ROUTING ALGORITHM FOR DISTRIBUTED WEB DOCUMENTS

By

MADHURIMA PAWAR

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

To my parents

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Sanguthevar Rajasekaran for giving me an opportunity to work on this interesting topic and for providing me with excellent advice, great guidance and support throughout the course of this research work. His tireless patience and knowledge have been invaluable. Without his guidance, this thesis would not have been possible.

I am grateful to Dr. Douglas Dankel and Dr. Michael Frank, for graciously agreeing to serve on my committee and for taking time from their busy schedules to read and comment on my thesis.

On a more personal note, I would like to thank my family whose love, support and constant encouragement were of great importance through this work.

TABLE OF CONTENTS

page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Abstract of thesis Presentation to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master Of Science

ROUTING ALGORITHM FOR DISTRIBUTED WEB DOCUMENTS

By

Madhurima Pawar

August, 2001

Chairman: Dr Sanguthevar Rajasekaran
Major Department: Computer and Information Science and Engineering

When we talk of documents on the World Wide Web, for a particular site, the entire document is saved on the site. Now if the site is to be globally distributed at various places on the Internet, the document is replicated at each of the sites. The purpose of having multiples sites is so that, depending on the geographical location of the client, the request can be directed to the site closest to it. This is the Distributed Homogenous System.

In this research, we present our approach for building a reliable, large-scale, heterogeneous system, in which the document is distributed at the various sites. Essentially, we combine mutual failure detection with group membership, and package them together into a group communication system on top of which reliable and scalable distributed services can be built.

To provide reliable services, the communication between sites depends on failure information reported by a local mutual failure detector as well as the defined properties of

the group communication system itself. We consider the problem of choosing among a collection of servers, the owner of the document, and the ones having the replicated copy of the document, focusing on the question of how to make choices that segregate client/server traffic according to network topology. We uncover a number of tradeoffs between effectiveness, network cost, and ease of deployment.

CHAPTER 1
INTRODUCTION

1.1 Problems and Motivation

The phenomenal growth of the World Wide Web is imposing considerable strain on the Internet resources and the Web Servers, prompting numerous concerns about the Web's continued viability. The success of the high performance Web servers in alleviating these performance problems is ultimately limited, unless Web services are designed to be inherently scalable. To construct scalable Web servers, system builders are increasingly turning to distributed designs [1,15,32].

As a network grows, popular network services often suffer degraded availability and response times [7,9,14,21]. This problem is painfully evident in the Internet, where individual servers and network links are often swamped with tremendous, global service demands [28,33]. The solution, of course up till now, is caching [6,9,17] and replication [8,12,18]. To solve this problem we focus on a particular aspect: given a replicated set of servers, how can we distribute service requests in a fashion that minimizes traffic across high-level parts of an Internet, such as regional and long haul links.

The problem is easy for extreme degrees of replication/distribution [5,6]. If there are only a small number of replicas, one can use crude approximations such as mapping Domain names to broad geographic regions, each containing one replica. At the opposite extreme, if a service is replicated into every subnet one can use broadcast to locate nearby server replicas.

The increasing popularity of the distributed information services [4,10,25,30] like the WWW has resulted in a number of problems of scale. Three scaling impediments for the distributed information services are excessive server load due to popular documents, wasted network bandwidth due to redundant document transfer and excessive latency in delivering documents to the client due to the potential from the transfers over the slow path. A technique that promises to address all of these problems is document distribution amongst various servers. However, when a document is distributed with partial replication, client face the additional task of finding the best provider of that service.

In many cases, clients know in advance which service providers are best for them. However, static server selection is inappropriate in a number of cases, like in our case in distributing WWW documents. In this context, dynamic server selection has the potential to reduce the elapsed time between document request and the document arrival, which we will call the response time. In addition, as we will show, a dynamic server selection mechanism enables a very flexible, simple policy for the document distribution and partial selection.

1.2 Our Approach

In this thesis we provide tools and techniques for selecting good service providers without assuming knowledge of server location or network topology.

Instead of following the policies of replicating the documents at each site, break the document into various pages. These pages of the document are saved at each of the sites, with that site being the owner for that particular document page.

The owner site has the right to make updates and any changes whatsoever required on that page. No other site can alter the contents of that page of which it is not an owner.

The client request to a particular site is first analyzed. If the request is for the page owned by that site, the request is catered, else it is directed to the owner site of that page.

This could result in a lot of traffic overload, by continuously redirecting the request, to the owner site. In cases, when the request for a particular page, is very frequent, the site, could request the owner site for a cache copy of the page, and retain the copy as long as the requests for that page keep coming or until the page is altered by the owner site.

Here we see, that the problem is not just successfully directing the request of the client to the desired site, but also keeping a consistency among all the databases in all the servers at the various sites. The problem of cache consistency also arises, in case of replicated copies of the page at the various servers. We uncover a number of tradeoffs between effectiveness, network cost, ease of deployment, and portability across different types of networks [10,24,25].

1.3 Contributions

The key contribution of this Master research are

- Definition of an efficient architecture and an asynchronous model for large scale collaborative distributed services over large network such as the Internet.
- Specification with a correctness proof and construction of a combined, efficient process and communication failure detector, which is suitable for

large-scale distributed systems experiencing frequent communication failures. This mutual failure detector is the foundation of our weak group communication system.

- Specification with a correctness proof and construction of a weak group communication system, namely group membership services that are suitable for distributed services with service reliability, availability and scalability goals.

1.4 Thesis Organization

We review previous works in Distributed Web Servers, with either document caching or replication scheme. Then we establish the problem requirements of large scale distributed services and also discuss the various scenarios in which such systems perform better than the previous systems in use.

We also list all the possible configurations possible, giving detail description of each environment. It is beautifully illustrated with hands-on examples, from real life, giving a vivid picture of how the system would behave under different situations.

Before carrying out the experiment, some initial setup has been taken into consideration, which is also discussed. Then we have vividly described the programming strategy to implement the Distributed Web Server Scheme.

The concept being new and innovation, there are various aspects of each scenario. However, towards the end, we have listed the aspects that need some more research work and are not being covered in this thesis report.

CHAPTER 2
BASICS AND PREVIOUS WORK

2.1 WWW, Client and Server

The World Wide Web is an architectural framework for accessing linked documents spread out over thousands of machines all over the Internet.

The Client/Server model is a programming paradigm that represents the interactions between processes and structures of the system. Any process in a system either provides services for others or requests services from them. Processes that request services are clients and those that provide services are servers. In any interaction a process is either a client or a server. In many cases, the same process plays the roles of both.

A client and a server interact through a sequence of requests and responses as shown in Figure 2.1. A client requests a service from the server and blocks itself. The server receives the request, performs the operation, and returns a reply message to the client. The client then resumes its execution. The only underlying assumption is a synchronous request/reply exchange of information. Logically, the clients communicate directly with the servers.

A client/server model can also be considered as a service-oriented communication model. It is a higher-level abstraction of inter-process communication, which in turn is implemented by either RPC or message passing communication, which in turn is

implemented by either a connection-oriented or connectionless transport service in the network. Services provided by a server, may be connection-oriented or connectionless.

Multiple servers can coexist to provide a similar service. They need to be identified either by their names or the functions that they perform. This requires a server that locate servers. These servers, called agent servers, bind the client process and the selected server process together, and they sometimes may also need to be located. At the end, there should be minimum number of servers with well-known names or addresses. Upon a request from a client, the binder server can choose a server suitable for the client or the one that best balances the load of the servers.

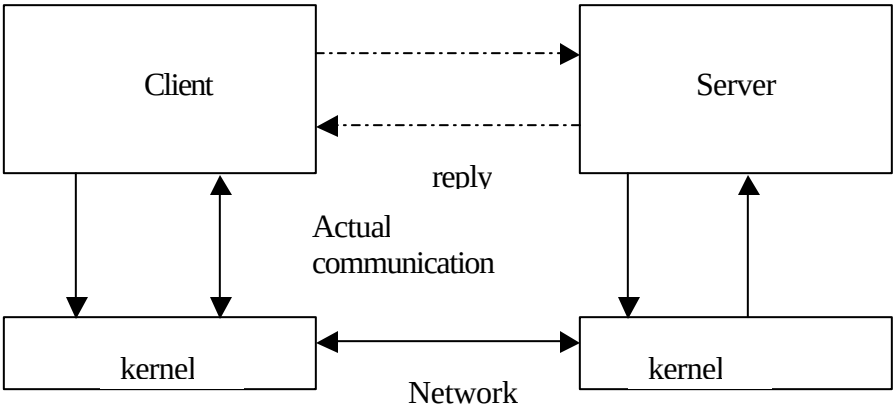

Figure 2.1 Schematic Representation of Client Server Architecture

2.2 Distributed Web Servers

Corporations and Internet service providers (ISPs) are starting to alter the design of their World Wide Web servers. Traditionally, companies placed servers in one central location, which all users would access. With the number of hits reaching into the millions

in a growing number of cases, limitations with this design are becoming evident. Server hardware is having difficulty keeping pace with growing processing requirements, and more and more network bandwidth is needed to move data between servers and end users.

