APPROVAL SHEET

Title of Thesis: Design and Analysis of a Spatially Aware Peer-to-Peer System.

Name of Candidate: Prashant Punjabi
Master of Science, 2003

Thesis and Abstract Approved: _____

Dr. Konstantinos Kalpakis
Associate Professor
Deptartment of Computer Science and
Electrical Engineering

Date Approved: _____

Curriculum Vitae

Name: Prashant Punjabi.

Permanent Address: 4737 Aldgate Green, Baltimore, MD 21227.

Degree and date to be conferred: M.S., 2003.

Date of Birth: July 17, 1980.

Place of Birth: Mumbai, India.

Education:

University of Maryland, Baltimore County, M.S. Computer Science, 2003.
B.S. Computer Engineering, Mumbai, India, 2001.

Major: Computer Science.

ABSTRACT

Title of Thesis:

Design and Analysis of a Spatially Aware Peer-to-Peer System.

Author: Prashant Punjabi, Master of Science, 2003

Thesis directed by: Dr. Konstantinos Kalpakis, Associate Professor
Deptartment of Computer Science and
Electrical Engineering

Peer-to-peer systems (P2P) are proving to be a very popular paradigm. They are seen as obvious replacements to client server technologies, especially in implementations of large scale distributed storage systems. Distributed Hash Tables have emerged as an effective mechanism to implement such systems. There are several protocols to implement a peer-to-peer system as a distributed hash table, but none of these protocols in their basic forms take the physical location of the node in the network into account, while routing a request. This is because the links between nodes of such a system are completely *virtual*, and there is no relation between a direct link between two nodes and their spatial proximity.

We propose a scheme in which the arrangement of the nodes and their connections in the P2P system is a direct result of their locations in space. The obvious reasoning behind this is that such an arrangement should considerably reduce the total distance traveled by a message in the system, given the total number of

hops taken by a message to reach its destination does not change with this new arrangement. This is because if two connected nodes are close spatially, each hop of a message will have to travel a lesser distance as compared to the possible distance it would have to cover otherwise. The scheme we propose is using Chord as a base protocol and we suggest changes to make the system spatially aware.

The changes are in the form of an additional logical ring - where the nodes are arranged based on their physical locations. Each node in the system is now part of two rings. The spatial ring is intended to be used as a *cache* to accelerate access to popular documents. The plain ring is used whenever a requested document is not found on the spatial ring. Proactive load balancing is used to ensure that no one node in the system gets overloaded with too many requests.

We then go on to thoroughly analyze our scheme using experiments based on simulations. We address some of the issues that may arise because of the changes we describe. We provide detailed experimental results demonstrating that our system can offer significant improvements in system performance, when compared with systems that do not incorporate spatial information and are completely virtual. All of the above is achieved without introducing unnecessary load in the system.

Design and Analysis of a Spatially Aware Peer-to-Peer System

by

Prashant Punjabi

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
2003

To my parents.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Dr. Konstantinos Kalpakis for his help, support and constant involvement in guiding me towards my goals. I am also grateful to the distinguished members of my thesis committee, Dr. Stephens and Dr.Peng, for providing guidance and support. I would like to thank the staff members in the Department of Computer Science and Electrical Enginnering for helping me with the administrative details regarding my thesis.

This thesis would not have been a success without the contribution of my colleagues at LIST and friends who motivated me and helped me at all times. I am particularly indebted to Koustuv Dasgupta, Ripin Natani, Parag Namjoshi, Vasundhara Puttagunta and Meghna Kukreja, without whose support and encouragement it would not have been possible for me to complete this thesis.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . . **ii**

List of Tables . . . . . **iv**

List of Figures . . . . . **v**

Chapter 1 Introduction **1**

I Unstructured 2

II Structured 3

A. CAN 3

B. Pastry 3

C. Tapestry 4

III Contribution 5

Chapter 2 Related Work **6**

I Interest-based Locality 6

II Proximity Neighbor Selection 7

III Proximity Routing 8

IV Geographical Layout 8

Chapter 3 Motivation **10**

I Introduction to Chord 10
A. Simple Lookup 11
B. Accelerated Lookup 11
C. Joins 12
D. Stabilization 12
E. Leaves and Failures 13
II A Case for Proximity Awareness 14

Chapter 4 Design 17

I Preliminary Design 17
A. Incorporating Proximity Information 18
B. Preliminary Results 19
II The Problem 19
III Solutions based on Collision Avoidance (Reactive Load Balancing) . . 20
IV Solutions based on a Single Spatially Aware Ring and Replication . . 21

Chapter 5 Two Rings 22

I Changes to Basic Chord Protocols 23
A. Joins 23
B. Leaves 24
II Making Copies on the Spatial Ring 24
III Non-Uniform Distribution of Requests - Popularity of Documents . . 25

Chapter 6 Replication 28

I Need for Replication 28
II Popularity of Documents 29
III Replicas on the Spatial Ring 30

IV The Protocol 30
V Controlling the Replication 33

Chapter 7 Experiments 36

I Experimental Setup 36
A. BRITE 36
B. The Trace 37
C. Zipf-like Behavior of Requests 38
II Results 39

Chapter 8 Conclusion and Further Work 51

List of Tables

7.1 Comparison of performance and load for plain and spatial Chord . . . 39

7.2 Comparison of performance, load and replication overhead for the different cases 40

List of Figures

3.1 Hops on the consistently hashed ring can be across large distances in the underlying network, making the total distance traveled to answer a query really large 15

