node runs one or more node manager process. Its main responsibilities are:

- Retrieve requests assigned to it by the front-end, obtain the response data and send it back to the front-end.
- Keep a local main memory cache of files, with its associated cache replacement policy.

There are two possible views that can be used when describing how these components (or instances of them) are organized in a running clustered Web server to handle client requests. Figure 3.2 shows a logical view where one front-end, two node managers and one cache manager work to accept a request, route it through the cluster software and send the right response back to the client.

Figure 3.3 describes the same cluster configuration as Figure 3.2 but from a physical point of view, showing how the component processes could be distributed in different computers. In this case there is one system dedicated as front-end, another one running the cache manager process and two nodes running one node manager process each. This configuration is the one that will be evaluated in this project work dissertation, with the only difference of the number of systems running node manager processes.

The server software is flexible with respect to the physical distribution of its components. It requires the existence of one front-end, one cache manager and one or more node manager, but does not require them to be physically distributed among the nodes of the cluster in a certain way. It is possible for example to have all the components running in the same system, even though that would obviously not be recommended.

Another aspect in which the cluster software offers high flexibility is that different implementations of its subsystems can be used by specifying a command

line option at startup time. For example, there are multi-threaded and non-blocking I/O implementations of the front-end server, and there are different request distribution and cache replacement policies available as well. It is a framework that can easily integrate new implementations for most of its parts, with clearly identified interfaces between the different subsystems so that they can inter-operate.

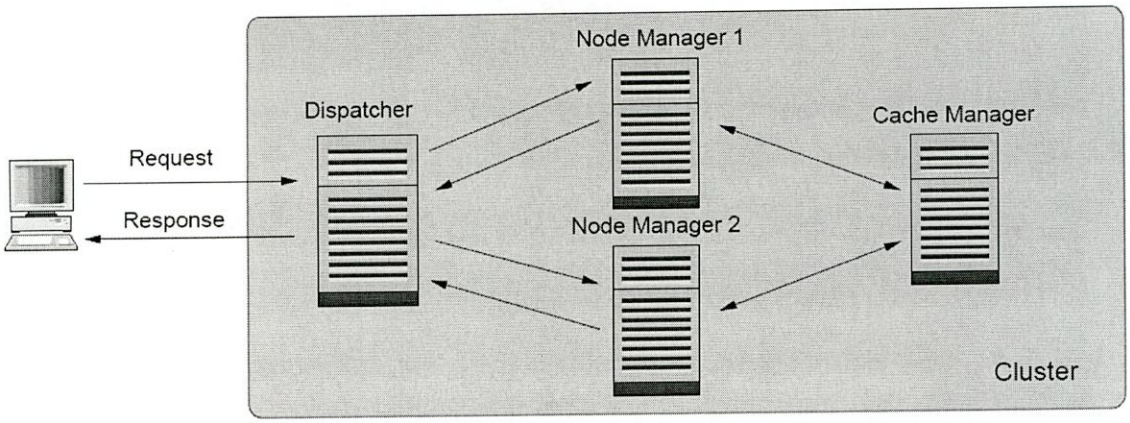

Figure 3.3: Physical view of the components in a running cluster.

3.2. Caching Dynamic Content

The main task of applications on the server's side is to get user parameters. Based on these, the applications must dynamically process and send back appropriate information to the user. Hereby, many security problems can occur during this process, e.g. buffer overloads or the execution of system commands.

To solve the problem, it is recommended use XML Document Object Model (DOM) Level 1 that the World Wide Consortium (W3C) has recently agreed on.

3.2.1. Document Object Model (DOM) Level 1

The Document Object Model (DOM) Level 1 is an API for XML and HTML documents. This interface, which is both platform- and language-independent, allows programs and scripts to dynamically access and updates the structure and style of documents. The DOM Level 1 provides a set of objects for representing XML

documents and a standard model of how these objects can be combined. In the DOM Level 1, documents have a logical structure, which is very much like a tree; to be more precise, it is like a "forest" or "grove" as the W3C puts it. It does not however, specify that documents must be implemented as tree or a grove. As an object model DOM Level 1 identifies:

- The interfaces and objects used to represent and manipulate a document
- The semantics, attributes and behavior of these interfaces and objects
- The relationships and collaboration among these interfaces and objects