Because of these problems, corporations and ISPs are starting to look for ways to distribute computer and networking chores: Load-balancing products enable a company to divide processing chores among multiple servers; intelligent network routers are directing users to the closest Web servers rather than letting them merrily traverse the Internet; and caching products are pushing commonly used files and images out from central sites closer to end users.

These products are new and come with the kinds of flaws associated with emerging technology. Deploying these products can be difficult because final testing may take place at a customer's site rather than in a vendor's laboratory. A company may need systems integration skills to put all of the necessary pieces in place. Managing distributed connections is difficult, and the tools needed for monitoring Web servers may not yet be available.

A new design is needed for several reasons. The amount of information stored on Web servers and the number of persons accessing them continues to grow at unprecedented rates. Installed computer and networking resources are straining to offer users adequate response times. New technologies are making it possible for users to work with more sophisticated graphics and video applications. These applications require more computer power to process and more bandwidth to move data over the Internet.

2.3 Message-Passing Systems

A message passing distributed system consists of a number of process and communication channels. Processes, in a message passing system, communicate with each other by explicitly exchanging messages over unreliable communication channels. These processes are concurrent, but they are usually cooperative and work collaboratively toward a common goal.

In real message passing systems, processes and communication channels are usually heterogeneous. Thus, they are always different in a number of characteristics. The differences of timing characteristics among these processes and communication channels have the most significant impact on the ability of the distributed system to make progress or to solve important problems. The timing characteristic of the system components are often referred to as system synchrony, which includes communication delays, local clock drift, and relative process speed. Communication delay probably has the most impact on the large-scale distributed services over wide-area networks.

Some system synchrony can easily be bounded under certain environments. When that happens, we have a synchronous system. This particular type of system is usually, a very small-distributed system consisting of mainly homogeneous components connected together by a very high-speed interconnection network.

It is difficult to build a message-passing synchronous system that always satisfies these timing properties. This is due to the fact that probabilistically, system components may fail, or system transient load may vary unpredictably. These failures, transient or permanent, lead to asynchrony. Thus, for a large-scale, wide area distributed system, which is an interconnection of a large number of heterogeneous components across a large geographic distance, it is unrealistic to assume that the bounds exist for system

synchrony. A weaker model that is suitable for large, heterogeneous distributed systems is the asynchronous model.

The message passing asynchronous model assumes less about timing properties compared to the synchronous model. Since the asynchronous model assumes very little about the timing information of events that are usually provided by the real distributed systems, algorithms or protocols designed for the asynchronous system are more general and more portable. They should run correctly in network with arbitrary timing guarantees. One obvious consequence is that the protocols do not utilize the system synchrony. This could severely hurt the performance of the systems.

In practice, even an asynchronous system may indeed satisfy synchrony properties temporarily. Also, most distributed protocols do not require that the system be synchronized indefinitely for these protocols to terminate. More often that not, distributed protocols only require that some synchrony exists for only a short period of time, in particular during a certain phase of the protocols. That is sufficient for the protocols to terminate and the system can make progress toward the termination of the protocols.

2.4 Remote Procedure Call

Remote Procedure Call (RPC) is a client/server infrastructure that increases the interoperability, portability, and flexibility of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the complexity of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces--function calls are the programmer's interface when using RPC.

To access the remote server portion of an application, special function calls, RPCs, are embedded within the client portion of the client/server application program. Because they are embedded, RPCs do not stand alone as a discreet middleware layer. When the client program is compiled, the compiler creates a local stub for the client portion and another stub for the server portion of the application. These stubs are invoked when the application requires a remote function and typically support synchronous calls between clients and servers.

RPC is appropriate for client/server applications in which the client can issue a request and wait for the server's response before continuing its own processing. Because most RPC implementations do not support peer-to-peer, or asynchronous, client/server interaction, RPC is not well suited for applications involving distributed objects or object-oriented programming

Asynchronous and synchronous mechanisms each have strengths and weaknesses that should be considered when designing any specific application. In contrast to asynchronous mechanisms employed by Message-Oriented Middleware, the use of a synchronous request-reply mechanism in RPC requires that the client and server are always available and functioning (i.e., the client or server is not blocked). To allow a client/server application to recover from a blocked condition, an implementation of a RPC is required to provide mechanisms such as error messages, request timers, retransmissions, or redirection to an alternate server. The complexity of the application using a RPC is dependent on the sophistication of the specific RPC implementation (i.e., the more sophisticated the recovery mechanisms supported by RPC, the less complex the

application utilizing the RPC is required to be). RPCs that implement asynchronous mechanisms are very few and are difficult to implement.

When utilizing RPC over a distributed network, the performance (or load) of the network should be considered. One of the strengths of RPC is that the synchronous, blocking mechanism of RPC guards against overloading a network, unlike the asynchronous mechanism of Message-Oriented Middleware (MOM). However, when recovery mechanisms, such as retransmissions, are employed by an RPC application, the resulting load on a network may increase, making the application inappropriate for a congested network. Also, because RPC uses static routing tables established at compile-time, the ability to perform load balancing across a network is difficult and should be considered when designing an RPC-based application.

2.5 Distributed Computing Environment

Developed and maintained by the Open Systems Foundation (OSF), the Distributed Computing Environment (DCE) is an integrated distributed environment, which incorporates technology from industry. The DCE is a set of integrated system services that provide an interoperable and flexible distributed environment with the primary goal of solving interoperability problems in heterogeneous, networked environments as shown in Figure 2.2.

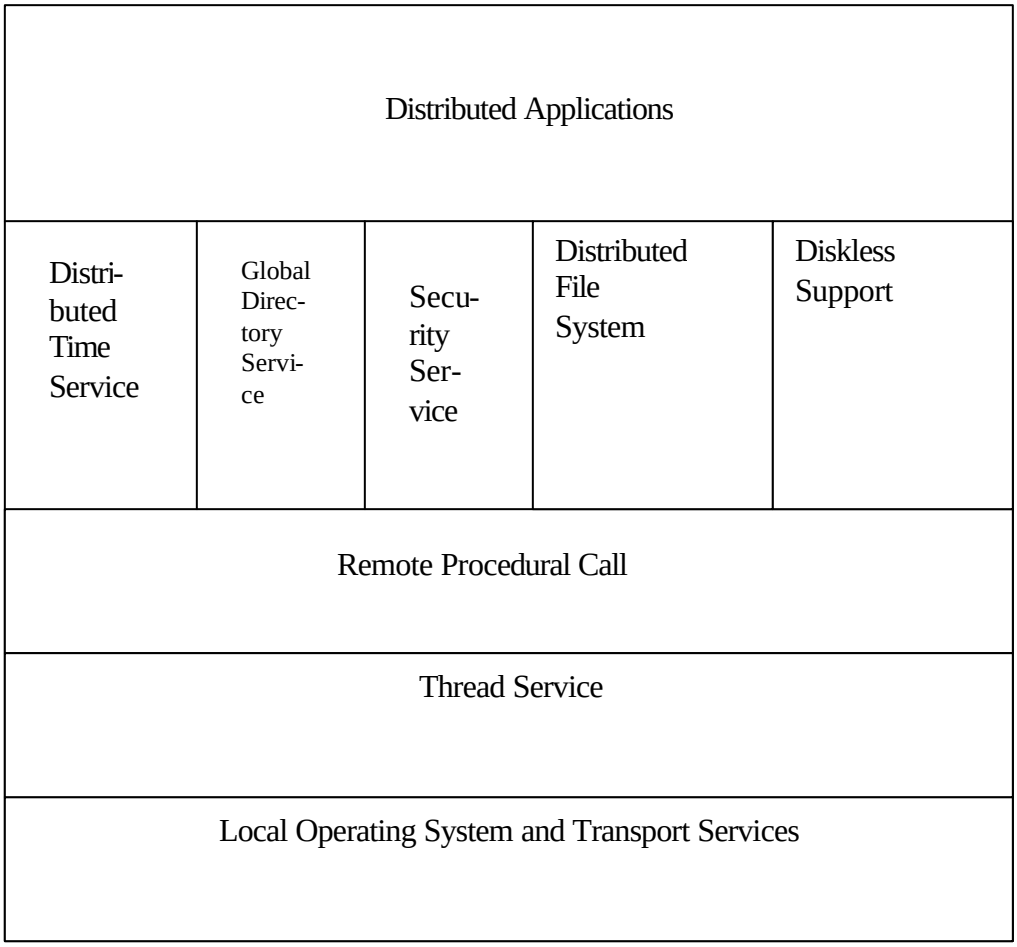

Figure 2.2 Distributed Computing Environment

DCE services are organized into two categories

1. Fundamental distributed services provide tools for software developers to create the end-user services needed for distributed computing. They include
   - Remote Procedure Call, which provides portability, network independence, and secure distributed applications.
   - Directory services, which provide full X.500 support and a single naming model to allow programmers and maintainers to identify and access distributed resources more easily.