3.2 The total distance traveled in the underlying network is directly proportional to the distance traveled along the spatial ring 16

5.1 A *find* query switching rings, because of a *miss* in the spatial ring . . 26

7.1 The distribution of all the nodes in a plane 37

7.2 Comparison of the distance traveled per request, across the 3 systems 40

7.3 Comparison of the hops taken per request, across the 3 systems . . . 41

7.4 Load in the system for plain Chord 42

7.5 Load in the system for Chord with 2 rings, single replica 43

7.6 Load in the system for Chord with 2 rings, multiple replicas, maximum 15 replicas per document, maximum 20 new replicas created on a node each time replication protocol is run 44

7.7 The performance of the system deteriorates as document count is increased, if replication period is kept the same 45

7.8 The number of replicas created settles down when the system reaches a *steady state* - 1000 documents, 2 rings, multiple replicas 46

7.9 The number of replicas created stays high throughout the simulation - 10000 documents, 2 rings, multiple replicas 47

7.10 Histogram for the load on each node, for the system with only the plain ring 48

7.11 Histogram for the load on each node, for the system with 2 rings and one replica per document 48

7.12 Histogram for the load on each node, for the system with 2 rings and multiple replicas per document 49

7.13 Comparison of the performance of the system with two rings for different values of maximum number of replicas per document and number of replicas that can be created at a node at one time 49

7.14 Comparison of the load in the system with two rings for different values of maximum number of replicas per document and number of replicas that can be created at a node at one time 50

Chapter 1

Introduction

The peer to peer (P2P) paradigm has been gaining popularity in recent times. The omnipresence of the Internet and enterprise networks along with relatively cheap workstations equipped with reasonable storage capacity and computation capability have laid the foundation (created a need) for peer to peer networks. There are numerous advantages of such systems: scalability, robustness and the lack of a need for administration to name a few. Since the P2P architecture is truly distributed applications using this architecture are being touted as more robust and secure than those that are centralized. In this time and age where security concerns are prime, especially for sensitive data, such as data related to the security of a nation, P2P is emerging as one of the leading architectures which can provide the foundation of a data-store resilient to cyber-terrorism as well as physical attacks. This is because using replication and distribution of storage it is virtually impossible to destroy knowledge by attacking a single site [**?**].

There are several issues that need to be considered while designing an architecture for a peer to peer system. Among them are how to devise truly distributed mechanisms for a node to join or leave the system, and the allocation of the shared resources (processor capacity, storage) among the nodes in the system.

However, the architecture of a P2P system revolves around the most common and important function that has to be implemented for it to be usable and scalable: the ability to search the P2P network for a resource. Attempts to come up with a efficient and fault-tolerant routing have led to two basic ways of carrying out a search operation in a P2P network : *Unstructured* and *Structured*.

I Unstructured

The unstructured approach is the one adopted by public P2P systems, such as Napster (which is now defunct), Gnutella or Kazaa [**?**, **?**]. The popularity of these systems has been growing tremendously in the past few years. In such networks users typically share resources such as documents, images and media files. A user that needs some resource can get the same from some other peer in the network that has it. This involves a search for the resource throughout the network of peers. The search in Napster is centralized and therefore, carries with it the baggage of such an implementation which includes not being scalable and having a single point of failure. In Gnutella, each node maintains an index of the resources it is exporting and the search proceeds via flooding. A node will query its neighbors and they will in turn query their neighbors. The potential of an explosion in network traffic is averted by limiting the radius of such a search, which in turn compromises the result of the search. Pandurangan and Raghavan [**?**, **?**] describe how a Gnutella type network can be constructed with a low diameter. They go on to describe a protocol that ensures the degree of any node in the system to within constant bounds and thus the lower and upper bound on the connectivity of the network.

II Structured

So far the more successful attempts to provide a *structured* architecture for P2P systems have focused on implementing them as *distributed hash tables*. In this paradigm for building P2P networks, the resource is represented using a *key* and then stored at that node in the system to which this key maps to. Examples of such systems include CAN [**?**], Chord [**?**], Pastry [**?**] and Tapestry [**?**].
In the following paragraphs we introduce CAN, Pastry and Tapestry. Chord is described later in considerably more detail.

A. CAN

CAN (Content Addressable Network) routes messages in a d-dimensional co-ordinate space. This space is purely logical and bears no resemblance to any physical space. At any given time the entire co-ordinate space is dynamically partitioned so that each node in the system controls and owns it individual space. A key that needs to be stored in the system is first hashed onto a point in the same d-dimensional space and stored on the node that controls that space. A node join takes place by splitting a zone with the node that is already in that zone. The neighbors of this node are then informed about the new node and this new node can now participate in the routing. Routing takes place by forwarding a request to node that has the coordinates closest to the coordinates of the destination node.

B. Pastry

In Pastry, each node has a 128-bit identifier that can be computed by applying a secure hash function on the nodes public-key or IP address. Any given key is reliably

routed to the node whose identifier is the closest to the key among all live nodes. The keys or identifiers can be visualized as a sequence of digits with base 2^b where b has the typical value of 4. For a network with N nodes, the routing table has $log_{2^b}N$ rows, with $2^b - 1$ entries per row. The entries in row n are those that match the current node in the first n digits and have any of the $2^b - 1$ possibilities for the $n+1^{th}$ digit. In addition to this table each node also maintains a leaf set with l entries, half of which are those nodes that have numerically closest larger identifiers and the other half are those with numerically smaller identifiers. Pastry can typically route a request in less than $log_{2^b}N$ steps and guarantees delivery unless all the nodes in either one of the 2 parts of the leaf set fail simultaneously. Pastry also tries to make routing more efficient by including the *closest* node in its routing table, when given a choice.