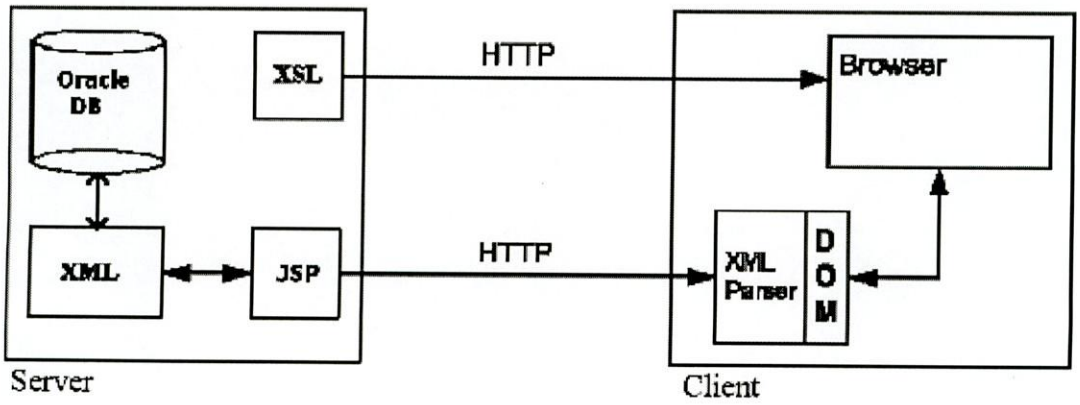

Figure 3.4: Solution using XML DOM.

The Document Object Model originated as a specification to allow JavaScript and Java programs to be portable among web browsers. The DOM Level 1 is strongly influenced by the DOM used in Dynamic HTML. Despite the strong influence from DHTML DOM, DOM Level 1 does not implement all of DHTML DOM. DOM Level 1 is also not a way for persisting objects in XML or HTML documents. Instead of describing how objects may be represented in XML and HTML documents, the DOM Level 1 describes how these documents are represented as objects, so that they may be used in object oriented programs. The Document Object Model Level 1 currently consists of two parts:

(1) DOM Core, which provides

- A low-level set of fundamental interfaces that can represent any structured document, and
- Extended interfaces for representing an XML document.

(2) DOM HTML, which Provides

- additional, higher level interfaces to provide a more convenient view of an HTML document.

3.2.2. Solution in BIIS

By using the Document Object Model Level 1 it will be very easy to manipulate XML documents. Using this idea, the pages sent to client, on the BIIS system, would mean that would be in XML. Once the client has received the XML documents it uses the XSL documents belonging to the XML pages in order to transform them to HTML documents so that the browser can render them. Now, any manipulation (e.g. sorting) will be done on the client in the following order:

- The XML document tree is accessed by the DOM Level 1 and manipulated by a second XSL document.
- The manipulated output (a new XML tree) from the second XSL document is then rendered by the first XSL document.
- The browser renders this new HTML document.

This solution is demonstrated in Figure 3.4

3.3 Multi-tier Architecture

3.3.1 Three-tier Architecture

Three-tier architecture is any system which enforces a general separation between the following three parts[41]: Client Tier or user interface, Middle Tier or business logic, and Data Storage Tier.

Applied to web applications and distributed programming, the three logical tiers usually correspond to the physical separation between three types of devices or hosts: Browser or GUI Application, Web Server or Application Server, and Database Server (often an RDBMS or Relational Database).

However, inside of the application server, there is a further division of program code into three logical tiers. This is kind of fractal: the part (app server object design) resembles the whole (physical system architecture). In a classic JSP/Servlet system, these objects are usually implemented as: JSPs or Servlets responsible for creating HTML or WML user interface pages, Servlets or JavaBeans responsible for business logic, and Servlets, JavaBeans, or Java classes responsible for data access. These objects usually use JDBC to query the database.

The precise definition of "tiers" can vary widely depending on the particular needs and choices of an application designer. However, they all maintain the general division of client-logic-storage.