- Time service, which provides a mechanism to monitor and track clocks in a distributed environment and accurate time stamps to reduce the load on system administrator.
- Security service, which provides the network with authentication, authorization, and user account management services to maintain the integrity, privacy, and authenticity of the distributed system.
- Thread service, which provides a simple, portable, programming model for building concurrent applications.

2. Data-sharing services provide end users with capabilities built upon the fundamental distributed services. These services require no programming on the part of the end user and facilitate better use of information. They include:
   - Distributed file system, which interoperates with the network file system to provide a high-performance, scalable, and secure file access system.
   - Diskless support, which allows low-cost workstations to use disks on servers, possibly reducing the need/cost for local disks, and provides performance enhancements to reduce network overhead.

DCE works internally with the client/server model and is well suited for the development of applications that are structured according to this model. Most DCE services are especially optimized for a structuring of distributed computing systems into a "cell" (a set of nodes/platforms) that is managed together by one authority.

2.6 Routing

The router [3] in an Internet performs much the same function as a packet-switching nodes (PSN) in a packet-switching network. Just as the PSN is responsible for

receiving and forwarding packets through a packet-switching network, the router is responsible for receiving and forwarding IP data grams through an Internet. For this purpose, the routers of an Internet need to make routing decisions based on knowledge of the topology and conditions of the Internet [29,30].

Routing involves two basic activities: determination of optimal routing paths and transporting information groups through an inter-network [16]. Routers communicate with each other and maintain their routing tables through the transmission of a variety of messages [13]. The routing update message is one such message that generally consists of all or a portion of a routing table. By analyzing routing updates from all other router, a router can build a detailed picture of network topology [19].

Routing algorithms [2] have used many different metrics to determine the best route. Sophisticated routing algorithm can base route selection on multiple metrics, combining them in a single metric. All the following metrics have been used: Path length, Reliability, Delay, Bandwidth, Load, Communication cost.

2.7 Shortest Path Determination

Communication networks are represented in a natural way by graphs, in which the server/routers are the vertices of the graph and the communication lines are the edge of the graph. A number of concepts from graph theory are useful in the design of networks and in the development of routing algorithms.

A weighted graph is one in which a number is associated with each edge. The distance between two vertices is equal to minimum of edges in any path connecting them.

The distance between any two nodes can be calculated by counting the number of hops (edges) between them, considering that each edge has value of one. The minimum

path distance can be calculated by counting the minimum number of hops. One such algorithm is the Bellman-Ford algorithm.

The algorithm can be stated as follows: find the shortest paths from a given source vertex subject to the constraint that the paths contain at most one link, then find the shortest paths with a constraint of paths of at most two links and so on. This algorithm also proceeds in stages:

Let distl [u] be the length of a shortest path from the source vertex v to vertex u under the constraint that the shortest path contains at most l edges. Then, distl[u] = cost[u][v], 1 <= u <= n. As noted earlier, when there are no cycles of negative length, we can limit our search for the shortest paths to paths with at most n-1 edges. Hence, distn-1[u] is the length of an unrestricted shortest path from v to u.

Our goal is to compute distn-1[u] for all u. This can be done using the dynamic programming methodology. First, we make the following observations:

- If the shortest path from v to u with at most k, k > 1, edges has no more than k-1 edges, then distk[u] = distk-1[u].
- If the shortest path from v to u with at most k, k > 1, edges has exactly k edges, then it is composed of a shortest path from v to some vertex j followed by the edge <j,u>. The path from v to j has k-1 edges, and its length is distk-1[j]. All vertices i such that the edge <i,u> is in the graph are candidates for j. Since we are interested in a shortest path, the i that minimizes distk-1[i] + cost[i][u] is the correct value for j.

These observations result in the following recurrence for the dist:

Dist k[u] = min{ fdistk-1[u], mini { distk-1[i] + cost[i][u]} }

This recurrence can be used to compute dist k from distk-1, for k = 2,3,…..,n-1.

2.8 One-copy Serializablity

Data objects and files are often replicated to increase system performance and availability. With replication, higher performance is achieved by allowing concurrent access to the replicas, and higher availability can be obtained by exploiting the redundancy of the data objects. These are parallelism and failure transparencies, which are desirable in a distributed system. However, they are useless unless we can also provide replication and concurrency transparencies. Replication transparency means that clients are not aware of the existence of replicas. A condition of concurrency transparency is that interference among sharing clients must be avoided.

We assume the generic system architecture shown in figure 2.3, for management of replicated data. Clients may choose one or more file service agents (FSA) for accessing data objects. File service agents serve as a front end to the replica managers (RM) to provide replication transparency for the clients. As FSA may contact one or more RMs' for the actual reading and updating of data objects. Depending on how replica management protocols are to be implemented, the responsibility of replica management may be divided between the FSA and the RM. The architecture is a client/server model, which is different from the peer architecture.

If both read/write operations must be directed to the primary replica manager, the issue of replication does not exist. The primary RM can serialize all operations. The secondary RM only supplies redundancy in case of primary failures. Consistency is easy

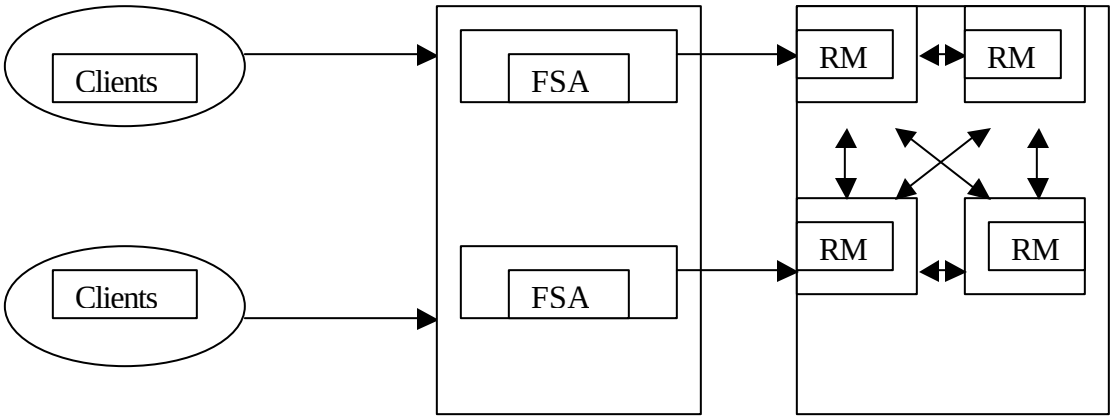

Figure 2.3 One Copy Serializability

to achieve, but concurrency is completely ignored. To provide concurrency, read operations should be performed at any RM site. However, this introduces the coherency problem since the propagation of updates from primary RM to secondary RM, preferably one that is closer to the requesting client. This is the read-one/write-all strategy. The consistency of the replicas can be enforced by using a standard concurrency control protocol such as the two-phase locking or the two-phase timestamp ordering protocol. If the read/write operations are sub-transactions, the read-one/write-all protocols achieve one-copy serializability, the execution of transactions on replicated objects is equivalent to the execution of the same transaction on non-replicated objects.

The assumption of write-all is not realistic for replica management. Data objects are replicated to tolerate failures. Requiring all replicas to participate in the atomic update contradicts the purpose of replication. Therefore, the atomic update should be performed only on all of the available replicas. A read-one/write-all-available protocol seems more appropriate.

CHAPTER 3
INITIAL CONFIGURATION

We have the web servers [31] distributed geographically at various locations [23,11,32]. The document is either replicated at each of the site, or is distributed uniformly, giving rise to two scenarios, which are handled separately in this thesis.

Before we discuss the initial set up in each case, it is important to note that the servers, not only act as web servers, catering to client request, but also act as routers. In cases when they cannot service the client request, they appropriately direct it to the site that can [20].

For this purpose the sites, along with the document, have a router table, which indicates all the sites connected to it, and their distances. In our case, we randomly generate a number of sites, and construct routing table, which is replicated at each site. This is the static approach, which overlooks the current dynamic state of the network. However, the changes can be made to the routing table, once the set up is initialized, according to the network conditions.

The first scenario is the one in which we have the document replicated at each of the sites. For our thesis, we have statically replicated the whole of document at each site, before the site is operational.

In the second scenario, where the document, is distributed to various sites, the distribution is done at the beginning. The document is spilt randomly into the number of servers, and the respective document parts/pages are copied to the respective servers. The Document table is initialized accordingly and copied to all the sites. This is the static

method of initialization. However, once the sites are up and functional, the Document table, like the routing table, is dynamically updated to take care of the network changes.

Hence we observe that for our experimentation, the initial set up is standardized. However, it assumes the realistic approach after the servers become functional.

The client requests are generated after a random amount of time, or can also be user initiated, in which the user decides the site to which to make request and the part of the document it requires. However, the randomly generated requests help in determining and comparing the performance of both the scenarios [26].

CHAPTER 4
HOMOGENOUS ENVIRONMENT

4.1 Definition

We have a scenario, in which all the web servers are geographically distributed. In a homogenous environment, the entire document is replicated at each of the web servers. Hence there is consistency in the all the web sites, and if any changes to the document are made, they are to be replicated at all the sites.