C. **Tapestry**

Tapestry is a routing protocol that forms the routing infrastructure for OceanStore [**?**]. It is based on the work done by Plaxton et al [**?**]. Routing is done by finding the longest prefix match between the destination node and the entries in a given nodes routing tables. Each node maintains routing table having a level for each digit in the node identifier where each level contains entries that point to nodes that match the suffix associated with that entry.

With the wide scale use of P2P applications, increasing portions of network traffic generated will be due to the use of such applications. Therefore, there is a need to minimize such traffic. Since P2P applications are typically implemented on top of an overlay network [**?**], if the the network distance between adjacent nodes

on the overlay is minimized, there will tend to be a decrease in the total amount of traffic generated due to that application. Pandurangan et al [?, ?] try and resolve this issue by restricting the number of connections to and from every node in the network and trying to keep the diameter of the network to the minimum possible. Also the protocol they propose ensures that the diameter of the network increases logarithmically with the size of the network.

III Contribution

Our contribution in this thesis is a scheme to incorporate proximity information in Chord, which is one of the architectures to build P2P systems. We describe the scheme and then analyze it thoroughly, and go on to show that incorporating proximity information leads to a significant performance improvement. We propose a unique scheme that uses proximity information to make search more efficient and replication for load balance, while ensuring correctness at all times by using the original Chord protocols as a safety net.

The rest of this thesis is organized as follows. In the next chapter we present a short survey of techniques that have been used for improving the routing efficiency of P2P systems. We then discuss Chord in detail and describe our motivation behind using proximity information to improve routing performance. In chapters 4, 5 and 6 we present the design of our system and the related protocols and algorithms. In chapter 7 we present the experiments we ran and the results observed. Finally we conclude the thesis.

Chapter 2

Related Work

In this thesis we are trying to put forward an architecture which will result enhancing the performance of peer to peer systems. We attempt to achieve this using techniques that exploit the relative positions of the peers in the real underlying network. In this chapter we will attempt to cover all efforts made which are geared toward similar goals of improving the performance of a P2P system.

I Interest-based Locality

In [?] Sripanidkulchai et al try and improve the performance of a Gnutella-like system by using interest-based locality. They do so by providing shortcuts to where the required content may be to improve the performance. If the shortcuts fail, the underlying system will always find the required content if it exists, since the shortcuts are supposed to be used only as hints. The main philosophy of creating these shortcuts is that a peer which is querying for some specific content is more likely to find that content on a peer that shares the same interests as itself. Shortcut discovery is done by piggybacking initial flooding queries with requests to create shortcuts. These shortcuts are then continuously ranked based on their perceived usefulness. This ranking is used to decide both the order in which the shortcuts are queried and also

which shortcuts to evict if space is needed to create more shortcuts. The authors also mention that this same technique can be used to improve the performance of Structured Distributed Hash Table based P2P systems, such as Chord, but with the assumption that a node in the system will store only the content that it itself is interested in.

There have been various efforts to make efficient peer to peer systems which utilize the proximity between nodes in the underlying network. Castro et al [**?**] mention 3 different ways in which the proximity of nodes can be exploited for improving the performance of a P2P system.

II Proximity Neighbor Selection

In a peer to peer system each node has to have some information about some other nodes in the system, in order to be able to route correctly and in the direction of the node eventually responsible for the request. For this purpose, nodes maintain routing tables. The number of entries and the manner in which these tables are arranged vary from system to system. Some of the systems mentioned earlier, like Pastry ([**?**]), Tapestry([**?**]) and CAN ([**?**]) have some flexibility in the manner the entries in their routing tables are populated. Therefore, whenever there is a choice of two or more nodes for the same entry in the routing tables, the node that is physically closest to the current node is chosen. This means that in addition to the requests moving closer to the final destination in every hop, this technique also minimizes the distance that the request has to cover in each hop. The goal of such a technique is that finally, the total distance traveled by a request packet (the sum of the distance covered in each hop) should be within a constant factor of the distance between the source and destination of that request.

III Proximity Routing

In this technique, the routing tables are constructed without considering any proximity information at all. While routing a request, the next hop is chosen from the existing entries in the routing table based on its proximity to the current node and also the progress it affords toward the final destination in the virtual node identifier space. This basically involves a compromise. In making large advances in the virtual space, a request may be forced to travel large distances for each hop. Using proximity routing techniques may result in covering lesser distances in the the virtual space and traveling smaller distances in the underlying Internet, but at the same time, risking additional hops to reach the final destination. A version of this protocol is used in CAN. This has also been suggested as one of the techniques that can be used to make Chord more efficient, and is used in CFS [**?**], a peer to peer wide area file system which uses the Chord protocol in its routing substrate.

IV Geographical Layout

This technique attempts to map the node identifier space onto a physical space while constructing a peer to peer network. The system is no longer *virtual*, since the positions of the nodes within it mirror their locations in the real internetwork that they are a part of. This technique has been explored as one of several to improve the routing performance of CAN. It managed to effect an improvement in performance, but at the cost of losing its ability to be self-organizing. A set of well-known landmark servers is required to decide the coordinates of a node in the CAN space.