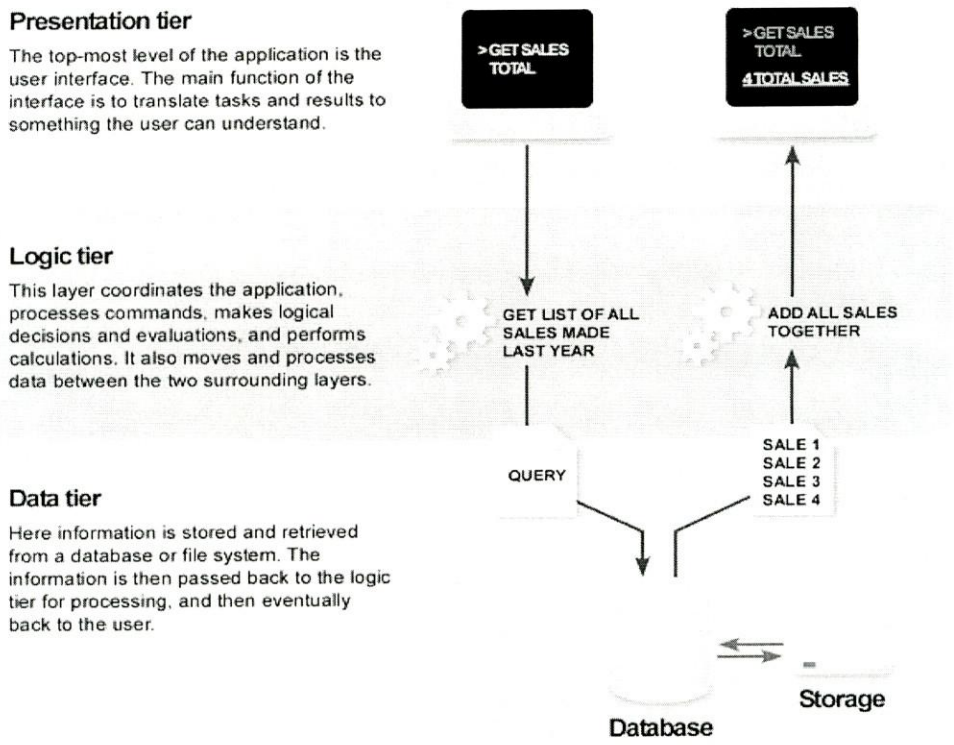

Figure 3.5: Typical Three Tier Architecture [41].

If the architecture contains more than three logical tiers -- for instance, multiple data feeds, multiple transactional data sources, and multiple client applications -- then it is typically called "N-tier" or "Distributed" architecture.

3.3.2 Solution in BIIS

In an institutional system there are mainly two types of data which can be accessed in public and private domain. For this reason we propose two three tier architectures for our system. Figure 3.6 and 3.7 show public and private domain three-tier architectures.