Note here we interchangeably use the terms web server and site.

4.2 Table Update

Each of the sites also acts as a router, and hence it has a routing table. This table, as mentioned earlier in the chapter, maintains information about the distance of all the other sites that are directly connected to it.

In our case, this is determined statically, at the creation time only.

When a particular server needs to forward a client request to a particular site, it first calculates the shortest distance using the Bellman and Ford algorithm as explained in the previous chapter. Then, it forwards the request to that site.

Depending on the network condition, the tables are updated. This is done by pinging all its neighbors regularly to check for their alive status. If a particular site goes down, then the changes are made the routing table, indicating that link to have an infinite distance i.e. unreachable state.

We have the whole of the document replicated at each site, to increase performance and availability. With replication, higher performance is achieved by allowing concurrent access to replicas, and higher availability can be obtained by exploiting the redundancy of the data objects. These are the parallelism and failure transparencies, which are desirable in a distributed system. Thus, any updates to the document at one site should be uniformly and transparently distributed to the other sites, to maintain consistency. As explained in the earlier chapter, we use the One-Copy Serializability protocol to achieve the consistency amongst the distributed documents.

4.3 Serving Request

Now consider a situation in which a client makes a request to any of the servers. As the entire document is replicated at each site, the request is immediately taken care of.

A server at some time can service only a threshold number of requests. If the number of request it receives at any point of time exceeds the threshold value, then the network is said to be overloaded and the performance decreases. To avoid excessive load on a particular site, this site, then calculates the site closes to it. The calculation is done on the basis of hop count, traffic congestion, and bandwidth availability.

The request is then forwarded to the other site. In situation, when the site closest is already overloaded, it is forwarded to the next closest site. This search is continued till; a particular site finally services it.

4.4 A Scenario

Consider the following scenario, in which the network comprises of following 5 servers, connected as shown in the Figure 4.1.

Initial Configuration: We have a document D, which is replicated on each server. The maximum number of requests a server can handle in order for precise and timely delivery is n request per server.

CaseI: We have a client request to server 3 for some part of the document. As the document is available at every server, and the server is below its threshold value, the request is immediately served.

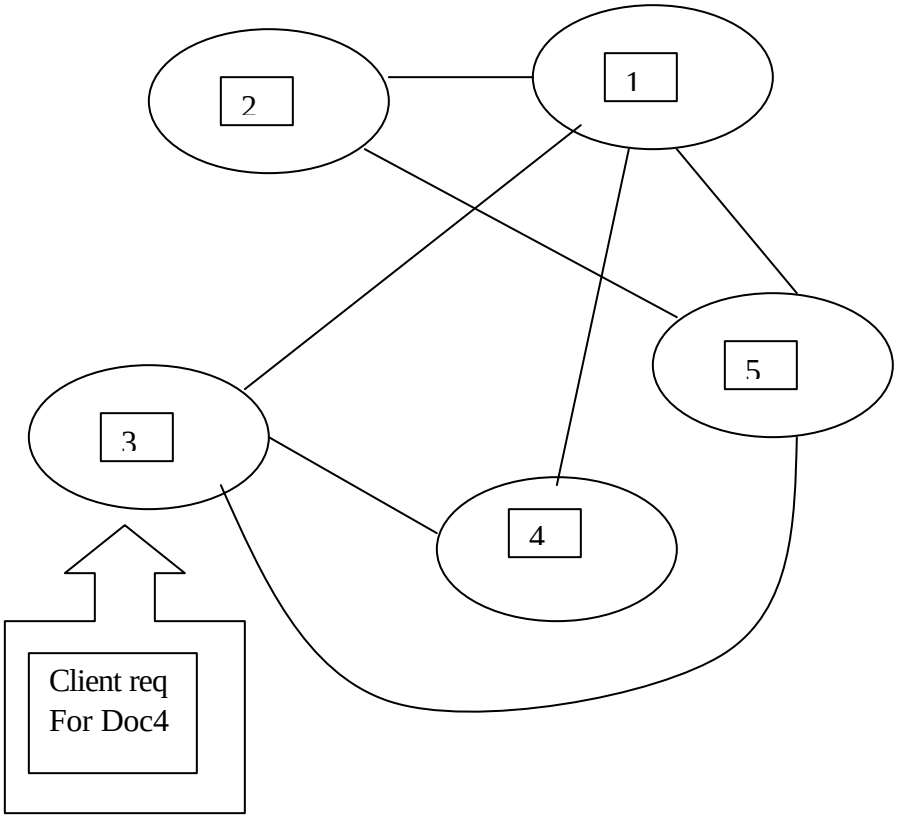

Figure 4.1 Scenario of Homogenous System

Table 4.1 Document Table for a Homogenous System

	Doc 1	Doc2	Doc 3	Doc 4	Doc 5
Site 1	1	1	1	1	1
Site 2	1	1	1	1	1
Site 3	1	1	1	1	1
Site 4	1	1	1	1	1
Site 5	1	1	1	1	1

Table 4.2 Routing Table for Homogenous System

	Site1	**Site 2**	**Site 3**	**Site 4**	**Site 5**
Site 1	-	6	2	7	2
Site 2	6	-	-	-	8
Site 3	2	-	-	5	3
Site 4	7	-	5	-	-
Site 5	2	8	3	-	-

Case II: Now consider the situation when server 3 already has n request running, and another client request serves. As it cannot process any further request, this request is directed to the server closest to it, that is running below its limit. The server closest is calculated using the dynamic state of the network, calculated at that instant time by the routing table, maintained at each server. Once a neighboring server is found, probably after several attempts, the server now acts as a router. It redirects the client's request to the next server. The server, to whom the request is now forwarded, caters to the request.

4.5 Drawback

Thus we observe, that no matter to which server the request arrives, it is immediately processed, or can be transferred to the neighboring site, in case of an overloaded network. When a request arrives for a particular document, it is cached for speedy delivery. In case the document is not in the cache, a miss occurs, and it is replaces by the document part in demand. Now there is a limit to cached documents, and in case of heavy traffic condition, each client requesting for different parts of the document, the number of misses increases. This reduces the performance of the server. Resulting in longer service time.

Also as the document is replicated at each site, the amount of memory required is enormous. Any updates, or changes with the document involve propagating those changes to all the sites, without considerable delay, so that when a client requests for the document, to any of the servers, only the updated version is provided. The time to propagating updates to all the servers depends on several factors such as the bandwidth availability and traffic conditions. Only the changes are propagated around the network, or in some extreme cases, the entire document could be transferred. The former is more efficient and is preferred to later method, as the chances of increasing the network traffic is less, and also the changes can be made quickly.

To overcome all these problems, and increase the efficiency under increase load conditions, we have the documents distributed at each site, as explained in the next chapter.

CHAPTER 5
HETEROGENOUS ENVIRONMENT

5.1 Definition

This is the environment in which instead of having all the pages of document replicated at each site, the pages are distributed to the various sites[1]. No site can have a copy of the same page of the document. The site, which has a page of a document, is the owner site of that page.

So we have the scenario in which each site is the owner of page of the document. Only the owner site has the right of making any updates/changes to the page. If any other site, besides the owner, performs the change, it is ineffective and illegal.

The distribution of the pages to the various sites could be done statically or dynamically. In static distribution, the pages are evenly distributed to all the sites. The site, which has a page, becomes the owner and remains the owner of the page, as long as it is alive. In dynamic distribution, the servers initially distributed the pages evenly, but in the event of time, depending on the number of request, it can change its ownership. This enables a better service request time to the clients who keep making frequent requests for the same page to the same site. However, this aspect is not handled in this thesis, as it

[1] Here when we state document, we refer to the entire web sites. The document being broken up into various pages, indicates different web pages for that particular site. Hence heterogeneous system is the one in which instead of having all the web pages of a particular web site on one server, like in a homogeneous environment, and replicated to all the servers, the web pages are distributed to the various servers, geographically distributed to different locations.

tends to get a little more complicated with time, and requires a much more understanding of the whole framework.

The disadvantage is that if any site fails, and a request comes for a page owned by that site, it can never be serviced. To avoid this situation, there is one Master site, which has the whole of the document. This site is a little different than other sites. It caters to requests only if a particular master site fails to respond. The other sites forward requests to Master site, only upon acknowledging the inability of the owner site to cater to the request and if no cache copies at other sites are available.

The partial caching mechanism is employed to reduce the service request time, by avoiding the delay caused due to the transfer of request from one site to another. The scheme can be used only when the number of requests arriving at a particular site is large and the overhead involved in redirecting the request to the owner site is large. Hence, the site with a request asks for a copy from the owner site, and it caches this copy. Now it can cater to any request for this page, as long as no changes are made by the master site, and this cache becomes null and void. Hence there is a considerable reduction in the overhead involved by transferring the request each time to the owner sites.