The main issue that has come about in discussions of using geographical layout to organize the connections in a peer to peer network, is that it will cause an imbalance in the way the node identifiers are distributed in the identifier space. This will in turn lead to an imbalance in how the resources are distributed, which may lead to

hot spots. Furthermore, protocols such as Chord and Pastry depend on the nodes that are close together in the virtual identifier space to be separated in the real world. This is because causing a set of nearby nodes to fail may lead to incorrect routing of requests and render some nodes in the system to be unreachable from some other nodes.

In this thesis we propose a technique that uses this geographical layout information about the nodes in the system to improve the performance of the system. We do so by using geographical information in addition to the basic protocol which in our case is Chord. Chord is used unchanged as a *safety net*, in case our technique fails to locate the required resource the underlying Chord system will always be able to find it. Not unlike the technique that uses shortcuts to find the nearest replica, we also attempt to locate the nearest replica and at the same time ensure that the distance traveled in finding this replica is proportional to the distance between the source and the destination (the node that holds the replica used to service the request).

Chapter 3

Motivation

I Introduction to Chord

Chord [?] presents a distributed look up scheme that can be the backbone of a peer to peer system. The placement of keys is based on consistent hashing, which has very good load balancing properties. This load balancing is with respect to all nodes in the system receiving approximately the same number of keys. For example, when the N^{th} node joins the system which already has K keys, the number of keys that are moved to a different location is only $O(K/N)$, which is the same as the minimum necessary to maintain a balanced load. This system is made scalable by not requiring every node in the system to be aware about every other node in the system. Efficient routing can be achieved by having a node to have information about only a small number of other nodes in the system.

Every node and key is assigned an m-bit identifier. This identifier is arrived at by hashing the node's IP-address. The identifier for the key is calculated by hashing the name of the key.

The range of identifiers possible for use with nodes and keys, for an m-bit identifier, is 0 to $2^m - 1$. These identifiers can be visualized as arranged along the circumference of *identifier circle* modulo 2^m. Consistent hashing is used to assign keys to

nodes. A key with identifier k will be placed on the first node whose identifier value is more than or equal to k. This node is called the *successor node* of the resource k and is denoted by *successor(k)*. In this manner, consistent hashing allows nodes to easily join and leave the system. When a node n joins the system, some of the keys that were previously assigned to the successor of n are now assigned to n. Similarly, whenever a node n leaves the system, all the keys it holds are assigned to its successor. These are the only changes in the allocation keys to nodes that occur in the system.

A. **Simple Lookup**

Each node in has a pointer to its *successor* node, which is the node that immediately follows the current node along the identifier circle. This is the only piece of information needed by each node in the system to ensure the correct working of the system. Correct working means that the node that the key being looked up for can always be found. In this case, where each node is aware of only its successor, the lookup proceeds by following successor pointers around the circle till the successor of the key being looked up is found. The number of messages that will be exchanged or hops that will be made in order for the lookup to terminate is linear in the number of nodes in the system.

B. **Accelerated Lookup**

Chord also maintains some additional routing information, which is used to accelerate lookups. For an identifier length of m bits, each node n maintains a table with m entries, and this table is called the *finger table*. The i^{th} entry in the table contains the identity of the first node s, that succeeds n by at least $2^i - 1$ on the identifier circle. s is therefore the i^{th} finger of node n. The first finger of node n is nothing but its successor node.

The finger table is used to efficiently route a query for a resource and the algorithm used can be described as follows. Say the resource has identifier i, and the query is initiated at a node n. Now, if id falls between n and its successor, node n returns its successor as the location of resource id. Otherwise n searches it finger table for a node p, whose identifier most immediately precedes id and this same lookup procedure is repeated at p.

Each node can forward a query at least half way along the remaining distance between the node and the target identifier. This means that, with high probability, the number of nodes visited in order to find a successor in an N-node network will be $O(logN)$

C. **Joins**

The only requirement for a node that needs to join the system is to be aware of at least one node that is already a part of the system. A node n that wants to join the system will call a method n.join(n') on itself, where n' is the node known to n. This join method in turn asks n' to find the successor of n. Once the node n knows its successor n_s, it can update its successor pointer and also finger table entries by copying those from n_s. The keys that n is the new successor of, instead of n_s are copied over from n_s. Also, n_s will now acquire node n as its new predecessor. The new node is now part of the system and can participate with its working as soon as other nodes discover its presence, which happens with the help of the stabilization protocol.

D. **Stabilization**

To ensure that lookups proceed as expected in the face of nodes joining and either leaving voluntarily or failing, the successor pointer at each node must be kept

current. This is done by using a basic *stabilization* protocol. Every node runs this stabilization protocol and this is the only way a node can learn about the presence of a new node. And since this stabilization routine is run fairly regularly on each node, information about a new node is incorporated into the system fairly quickly. The stabilization routine proceeds as follows. Every node n will send a message to its successor s, inquiring about its (the successor's) predecessor p. Next it will decide whether in fact it should have p as its successor instead of s. This will be the case if the node p has recently joined the system. If in fact the node p should be the new successor of n, node n will update its successor pointer and also send a notification to node p to check whether node n should be its true predecessor instead of its current predecessor.

The finger tables are kept up-to-date using a similar protocol. For a more detailed description of the stabilization protocols, please refer to [**?**].

E. Leaves and Failures

Nodes failures can undermine Chord and cause it to crash if each node is not aware about its successor. To make the system more robust, each node can be required to maintain a list of successors of some size r. If the successor of a node has failed, the next entry in the successor list can be used in its place. All the nodes in a nodes' successor list will have to fail simultaneously for the correct operation of Chord to be disrupted, a scenario which is unlikely, given a reasonable value for r. A modified version of the stabilization protocol, not discussed here, can be used to maintain the successor list. Procedures for finding closest-preceding-node is also updated suitable to accommodate the successor list.