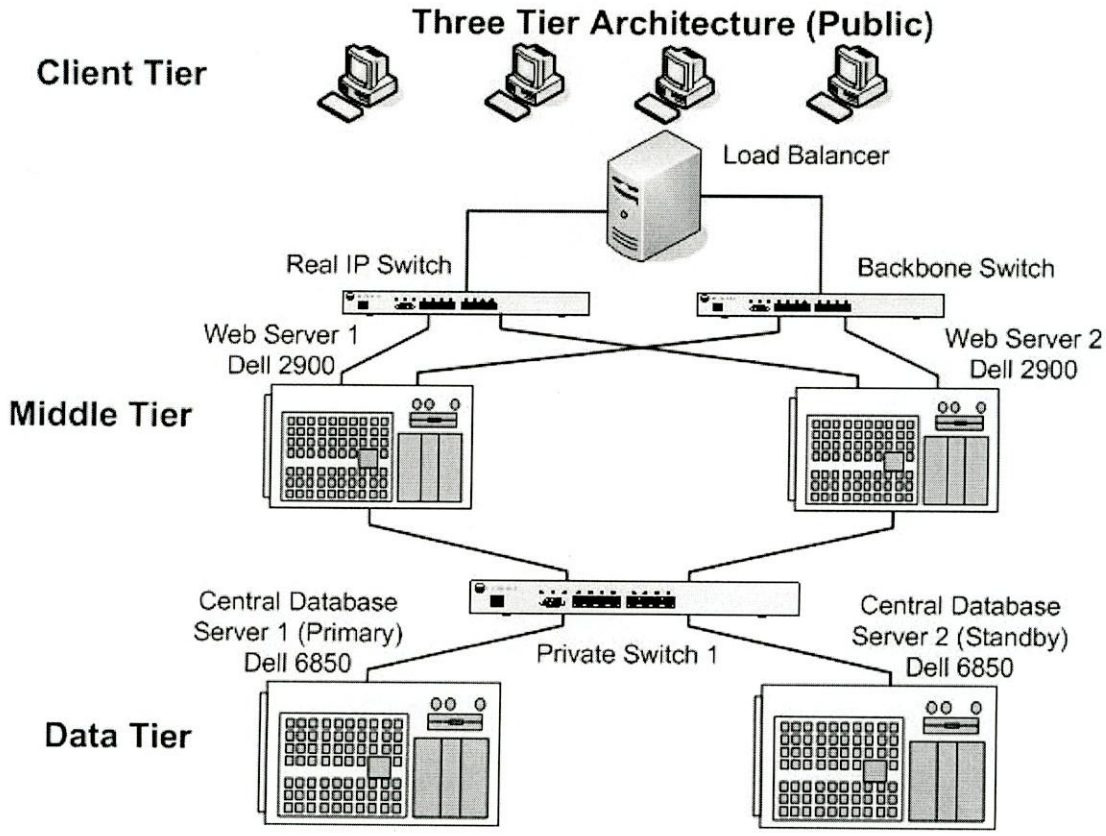

Figure 3.6: Public domain three-tier architecture.

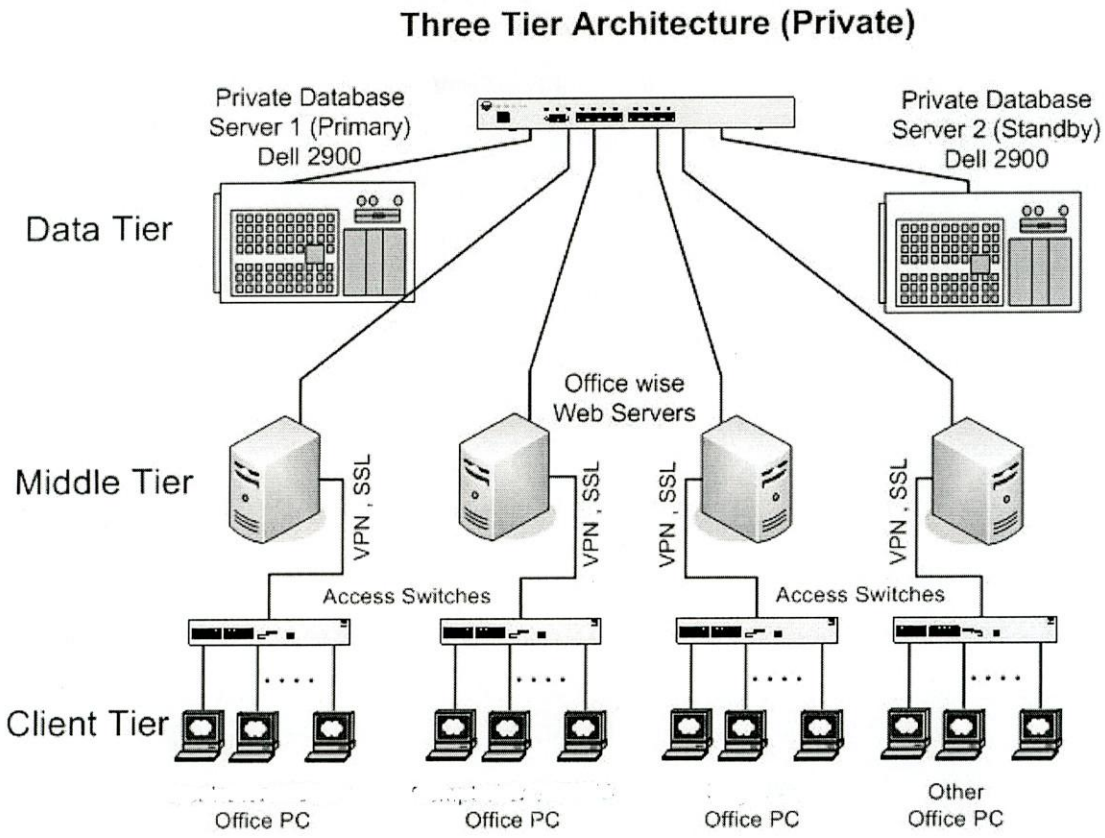

Figure 3.7: Private domain three-tier architecture.