5.2 Table Update

Each of the site also acts as router, and hence it has a routing table. This table, as mentioned earlier in the chapter, maintains information about the distance of all the other sites that are directly connected to it. In our case, this is determined statically, at the creation time only.

When a particular server needs to forward a client request to a particular site, it first calculates the shortest distance using Bellman and Ford algorithm as explained in the previous chapter. Then it forwards the request to that site.

Depending on the network condition, the tables are updated. This is done by pinging all its neighbors regularly to check for their alive status. If a particular site goes down, then the changes are made to the routing table, indicating that the link has an infinite distance i.e. unreachable state.

Each site also maintains a Document Table. This has the listing of all the sites and the pages of which they are the owner. Hence whenever a request for a particular page arrives, it first checks the owner site, and then redirects the request to that site. As the site also can cache the page, which they do not own, it becomes all the more important for each site to keep track of all the available copies of that document. Hence to speed up the service time, not only can it redirect the request to the master site, but also to the site which have a cache copy of that page, and are closer than the master site.

Now as mentioned earlier, any changes made to the page, by the owner need not be propagated to all the sites. This helps in making frequent updates to the page, without having to take into account the overload required to propagate the changes to the other sites, as in the earlier chapter. But it also becomes important, to propagate the changes to sites which have a cache copy of the document. This is achieved by first sending a signal to all the sites that have the cache copies, to nullify their cached document. After all the responses are received from all the servers, the master site then proceeds to make any changes to the document. During this update, if any request arrives either from a client or

another site for a cache copy, it is either put into a waiting queue, or is denied. After the update is completed, the requests are processed the waiting queue.

All the sites that had the copy have now nullified their cached documents, after receiving a nullify request from the Master site. If it wishes to again access the page, it makes a new request to the owner, all from the start, and gains a new updated copy of the page.

5.3 Serving Request

Now consider a situation in which a client makes a request to any of servers, for a part of the document. There could be two situations, either the site is the owner of the document or is not. Let us consider the two situations in detail:

Case I: Owner Site

If the request for a document arrives to the site that owns the page, it is immediately processed. This site in this case is not overloaded. If it is, then it checks its Document table, and for other sites that have a cached copy of the page. It then determines which site, which has a cache copy and is the closest, and redirects the request to that site. In case no site has a cache copy exists, then the request is denied.

Case II: Requesting the owner

If the request for a document arrives to the site that does not own the page, the site first checks whether it has a cached copy of the page. If it does, then it can service the request.

If it does not have the cache copy, it increments the cache hit variable, which keeps track of the number of requests the site has received for the page. This will, as we will see later, help to determine when a site should request for a copy of the page from

the owner site. It then checks the Document table to locate the sites, having a cache copy of this page, as well as the master site, and then forwards to the site closest to it.

If there happen to be no cache copy elsewhere, the request is redirected to the master site. But before this, some checks are made. If the cache-hit variable is greater than some predefined threshold value, then the site instead of redirecting the request to the owner site, requests the owner site for the copy of the document. It updates the Document table, and the master propagates the changes to all the sites. Now this site services the request.

5.4 A Scenario

Consider the following scenario, in which the network consists of 5 servers, connected as shown in the graph below. The Routing Table and the Document Table are shown in Figure 5.1.

Case I: Request to the owner

The request arrives at site 3 for page 3, of which it is owner. If the site is not very heavily loaded, the site services the request. If the site is already overloaded, it searches the Document table for any cache copy. Here we have site 1 also having the cache copy, so it is redirected to that site.

Case II: Request to the site not the owner.

Consider now if a request appears on site 5 for page 3. As site 5 does not have the copy of the page, it first increments the cache hit variable. Then, it redirects the request to the site, which is either the owner, or has a copy of the page.

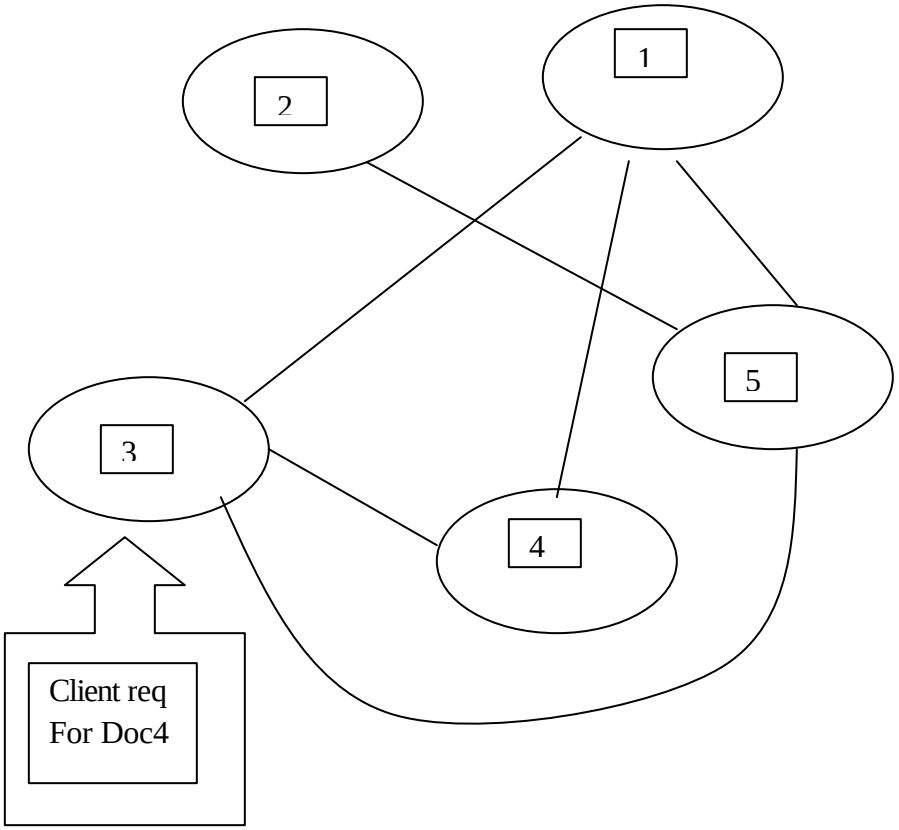

Figure 5.1 Heterogeneous Scenario

Table 5.1 Document Table for Heterogeneous System

	Doc 1	**Doc2**	**Doc 3**	**Doc 4**	**Doc 5**
Site 1	1	0	0	0	0
Site 2	0	1	0	1	0
Site 3	0	0	1	0	0
Site 4	0	0	0	1	0
Site 5	0	0	0	0	1

Table 5.2 Routing Table for Heterogeneous System

	Site1	**Site 2**	**Site 3**	**Site 4**	**Site 5**
Site 1	-	6	2	7	2
Site 2	6	-	-	-	8
Site 3	2	-	-	5	3
Site 4	7	-	5	-	-
Site 5	2	8	3	-	-

If however, the cache-hit variable reaches the threshold value, it indicates that site has been receiving many requests for the same page. It becomes expensive in such situations, to every time redirect the request, not only increasing the network traffic, but also increasing the service time delay. To overcome this drawback, the site sends a request to the master site, for a copy of the document. An update is made to all the Document Tables of various sites. Anytime in the future, this site also is capable of servicing request for the page of which it is not the owner.

When a site has a cache copy of the document, and is not requested for quite some time, it becomes essential to drop the cache copy. This is achieved by starting a timer, every time a request is completed and there are no pending requests for the page. If no requests arrive for quite some time, the cache copy is dropped and the updates are made to the document table.

This feature enables a site to have cache copies of various pages depending upon the frequency of requests it receives.

5.5 Advantage

The advantage of the distributing the pages to all the sites enables a considerable reduction in the memory requirement. Instead of having to save the entire document at each site, only a subset of the pages are saved at each site.

When the network is heavily loaded, this structure gives a very good performance. The traffic is not loaded by propagating changes made any part of the document to the entire site. Instead, only the Document Tables are updated at each site. This tremendously reduces the traffic and improves performance.

This structure also allows a site to have a cached copy of a page should the client request increases tremendously. This is a very important feature for the dynamically changing Internet, where the request keeps changing with time. Once again, this feature allows for faster delivery with changing demands.

As the entire document is distributed to the various sites, it helps in maintain the document with ease. The master site is responsible for maintaining and handling any changes and updates made to that a part of the document, instead of the entire document. This makes the work less complicated and more efficient. The load of the servers reduces, besides catering to the service request from the clients.

Hence we observe that there is a tremendous benefit in this configuration, instead of the traditional replication, as mentioned in the earlier chapter.

CHAPTER 6
IMPLEMENTATION

6.1 Java RMI Architecture

To implement the client-server architecture, we have used JAVA RMI [22]. Remote Method Invocation (RMI) technology elevates network programming to a higher plane. Although RMI is relatively easy to use, it is a remarkably powerful technology and exposes the average Java developer to an entirely new paradigm--the world of distributed object computing. The design goal for the RMI architecture was to create a Java distributed object model that integrates naturally into the Java programming language and the local object model. RMI architects have succeeded; creating a system that extends the safety and robustness of the Java architecture to the distributed computing world.