A scenario where a node that is leaving the system voluntarily, can also be treated as a failure. The robustness of the Chord protocol will take care of the correct

working of the system. However, some operations can be carried out to improve the performance of the system when a node leaves gracefully. First, all the keys that are assigned to the node that is now leaving can be assigned to its successor, which will in any case be the new successor of the keys. Also the node n that is leaving the system may inform its successor s and its predecessor p before leaving. The successor s will then update its own predecessor pointer to point to p, the predecessor of n. Also, the predecessor will remove the entry for node n from its successor list and add the last node in n's successor list as the last node in its own successor list.

II A Case for Proximity Awareness

In Chord the placement of the nodes is purely virtual, which means that the physical locations of the nodes have no bearing on where the nodes will be placed along the ring. This is because of the method of generating the identifiers that decide the positions of the nodes along the ring, as mentioned earlier, is based on consistent hashing.

A query in Chord may proceed in an iterative or recursive manner. In the iterative technique, the initiator of a requests controls its routing. At the end point of each hop, the control goes back to the initiator and the request makes the next hop from there. In the recursive technique, the next hop for a request is decided by the end point of the previous hop. The initiator of the request just starts the routing of the request and then lets the other nodes bring it to completion. The case we will consider is the one in which the queries (requests) are handled in a recursive manner.

For a peer to peer network with diameter D, each hop on average will travel a distance of $D/2$. Some hops may have to travel the complete diameter of the network. With each request taking $O(log(N))$ hops to completion, the total distance traveled by the request packet is of the order of $O(D * logN)$. Figure 3.1 shows an example

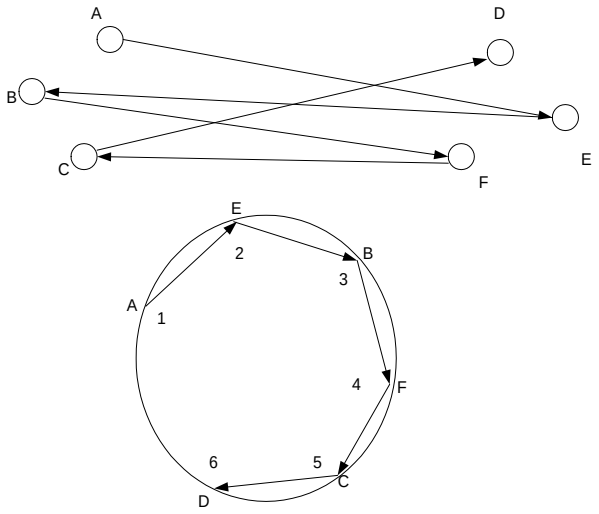

FIG. 3.1. Hops on the consistently hashed ring can be across large distances in the underlying network, making the total distance traveled to answer a query really large

of a query routed over a consistently hashed Chord ring. The actual locations of the nodes are also shown. The query is initiated at node 1, which is mapped to location A and the destination is node 6 which is location D. The path followed by the request is shown as 1 - 2 - 3 - 4 - 5 - 6. Since the mapping of nodes is completely virtual, in the above shown scenario it turns out that each hop has the request packet going all the way across the breadth of the network, which makes its total distance traveled really large.

Figure 3.2 shows the path followed by the same request, but in the scenario where the mapping of nodes to locations is done in a spatially-aware fashion. Because of the way the nodes are mapped to their locations, the total distance traveled by a request will be proportional to the total distance traveled by the request along the ring. For a network with diameter D, the total distance traveled by a request will be $O(D)$. This will definitely lead to an improvement as far as the total distance traveled by the request packet is concerned.

For a network with the properties of the Internet, the latency of a data packet

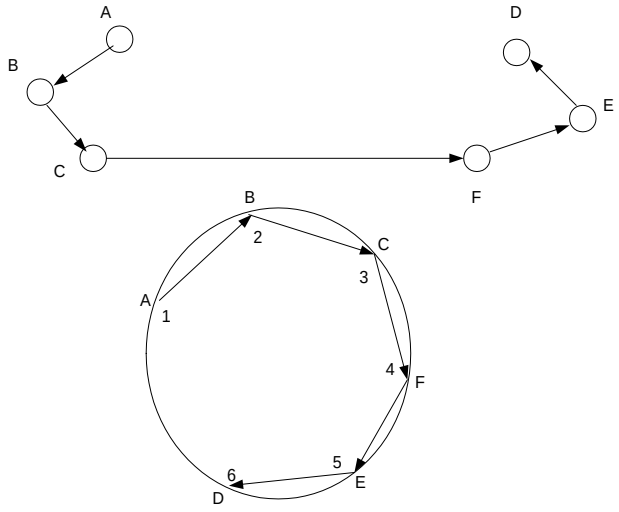

FIG. 3.2. The total distance traveled in the underlying network is directly proportional to the distance traveled along the spatial ring

between a pair of nodes can be approximated as proportional to the physical (network) distance between these two nodes. A system that attempts to reduce the distance between communicating nodes in such a network will therefore lead to an improvement in the overall performance of the system.