Back-to-back Concatenation: To alleviate the security problem, to maintain the confidentiality of some restricted data, and to solve data between public and private domains two there-tier architectures are connected back-to-back to make a multi-tier architecture that expands the middle tier and database layer into multi-layer architecture, wherein a single Web server is replaced with multiple Web servers to increase performance of the middle tier while keeping the functionality transparent from the first layer clients. Thus we propose a six-tier architecture for our Huge and Secured Online Transactional System (HSOTS). We divide the database into following two tiers.

Private Database which contains only most secured information like exam data. Web or middle tier application can fetch this via a non-editable view through public database.

Public Database which contains only non secured data like exam routine, class routine, advising information etc. This database can be accessible only via Web Application server. Figure 3.8 shows our proposed six-tier architecture for a typical Huge and Secured Online Transactional System (HSOTS) like BIIS.

3.4 Implementation of SSL

The most dominating factor in the growth of the Internet in recent years has been the success of the World Wide Web (WWW). The Secure Sockets Layer (SSL) solution, an initiative of Netscape Communications, is an attempt to provide security to the WWW. It can also be used to secure other applications such as telnet and ftp.

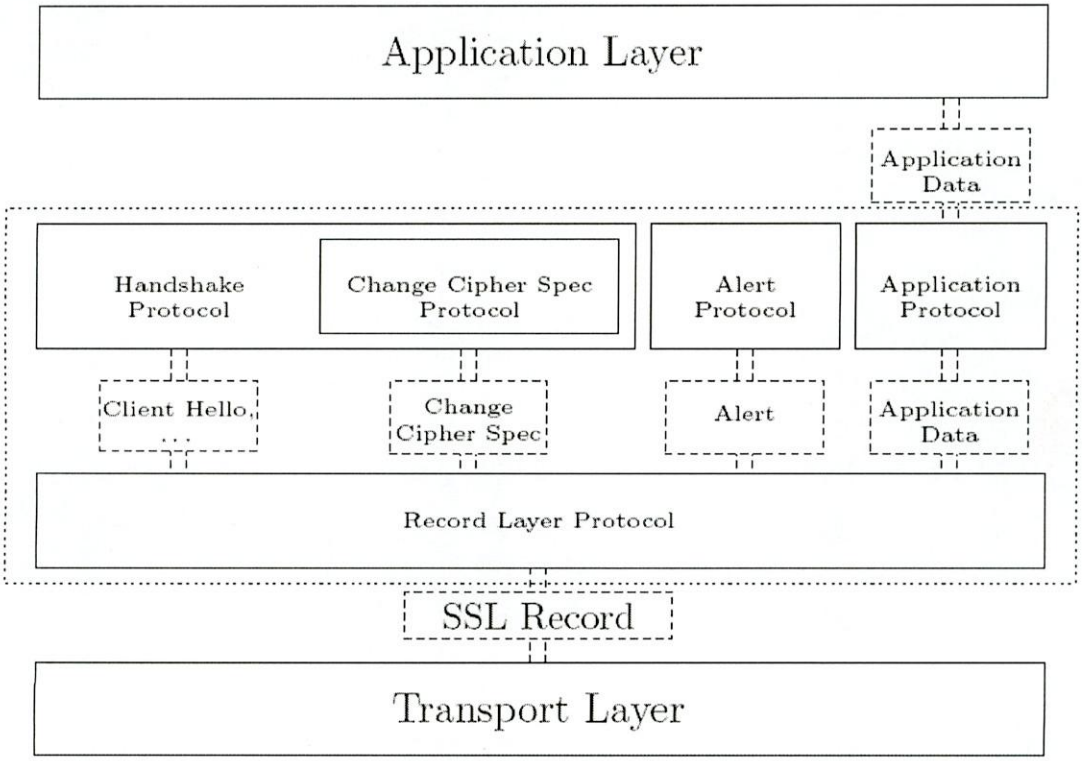

Figure 3.9: SSL structure

Typically, not all of an application's data is equally sensitive. SSL is bottleneck for a web application, if is not configured properly. If we implement SSL in whole system then the system will become slow. For this reason BIIS use SSL in some portion of the system which has confidential information.