The RMI architecture is based on one important principle: the definition of behavior and the implementation of that behavior are separate concepts. RMI allows the code that defines the behavior and the code that implements the behavior to remain separate and to run on separate JVMs.

This fits nicely with the needs of a distributed system where clients are concerned about the definition of a service and servers are focused on providing the service.

Specifically, in RMI, the definition of a remote service is coded using a Java interface. The implementation of the remote service is coded in a class. Therefore, the key to understanding RMI is to remember that interfaces define behavior and classes define implementation.

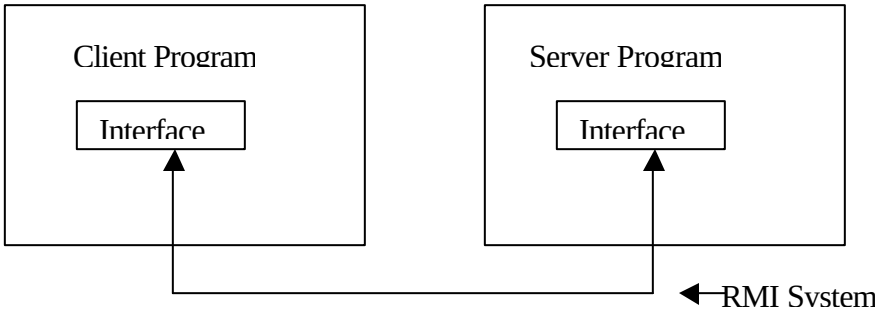

Figure 6.1 Remote Method Invocation

While the following diagram illustrates this separation, remember that a Java interface does not contain executable code. RMI supports two classes that implement the same interface. The first class is the implementation of the behavior, and it runs on the server. The second class acts as a proxy for the remote service and it runs on the client. This is shown in the Figure 6.1.

A client program makes method calls on the proxy object, RMI sends the request to the remote JVM and forwards it to the implementation. Any return values provided by the implementation are sent back to the proxy and then to the client's program.

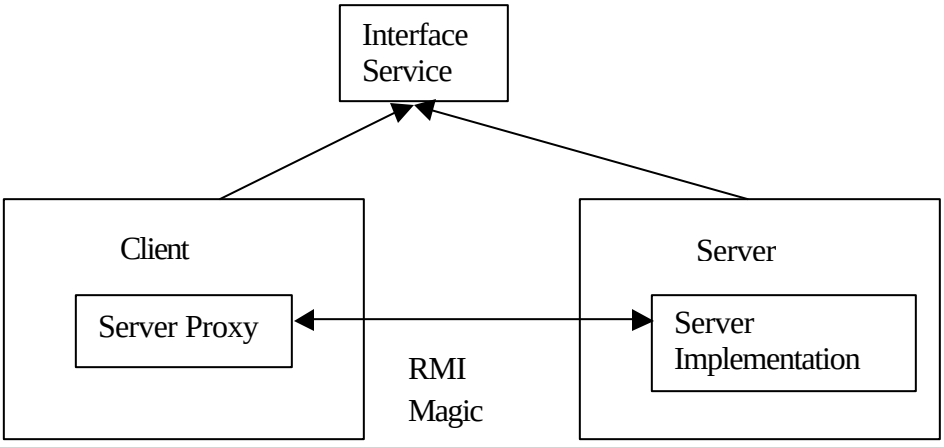

Figure 6.2 Client / Server in RMI

The RMI implementation is essentially built from three abstraction layers, as shown in Figure 6.3. The first is the Stub and Skeleton layer, which lies just beneath the view of the developer. This layer intercepts method calls made by the client to the interface reference variable and redirects these calls to a remote RMI service.

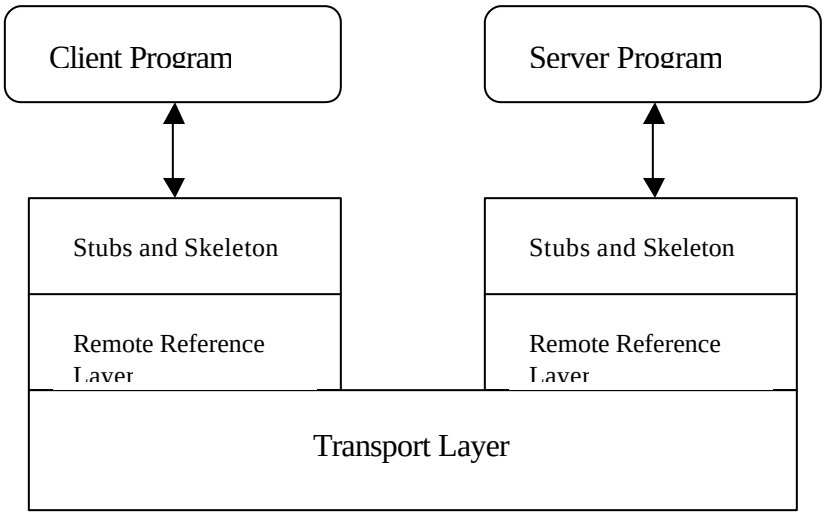

Figure 6.3 RMI Layers

The next layer is the Remote Reference Layer. This layer understands how to interpret and manage references made from clients to the remote service objects. The connection is a one-to-one (unicast) link, as shown in Figure 6.4. In the Java 2 SDK, this layer was enhanced to support the activation of dormant remote service objects via Remote Object Activation.

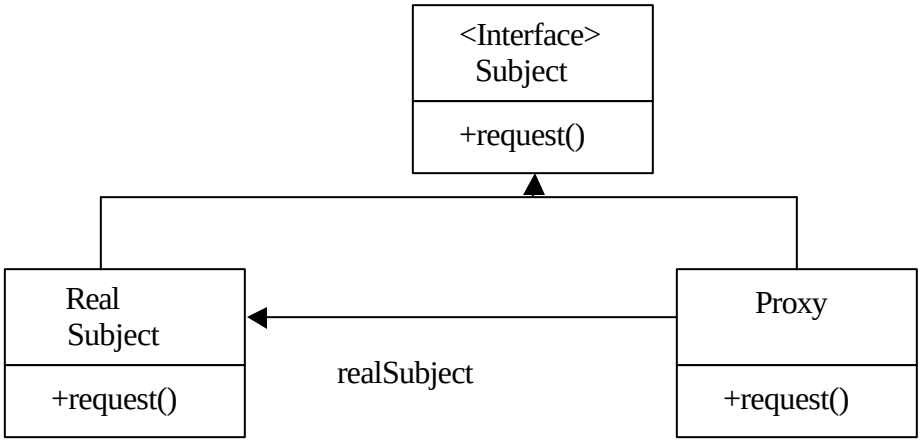

Figure 6.4 Interface and Proxy Objects

The transport layer is based on TCP/IP connections [27] between machines in a network. It provides basic connectivity, as well as some firewall penetration strategies.

6.2 Server Side Implementation

On a host machine, a server program creates a remote service by first creating a local object that implements that service. Next, it exports that object to RMI. When the object is exported, RMI creates a listening service that waits for clients to connect and request the service. After exporting, the server registers the object in the RMI Registry under a public name.

Interfaces: The first step is to write and compile the Java code for the service interface. The Site interface defines all of the remote features offered by the site to the clients, in Table 6.1.

Table 6.1 Site Interface Implementation

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Site extends Remote{
//cater s to the request made by the client
  int ServiceReq(int docNum) throws RemoteException;

//caters to the request made by other servers
  void ServerReq(int docNum) throws RemoteException;
}
```

Notice this interface extends Remote, and each method signature declares that it may throw a RemoteException object.

This is the pseudo code of the Site.java. We observe that one method ServiceReq is called by the Client to service a request. The Client passes the parameter, DocNum: Document Number, of the document it wishes to obtain.

Implementation: The implementation class uses UnicastRemoteObject to link into the RMI system. In the example the implementation class directly extends UnicastRemoteObject. This is not a requirement. A class that does not extend UnicastRemoteObject may use its exportObject() method to be linked into RMI.

When a class extends UnicastRemoteObject, it must provide a constructor that declares that it may throw a RemoteException object. When this constructor calls super(), it activates code in UnicastRemoteObject that performs the RMI linking and remote object initialization. The SiteImpl() Class is as shown in Table 6.2.

As our server also acts as a router, incases when it has to forward the request to other sites, it becomes necessary to calculate the closes server by the information provided by the Routing Table. The algorithm used to calculate the closest server is shown in Table 6.3.

Host Server: Remote RMI services must be hosted in a server process. The class SiteServer is a very simple server that provides the bare essentials for hosting.

Table 6.2 Site Implementation

```
import java.lang.*;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;

public class SiteImpl extends UnicastRemoteObject implements Site {

        //initializing all the variables.

        public SiteImpl(String host) throws RemoteException{
                super();
                 //calling methods for initialization of tables
        }

        public int ServiceReq (int docNum){
                //body
        }
        //other methods
}
```

Table 6.3 Computing Shortest Distance