Chapter 4

Design

In this chapter we introduce the preliminary design of our system. This explains in detail how we incorporate proximity information into Chord and also reveal some results that point to the tremendous potential of such a scheme. We also discuss how this first version of our system destroys the distribution of nodes about the Chord ring. This imbalance in node distribution leads to hot spots where some of the nodes are overloaded as compared to the rest of the nodes in the system. In the next chapter, we introduce our scheme where we have a system with 2 rings, one ring that incorporates proximity information (spatial ring) and the other that is similar to the one in chord (plain ring). This is intended to reduce the load on the hot-spots by delegating the responsibility to serve some requests to the plain ring. We present results that show that this method succeeds, in some cases, to alleviate the load imbalance created by the non uniform distribution of the node identifiers in the identifier space.

I Preliminary Design

In the system we are proposing, the positions of the peer nodes along the circumference of the Chord ring should reflect their actual physical locations. Consequently, the distance between two nodes along the ring will be indicative of the real distance

between the two peers in Euclidean space. This is achieved by using the location of the node to generate its identifier, which decides its placement on the Chord ring. Therefore, the only difference between Chord and the first version of the system we are proposing: instead of using consistent hashing on the address of the node, we use its physical location to generate its identifier which decides its place on the ring. All the other protocols for the working of the system and also the data structures used to accelerate routing will stay the same. This means that the performance guarantees that Chord gives will still be valid. Each request will be, on average, serviced within $O(logN)$ hops. But now the distance traveled in hops across the circumference of the ring is proportional to the actual distance between the source and destination of the hops. It is not any arbitrary distance as it is with Chord.

A. Incorporating Proximity Information

For our system, we have the location of a node specified as its coordinates on a plane. The identifiers for these nodes are generated using these coordinates. The two dimensional coordinates are mapped into a one dimensional scalar value which serves as its identifier. This mapping/transformation is done using methods such as bit interleaving or Hilbert Space filling curve. In our experiments we used the *bit interleaving* to generate identifiers for the nodes in the system from their physical locations. For identifiers generated in this manner, nodes that are physically close to each other in space will have identifiers values that are close to each other. As a result, these nodes will be placed close to each other along the circumference of the ring.

B. Preliminary Results

Preliminary experiments reveal that using geographical information to construct the Chord ring has a tremendous potential to improve performance of the protocol. For example, in our simulations we observed a performance gain of almost 100% when we put the same event traces through the two systems and compared the results. But making this change also results in a severe load imbalance being introduced in the system. For details please refer to the **Experiments** chapter.

II The Problem

The preliminary results reveal that simply incorporating proximity information in Chord will not be a complete solution. This is because of the fact that we are discarding the *consistent hashing* approach of generating the node identifiers. Due to this change, the node identifiers are not evenly distributed throughout the identifier space. If the node identifiers are visualized as points along the circumference of a ring, there will be some parts of the ring that have a lot of points, and then there will be some parts that have no points at all. This in turn will affect how the resources these nodes are responsible for will be distributed among the nodes. For example, some nodes will be responsible for a lot of documents and others may have very few or no documents placed on them. This change may have adverse effects to the load balance in the system and cause it to deteriorate. This is one of the main concerns about such a system expressed by Ratnasamy et al in [**?**]. Thus, there is a need for some load balancing mechanisms to be introduced to make the operation of the system feasible and at the same time retain a large percentage of the benefit that comes from making the system *spatially aware*.

III Solutions based on Collision Avoidance (Reactive Load Balancing)

Since Chord is based on a hashing scheme, the collision avoidance/ resolution schemes used normally in hashing would seem to offer promising solutions. But this is not the case. One would imagine schemes such as double hashing, or quadratic probing to work as follows. If a document arrives at a node that already has too many documents which means it is excessively loaded, because this node will now be serving requests for all these documents. The solution can be that some of these documents are moved to some other node. This other node will be found using the open addressing techniques used to resolve collisions during hashing. And if a query arrives at a node, which has moved some documents off to some other node, it will have to somehow forward the request to the node that actually stores that document.

But this approach gives rise to a host of other problems. The first among these is keeping track of documents that have been moved off a node because of excessive load on that node. This means that either the loaded node keeps a pointer to the node that has the document or initiates the same collision resolution algorithm every time a query that comes to it finds that the resource (document) requested is not present.

To do this in the presence of the possibility of all the nodes involved leaving and new nodes that might be successors of some of the documents joining the system make these possible solutions practically expensive to manage with strict requirements of correctness and consistency that this system has to live up to.

But on the other hand Chord as it is, without proximity awareness and without any kind of spatial information achieves a reasonably good balance of the load. This is done by taking advantage of the consistent hashing function in generating node and document identifiers.

Also this scheme does not accommodate the presence of an extremely popular

document on a node. The rate of requests for this document at a single node may be sufficient to overload that node.

IV Solutions based on a Single Spatially Aware Ring and Replication

One more avenue we explored in attempting to come up with a load balancing scheme for the spatially aware system was *replication.*To relieve the load on the nodes that are overloaded, we could make replicas of documents that are stored on these nodes on other nodes that do not store too many documents and so are not inundated with queries. This would result in the load on the nodes that store a large number of documents being reduced, if the replicas of these documents were used to respond to a query.

The problem in this scheme is guaranteeing that if a replica of a document on an overloaded node cannot be found, it can still be served off its original location. Given the Chord routing protocol, there is no known set of nodes that a given request for a document always travels through. It is very hard, if not impossible, to place replicas of a document on some nodes other than the successor and then to guarantee that a replica will be used should the successor become overloaded. A request for a document may skip over all nodes that hold replicas and reach the successor of the document being queried. And if the successor is overloaded, it will not be able to serve this request.