Chapter 4

Conclusion

Today's Web applications need a platform that is fast, open, maintainable, straightforward, stable, and secure. A Huge and Secured Online Transactional System (HSOTS) meets these requirements, and provides a logical migration path from the proven and widely deployed multi-tier technology.

This project addresses a critical issue in the design and deployment of services on the Internet: that of implementing secure and massively scalable services that are robust to huge variations in load. We argue that traditional programming models, based on thread-driven and event-driven concurrency, fail to address the scalability and overload-management needs for large-scale Internet services. This domain gives rise to a new set of systems design requirements that are less evident in systems targeted at low levels of concurrency and load.

Our primary goal is to develop a system architecture that simplifies the development of highly concurrent and well-secured services. In the Huge and Secured Online Transactional System (HSOTS), services are constructed as a single application which can maintain huge load securely. Dynamic resource control is used to ensure that the service stays within its ideal operating regime despite large fluctuations in load.

The system is designed based on the present course registration system of BUET as a model University. Considering the International exposure and convenience, BUET has been selected as the model University.

The course registration process of students, curricula of different departments, course offering criteria and other relevant information of BUET is collected and reviewed. Based on the findings and extensive analysis a system is formulated and implemented. The system is tested in public domain against massive load and

monitored carefully for errors and bugs. Finally the bugs and error are fixed to make the system secure and stable. At the same time, the system is also tested in private network which needs the same database with confidential data transactions like exam data.

Our prototype of HSOTS, called BUET Institutional Information System (BIIS), is implemented in Java and makes use of native code bindings to provide non-blocking I/O primitives. We have found Java to be an excellent language for Internet service development, which is greatly facilitated by strong typing, built-in threading, and automatic memory management. Despite expectations to the contrary, we have found that Java has not had a negative impact on the performance and scalability of services developed using the BIIS framework.

4.1 Future Works

The future work may involve enabling alumni to access the application. The alumni would also be able to see different information of the database which will be depend on final decision made by the Bangladesh University of Engineering and Technology (BUET), our model university.

In this project work we showed six-tier system architecture for the concurrent and well-secured services. It will be a challenge to design system architecture with less number of tiers by keeping the system highly concurrent, well-secured and confident to the sensitive users.

In this paper we have tried to provide a solution to the question how to add semantics to a typical Web application that is developed using JSP-Servlet technology. But what still stays unanswered is what to do with all these legacy applications that may be developed using other technologies? This question is a research challenge.

References

[1] Amiri, K., Sprenkle, S., Tewari, R., Padmanabhan, S., "Scalable Consistency Maintenance for Edge Query Caches." Web Content Caching and Distribution, F. Douglis and B. Davison (Eds), Kluwer Academic Publishers, 2004.

[2] Urgaonkar, B., Shenoy, P., "Cataclysm: Policing Extreme Overloads in Internet Applications", Proceedings of the 14th International Conference on World Wide Web, ACM, Chiba, Japan, 2005, pp. 740-749.

[3] Manjhi, A., Ailamaki, A., Maggs, B.M., "Simultaneous Scalability and Security for Data-Intensive Web Applications", Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, USA, 2006, pp. 241-252.

[4] Welsh, M. D., "An Architecture for Highly Concurrent, Well-Conditioned Internet Services", PhD Thesis, University of California, Fall 2002.

[5] Yahoo! Inc. Yahoo! Reports Fourth Quarter, Year End 2001 Financial Results. http: //docs.yahoo.com/docs/pr/4q01pr.html.

[6] America Online. Press Data Points.http://corp.aol.com/press/press_datapoints.html.

[7] Superior Court of Santa Clara County, California. Cooper v. E*Trade Group, Inc. Case No. CV770328, http://www.cybersecuritieslaw.com/lawsuits/ primary_sources/ cooper_v_etr%ade.htm, November 1997.

[8] Wilcox, J. Users say MSN outage continues, ZDNet News. http://zdnet.com.com/ 2100-11-269529.html, July 2001.

[9] Wald, L. A., Schwarz, S., The 1999 Southern California Seismic Network Bulletin< Seismological Research Letters, 71(4), July/August 2000.