```
public static void ComputeDistance(){
        int i,j,k;
           for(i = 0; i< maxNumOfSites; i++){
                 dist[0][i] = i;
                 dist[1][i] = networkTable[localServerName-1][i];          }

           for(k = 2; k <= maxNumOfSites; k++){
                 for(i = 0; i < maxNumOfSites; i++){
                    for(j = 0; j < maxNumOfSites; j++){
                                  if(i == j)
                                  {}
                                  else{
                                     if(dist[1][i] > dist[1][i] + networkTable[i][j])
                                    dist[1][i] = dist[1][i] + networkTable[i][j];
                                  }
                            }
                  }
           }
     }
```

Table 6.4 Server Site Implementation

```
Public SiteServer(String host){

       try{

          Site obj = new SiteImpl(host);

          Naming.rebind("rmi://"+host+"/HelloService", obj);

         System.out.println(host+" server is running…");

       }catch(Exception ex){

          System.out.println("Trouble…");

          ex.printStackTrace();

}
```

6.3 Client Side Implementation

On the client side, the RMI Registry is accessed through the static class Naming. It provides the method lookup() at a client uses to query a registry. The method lookup() accepts a URL that specifies the server host name and the name of the desired service. The method returns a remote reference to the service object. The URL takes the form:

rmi://<host_name>

[:<name_service_port>]

/<service_name>

where the host_name is a name recognized on the local area network (LAN) or a DNS name on the Internet. The name_service_port only needs to be specified only if the naming service is running on a different port to the default 1099.

The simplest version of our Client Code is shown in Table 6.5.

Table 6.5 Client Implementation

```
import java.lang.*;
import java.rmi.Naming;
import java.rmi.RemoteException;