The inherent problem with this scheme is the intention to use the successor of the document on the spatially aware ring, if a replica cannot be located. This successor may already be overloaded and unable to serve any more requests.

Chapter 5

Two Rings

The problem with the spatial approach as discussed earlier is that it might affect the load distribution. So using a completely spatially aware architecture to replace the existing mechanism of generating node identifiers using consistent hashing will not work because a few nodes will be extremely overloaded as compared to the rest of the nodes in the system. However, the scheme of having nodes arranged on the ring according to their actual locations such that nodes that are close to each other in Euclidean space will also be close to each other on the Chord ring, has some obvious advantages of lesser *distance* to traverse for answering a query. This translates into better performance since now the query will be traversing lesser distances in the underlying real inter-network. This also means that a request packet will stay in the network for a smaller period of time, which eventually will lead to fewer packets being routed through the network at any given time.

And so what we are proposing is a novel scheme which improves the performance of the system as a whole by incorporating the 'spatially-aware' scheme as a component. This is achieved in the following manner.

Instead of having a system with just one ring with node identifiers that are generated using a consistent hashing, we now require all nodes to be a part of two

rings. One of the rings is the one mentioned above, which we will refer to from this point on as the *consistently hashed ring* or simply as the *plain ring*. The other ring has the nodes arranged according to their actual locations, by using identifiers that are spatially aware. In the rest of the thesis we will refer, to this ring as the *spatially aware ring* or just the *spatial ring*.

The spatially aware ring is intended to be used as a cache. The purpose of this cache is to speed up access to documents that are frequently requested for by other nodes in the system.

The protocols to handle nodes joining, leaving or failing have to be extended to incorporate this important change of the addition of another logical ring.

I Changes to Basic Chord Protocols

A. Joins

Every node in the system is a part of these two virtual rings, which means that when a node *joins* the system it acquires a pair of identifiers. One is the Chord identifier generated using consistent hashing. The other identifier is the one that will be used in the spatially aware ring and is generated by using a transformation on the actual location of the node. The protocol by which a node joins a Chord P2P system is then run twice for this new node, once for each of the 2 virtual rings using the respective identifier that this node is assigned for that ring. In this manner this node becomes a part of each of the two rings. Each node will therefore have to maintain complete local information, viz successor pointers, predecessor pointers and finger table and finger cache for each of the two rings that it is a part of. The relative positions of two nodes will more often than not be completely different for the two different rings.

B. Leaves

Similarly, when a node *leaves* the system, the pointers to it in both rings will have to be updated. The leave operations in Chord will happen as usual in both the rings. These include moving of documents in the plain ring to their new successors and updating the successor and predecessor pointers for the neighbors of the node leaving the system in both the rings. The documents on the spatial ring are not moved to their successors since they are essentially replicas and are expendable.

All the system maintenance messages like *stabilize, replace successor*, etc will always be between two nodes in the same ring.

A document when inserted in the system is placed only on the plain Chord ring. The spatially aware ring essentially serves as a *cache*. Documents are moved over to this ring as and when they are accessed.

When a query for a document is initiated, first the spatially aware ring is searched for it. If this document is present on its successor on the spatial ring, the request is served from this ring itself. If this document is not present on its successor on the spatially aware ring, the query is forwarded to the plain ring. The document will always be present on its successor on the plain chord ring because this ring holds *all* the documents. The request is now served from the successor of the document in the plain ring, and the fact that there was a request for this particular document is recorded on the spatial ring. This information will be used later to possibly make copy of the requested document on its successor node on the spatial ring.

II Making Copies on the Spatial Ring

This is done by periodically running a protocol on each node. Each node has an indication of the average load in the system. This average load is used to decide the threshold load on that node. It is typically within a constant number of standard

deviations of the mean load. Now, if the load (number of requests serviced) of this node in the last period is lesser than the threshold load, it is considered to be a lightly loaded node. It can now make replicas of documents on its spatial part, based on the number of requests it has forwarded for each of them. The expected increase in the load on this node due to the creation of a replicas is taken to be the number of requests this node has forwarded for this replica. Replicas are created on the node in this manner for as long as the threshold set at the node allows. For example, if the creation of a copy on the spatial part of a node will send its expected usage for the next period to above the threshold, that copy is not created. This replication function is run on every node periodically, and replicas are created based on the requests forwarded by the successors of the original documents on the spatial ring and also the available capacity of the node to handle additional requests.

When this protocol is run again on a node that already has some replicas on it, first all the replicas are deactivated. A replica that is not active cannot be used to answer a request for the document it is the replica for. The mechanism that will be used to create replicas is almost exactly the same. The only difference is that the documents currently cached are also considered in the calculation of which documents should be placed on the spatial node. The counts used are the number of times these documents were accessed from the spatial part. If a document that is already present on a node needs to be cached, it is merely *activated* and now it can be used to respond to queries.

III Non-Uniform Distribution of Requests - Popularity of Documents

The system that we are proposing can be especially useful in a system where there is a non-uniform distribution of requests for documents.