[10] W. LeFebvre, CNN.com: Facing a world crisis. Invited talk at USENIX LISA'01, December 2001.

[11] British Broadcasting Corporation. Net surge for news sites. http://news.bbc.co.uk/hi/english/sci/tech/newsid_1538000/1538149.stm, September 2001.

[12] Lemos, R., Web worm targets White House, CNET News.com. http://news.com.com/2100-1001-270272.html, July 2001.

[13] Lemos, R., Antivirus firms team to thwart DoS attacks, CNET News.com. http://news.cnet.com/news/0-1003-200-6931389.html, August 2001.

[14] Sun Microsystems Inc. Java 2 Enterprise Edition. http://java.sun.com/j2ee.

[15] The PHP Group. PHP: Hypertext Preprocessor. http://www.php.net.

[16] Google. http://www.google.com.

[17] CNN. 20 million Americans see Starr's report on Internet. http://www.cnn.com/ TECH/computing/9809/13/internet.starr/, September 1998.

[18] Borland, J., Net video not yet ready for prime time, CNET news.com. http://news.cnet.com/news/0-1004-200-338361.html, February 1999.

[19] Fox, A., Gribble, S. D., Chawathe, Brewer, Y., E. A., and Gauthier, P., Cluster-based scalable network services. In Proceedings of the 16th ACM Symposium on Operating Systems Principles, St.-Malo, France, October 1997.

[20] Akamai, Inc. http://www.akamai.com/.

[21] Digital Island, Inc. http://www.digitalisland.com/.

[22] Gnutella. http://gnutella.wego.com.

[23] Sun Microsystems Inc. Java Servlet API. http://java.sun.com/products/ servlet/index.html.

[24] Dertouzos, M. L., The future of computing. Scientific American, July 1999.

[25] Gribble, S., Welsh, M., Behren, R. V., Brewer, E., Culler, D., Borisov, N., Czerwinski, S., Gummadi, R., Hill, J., Joseph, A., Katz, R., Mao, Z., Ross, S., and Zhao, B., The Ninja architecture for robust Internet-scale systems and services. Computer Networks, June 2000. Special Issue on Pervasive Computing.

[26] Hewlett-Packard Corporation. e-speak Open Services Platform. http://www.e-speak.net/.

[27] Sun Microsystems Inc. Enterprise Java Beans Technology, http://java.sun.com/products/ejb/.

[28] Sun Microsystems Inc. Jini Connection Technology, http://www.sun.com/jini/.

[29] Goddard, S., Schroeder, Trevor., The SASHA Architecture for Network-Clustered Web Servers, In proceedings of the 6th IEEE International Symposium on High Assurance Systems Engineering, 2001.

[30] Schroeder, T., Goddard, S., and Ramamurthy, B., Scalable web server clustering technologies, IEEE Network, May/June 2000.

[31] Srisuresh, P., and Load, D., Sharing using IP Network Address Translation (RFC 2391), August 1998.

[32] Cisco Systems, http://www.cisco.com.

[33] Nortel Networks, http://www.nortelnetworks.com.

[34] Foundry Networks. http://www.foundrynet.com.

[35] Brisco, T., DNS Support for Load Balancing, RFC 1794. April 1995,

http://www.rfc-editor.org/rfc/rfc1794.txt.

[36] Shaikh, A., Tewari, R., and Agrawal, M., On the Effectiveness of DNS-based Server Selection, IBM Research Report, Proceedings of IEEE Infocom, April 2001.

[37] Vivek S. Pai et. al., Locality-Aware Request Distribution in Cluster-based Network Servers, Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, October 1998.

[38] Zhang, W., Linux Virtual Server For Scalable Network Services,` http://www.LinuxVirtualServer.org.

[39] Cherkasova, L., and Karlsson, M., Scalable Web Server Cluster Design with Workload-Aware Request Distribution Strategy WARD, Computer Systems and Technology Laboratory, HP Laboratories Palo Alto, July 2001.

[40] Cecchet, E., Parallel Pull-Based LRU: a Request Distribution Algorithm for Clustered Web Caches using a DSM for Memory Mapped Networks. Third International Workshop on Software Distributed Shared Memory (WSDSM'01) in Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'2001) Brisbane, Australia - May15-18, 2001.

[41] Chaffee, A., http://www.jguru.com/faq/view.jsp?EID=125072