public class Client{
//global variables declared.
   public static void main(String args[]){
        try{
           obj = (Site)Naming.lookup("rmi://"+args[0]+"/HelloService");
           message = obj.ServiceReq(Integer.parseInt(args[1]));
           if(message == -1)
                System.out.println("request was not serviced..");
        else
              System.out.println("request serviced with "+message+" cost");
        }catch (Exception ex){
           System.out.println("Error..");
           ex.printStackTrace();
        }
   }
  }
```

CHAPTER 7
EXPERIMENTAL RESULTS

We test ran our code for two scenarios. The first one has the document replicated at each site, and the other has the pages of document distributed at various sites. The efficiency of each algorithm depends on the number of client request, the traffic load, and also the frequency of updates made to the document.

The results were plotted in a graph as shown in Figure 7.1. Here we observe that initially, when the client request is low, Algorithm I works better. However, in the event of time, as the client request goes on increasing for various pages, the efficiency of the Algorithm II increases.

Then we plot a graph to show how frequency of updates affects the efficiency of the two algorithms, as shown in Figure 7.2. Here we observe that as the frequency of updates increase, the output of Algorithm I decreases. This happens because when the documents are updated too often, the update is to be propagated to the various sites. If the updates are large, a lot bandwidth is used up and also the traffic becomes more congested. In case of Algorithm II only the Master site has the ability to update the document. If updated, the changes are propagated only to those sites that have a cached copy of the document. The rest of the sites are unaffected by this change.

.

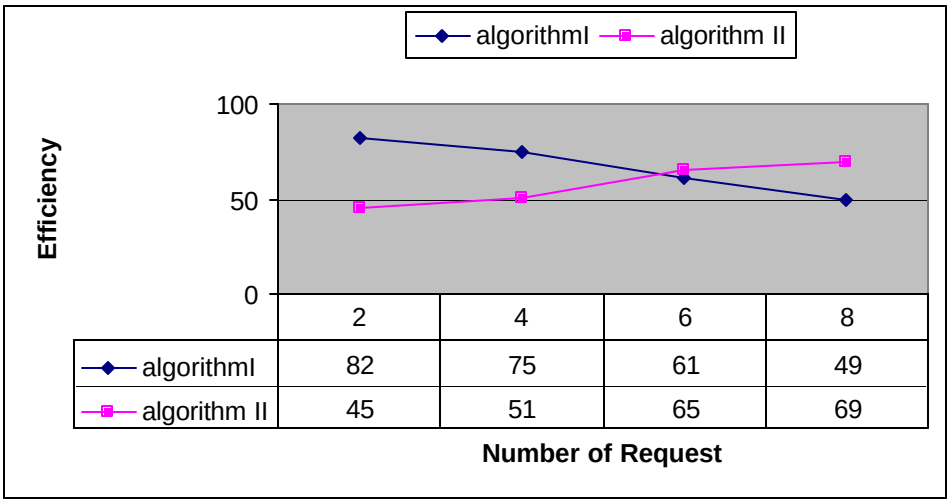

Figure 7.1 Graph of Efficiency versus Number of Client Request

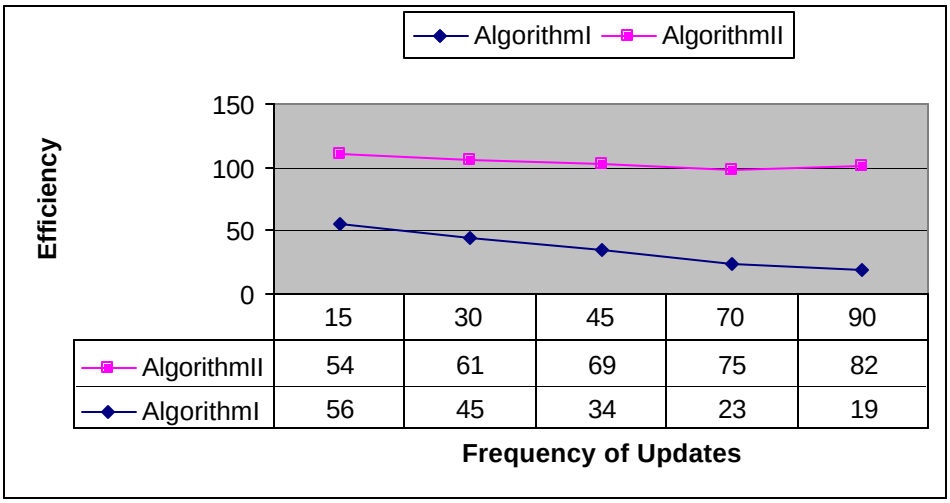

Figure 7.2 Graph of Efficiency versus Frequency of Updates

CHAPTER 8
SUMMARY AND FUTURE WORK

8.1 Summary

The challenges that developers and administrators face when creating a distributed Web site are content management, content distribution, distributed monitoring, site development, and security models. Site development is the process and technology that creates, stages, and deploys new content and software. Distributed sites have different staging and testing needs from sites served from a single machine. Those needs depend greatly on each site's structure, and are reflected in the procedures and technologies they use. Security models for Web sites are frequently written with the understanding that the Web site is run on systems within the enterprise, in one location, and on equipment owned by the company. Distributed sites require radical changes from the traditional security model. Security models for Web sites are frequently written with the understanding that the Web site is run on systems within the enterprise, in one location, and on equipment owned by the company. Distributed sites require radical changes from the traditional security model.

In our approach we have created an environment in which the pages of document are distributed to the various sites. This thesis explores the behavior of such a model under various situations. Different scenarios of client request are taken into consideration, and how they will be serviced is also predicted. We also ran various test cases and matched the output with those predicted theoretically. Due to the vastness of Internet, and

the continuous changing traffic conditions, bandwidth availability, security and reliability over network, are taken care of, by simulating the scenario.

The pros and cons of distributed and replicated documents are listed. It is observed that in situations when the traffic load is heavy and the client requests are concentrated in a particular area, distributed concepts works best. Also the hassles of maintaining the updated replicas of document at various sites, is reduced considerably. And the change is propagated easily across all the sites. Also this mechanism concentrates and reduces the work of a server, to a specific part of the document where changes occur very frequently.

8.2 Contribution

The specific contributions of this thesis are as follows:

- We formulated the various Routing Protocols for routing of client request efficiently to its destination.
- We implemented the various clients request that can occur and how our algorithm would process the request.
- We have an initial configuration of the network system, to start off with, which then takes care of the rapidly changing changes, simulating the real life Internet scenario.
- We implemented our servers to act as servers, catering to client request, and also as routers, to direct the request to the desired server site, in accordance with the algorithm
- We provided techniques in which the servers communicate with each other in a systematic way to have an updated picture of the whole set up, and also for

exchanging information especially when the changes to documents are carried out at one site, and the information is to be propagated to all the other sites for consistency.

- There are sets of Tables maintained at each of the site. We have described methods to update, and maintain consistency amongst all the tables at each site. Here the concept of Distributed Computing mechanism is extensively employed.
- There are also scenarios when a client request cannot be serviced. These situations and conditions are also explained in details. Approach to reduce such situation is another important feature of this thesis work

8.3 Future Work

In the design of non-distributed or centralized sites, the process flow from development to production follows the stages shown in Figure 7.1

The site's builder first works on a development site, which is typically unstable and marginally close to the production site. The code and content is moved to the staging site.

Figure 8.1 Non-Distributed System

The environment is then tested thoroughly, and any final bugs are ironed out. The last step is moving from staging to production. Once on the production server, a site is

live and accessible to the audience. When building centralized sites, the code is moved from a single server to a single server.

Distributed sites follow a different process, especially for the final two steps. As shown in Figure 7.2, multiple sites must be updated with the relevant content. At this point, developers run into a number of challenges with the speed and accuracy of the updates. Once the content has been distributed, how do you secure it against being tainted or attacked? Distributed site security models need to incorporate four key areas: enabling and disabling practices, remote management, physical security, and contractual obligations with third party providers.

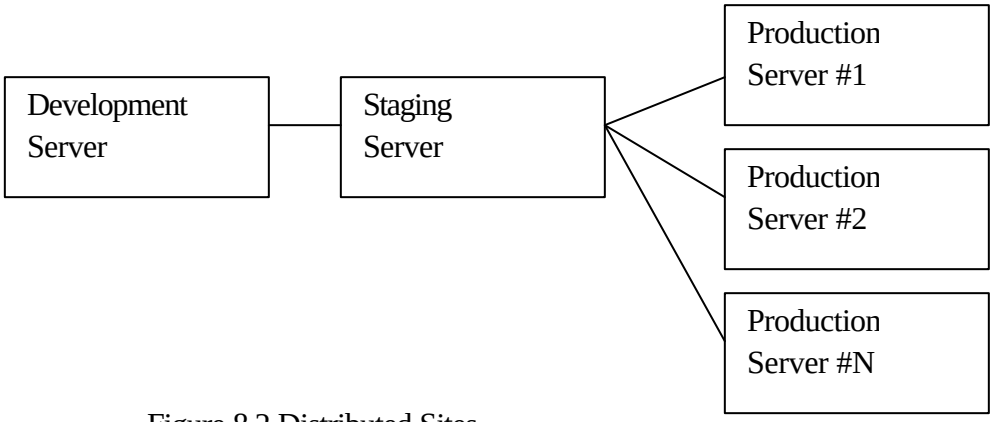

Figure 8.2 Distributed Sites

Updating the security policy to handle distributed conditions requires that you understand the security practices of your providers. For example, if a hosting provider does not meet or exceed the policies your organization observes, and then placing part of a distributed Web site in its care would weaken the security model you have created for the rest of your site.

Deciding to distribute your Web site is easy. Ensuring that each distributed site is current and secure, is the most difficult challenge. It all comes down to change practices. Development, staging, and production practices are different in fully distributed sites.

There are multiple locations that are deemed production servers and these must be consistent (or the Web site will be incorrect). Trusting a hosting provider to perform the same level of security or better than your own comes through understanding what the provider already does, through procedure and practice.

LIST OF REFERENCES

1. Accetta, Michael, Resourse Location Protocol, RFC 887, December 1983.

2. Akl, S. G., Parallel Sorting Algorithms, Academic Press, Orlando, Florida, 1985

3. Anderson, Eric, The Magic Router: An application of fast packet interposing, available from http://HHTP.CS.BERKELY.edu/~eanders/projects/magicrouter/osdi96-mr-submission.ps, May 1996.

4. Aversa, Luis, and Bestavros, Azer, Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting, Technical Report BUCS-TR-98-003, Computer Science Department, Boston University, Boston, February, 1998.

5. Besavros, Azer, Demand Based Resource Allocation to Reduce Traffic and Balance Load in Distributed Information Systems, Proceedings of SPDP'95, The 7th IEEE Symposium on Parallel and Distributed Processing, San Anotonio, Texas, October 1995.

6. Bestavoros, Azer, Carter, Robert, Crovella, Mark, Carlos, Cunha, Heddaya Abdelsalam, and Mirdad Sulaiman, Application Level Document Caching in the Internet, Technical report BU-CS-95-002, Boston University, Boston, February 1995.

7. Bolot, Jean-Chrysostome, End-to-end Packet Delay and Loss Behaviour in the Internet, Journel of High Speed Networks, vol. 3, page 345-372, 1993.

8. Braun, Hanswerner, and Claffy Kimberly C., An Experimental Means of Providing Geographically Oriented Responses Relative to the Source of Domain Name Server Queries, Technical Report, San Diego Supercomputing Center, San Diego, April 1994.

9. Carter, Robert, and Crovella, Mark, Measuring Bottleneck Link Speed in Packet-Switched Networks, Technical Report BU-CS-96-006, Boston University, Boston, 1996.

10. Carter, Robert, and Crovella, Mark, Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks, Technical Report BU-CS-96-007, Computer Science Department, Boston University, Boston, 1996.

11. The Commonwealth Scalable Server Project, available at http://www.cs.bu.edu/groups/cwealth, 1995.

12. Danzig, Peter, Obraczka, Katia, Dante, Delucia, Naveed, Alam , Massively Replicating Services in Autonomously Managed Wide-area Inter-networks, Technical Report USC-CS-93-5411, University of Southern California, Los Angeles, January 1994.

13. Deering Steven E., ICMP router discovery messages, RFC 1256, Cisco Corp., San Jose, California, 1991.

14. Guyton, James D., and Schwartz, Michael F., Experiences with a Survey Tool for Discoveting Network Time Protocol Server, Proceedings of the USENIX Summer Conference, San Diego, pages 257-265, June 1994.

15. Guyton, James D., and Schwartz Michael F., Locating Nearby Copies of Replicated Internet Servers, Technical Report CU-CS-762-95, Department of Computer Science, University of Colorado-Boulder, 1995.

16. Gwertzman, James, Autonomous Replication in Wide Area Inter-networks, Technical report TR-17-95, Havard University, Cambridge, April 1995.

17. Gwertzman, James, and Seltzer Margo, The Case for Geographical Push-Caching, Technical report, Havard University, Cambridge,1994.

18. Heddaya, Abdelsalam, and Mirdad Sulaiman, Web Wave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents, Computer Science Department, Boston University, Boston, 1993.

19. Hotz, Steven Michael, Routing Information Organization to Support Scalable Inter-Domain Routing with Heterogeneous Path Requirements, Technical Report, Computer Science Department, University of Southern California, Los Angeles, 1994.

20. The IBM Interactive Network Dispatcher, IBM Corporation, available at http://www.ics.raleigh.ibm.com/netdispatch, 1992.

21. Jacobson, Van, Congestion Avoidance and Control, Proceedings SIGCOMM '88 Symposium on Communications Architectures and Protocols Stanford, CA, pages 314-329, August 1988.

22. Knuth, D. E., The Art of Computer Programming, Vol.3, Addison-Wesley, Reading, Mass., 1981.

23. Krizanc, D., Oblivious Routing with Limited Buffer Capacity, Technical Report TR-14-87, Aiken Computation Laboratory, Harvard University, Cambridge, Mass., 1996.

24. Kunde, M., Optimal Sorting On Multi-dimensionally Mesh-Connected Computers, , STACS 1987, Lecture Notes in Computer Science 247, pp.408-419, Springer-Verlag, Berlin 1987.

25. Law, K, and Nandy, B. , Chapman A., A Scalable and Distributed WWW Proxy System Nortel Limited Research Report, Austin,1997.

26. Mogul, J., Network behavior of a busy Web Server and its client, Technical report DEC WRL Research Report, IBM, Austin, October 1995.

27. Perkins, C., IETF RFC2003: IP Encapsulation within IP, available at http://ds.internic.net/rfc/rfc2003.txt, 1991.

28. Postel, Jon, Internet Control Message Protocol, RFC 792, Cisco Corp., California, September 1981.

29. Rajashekaran Sanguthevar, and Thanasis Tsantilas, Optimal Routing Algorithm For Mesh-Connected Processor Arrays, Aiken Computation Laboratory, Harvard University, Cambridge, Mass., 1989.

30. Rajashekaran, S., and Tsantilas, T, An Optimal Randomized Routing Algorithm for the Mesh and A Class of Efficient Mesh-like Routing Networks, 7th Conference on Foundations of Software Technology and Theoretical Computer Science, Pune, India, 1986.

31. Scaling the Internet Web Servers, CISCO Systems, San Jose, California, a white paper available on the Web from http://www.cisco.com/warp/public/751/lodir/scale_wp.htm, November, 1997.

32. Valiant, L. G., A Scheme for Fast Parallel Communication, SIAM J. Comp. 11(1982), pp.350-361.

33. Wood, David C. M., Coleman, Sean S., and Schwartz, Michael F., A System for Discovering Network Characteristics and Problems, Proceedings of the USENIX Winter Conference, Boulder, Colorado, pages 335-348, January 1993.

BIOGRAPHICAL SKETCH

Madhurima Pawar, was born on December 12th 1977, in Pune, India. She received a bachelor's degree in computer engineering securing first class with distinction from University of Pune, Pune, India, in August 1999.

She joined the University of Florida in August 1999, to pursue a master's degree in the Department of Computer and Information Science and Engineering. She received her master's degree in August 2001.

Her research interests include routing, multicasting, TCP/IP protocol, and wireless networks.