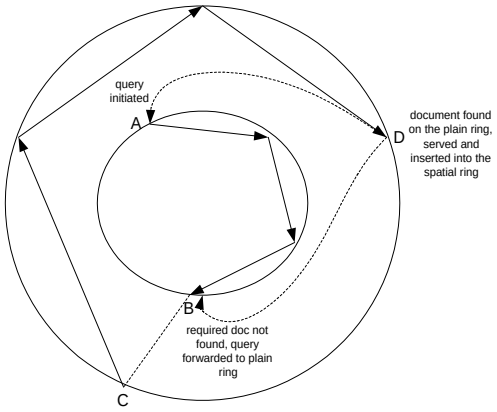

FIG. 5.1. A *find* query switching rings, because of a *miss* in the spatial ring

Data access patterns for certain class of systems suggest that a very small portion of the data is accessed very frequently and most of the data is accessed very rarely. This means that, if we manage to keep the frequently accessed documents in the spatially aware ring, there will be sufficient improvement in the performance for most requests to overcome the occasional extra overhead of a document being needed to be fetched from the other ring. This overhead is obviously due to the prolonged search procedure that will have to be executed for such requests. The above statements can be verified using experiments based on simulations.

For example, for a sequence of n requests, let r be the hit ratio for requests being served from the spatial ring. Let the average distance traveled by a query in the spatial ring and the plain ring be d_s and d_p, respectively. The sum of the distances traveled by all n requests, in a system with 2 rings, will therefore be

$$d_T = n * (r * d_s + (1 - r) * (d_s + d_p)) = n * (d_s + (1 - r) * d_p)$$

The sum of the distances traveled by n requests in a system with only a plain ring will be

$$d_T = n * d_p$$

As is obvious from the above equations, for the system with two rings, every request has to go around the spatial ring. To make our technique effective, the hit ratio should be high enough such that $d_s - r * d_p < 0$. If $d_s = r * d_p$, this means that there is no improvement in the performance and so all the extra effort being expended in maintaining two rings is not yielding any benefits. To make the extra work worthwhile, we need $d_s << r * d_p$, and thus r should be sufficiently high.

Preliminary experiments suggest that d_s/d_p is approximately 0.5 and so if we are able to realize a hit rate of about 80%, we should be able to extract a significant improvement in performance.

Therefore, if trends for accesses on documents stored in this system are based on *popularity of documents*, and most of the popular documents have replicas in the spatially aware ring, there will be an improvement in the performance of the system, because most of the requests will now be served off the spatially aware component of the system.

Chapter 6

Replication

I Need for Replication

Typically in a scenario where the files that are being stored in the system are web documents, not all documents will be accessed with equal probability. Some documents are bound to be more popular than others. This behavior is also prevalent in peer to peer file sharing systems. In this type of a situation, the nodes with the more popular documents might get overloaded. This overloading will occur even if the system is arranged in the manner described by Chord i.e. without any spatial awareness in the system.

The only way to alleviate the load on nodes that store these popular documents is to make replicas of these documents at various other nodes, and use these replicas to service requests for these documents.

In this chapter we describe a replication scheme that is designed to relieve the load in the system due to documents that are very popular and so have a high request rate. We show how this scheme also contributes to make the performance system more efficient. Basically it involves making replicas of the popular documents on the spatially aware ring. These replicas are made in such a way that it reduces the distance a request has to travel along the ring, and so, as a result reduces the

real distance that this request needs to cover. A detailed discussion of this protocol follows.

II Popularity of Documents

In any system where a few of the documents tend to be extremely popular, the nodes that store those documents will, over the course of time, have to bear a large portion of the requests for documents in that system. If, in addition, these requests are arriving at a rapid rate, it may lead to a node that stores popular documents to be overwhelmed and break down. One of the obvious way of alleviating the load on these possibly overloaded nodes is to make replicas of the popular documents that are stored on them on other nodes, and then use these replicas to serve requests for those documents.

Having decided to make replicas, the next question that needs to be answered is *where* these replicas should be placed. The placement of these replicas should be done sensibly i.e. in a manner that these replicas can be used to serve requests for the original document for a significant proportion of requests for that document. One good heuristic is to make replicas on nodes that lie along the path leading up to the successor of a popular document. Replicas created on nodes that precede a possibly overloaded node will lead to most of the requests for that document to be satisfied before that node itself is reached. This will in turn reduce the load on the node that has the original document. Also, the processing of a request that is serviced using a replica will be done efficiently. This is because of the fact that this request has been served either by its successor or by some node preceding the successor.

III Replicas on the Spatial Ring

We have established that it is beneficial to place the replicas along the path to the successor of the document. We now introduce a case for utilizing the spatially aware ring for storing replicas of popular documents.

In the system that we describe earlier, we first attempt to service a request for a document off the spatial ring. If the successor of the document in the spatial ring does not have that document, this request is forwarded to the plain ring. This request will now be serviced by its successor in the plain ring. Results showed that for a set of requests, if a sufficient number were serviced by nodes in the spatial ring, there is a significant improvement in the performance of the system. This is achieved in the presence of controls that ensure that the load on a particular node does not exceed the average load throughout the system by too much. The improvement in the performance is due to the shorter distances traveled along the spatially aware ring to reach the successor of the the document being looked up.

If replicas are placed on the spatially aware ring, along the path to to the successor of a document in that ring, intuitively this should result in a significant performance improvement for requests serviced by the spatial ring.

The complete protocol used to decide when to make replicas, where to place them and finally when to serve a request for a replica stored on a node is described below. Following that is a description of all the experiments we carried out using this protocol and the analysis of the results, which support the claims we have made to a large extent.

IV The Protocol

There is a "replication protocol" that is run periodically at every node. This protocol decides

- which documents need to be replicated at this node
- of the documents replicated at this node, requests for which of those will actually be served by this node

These calculations are made taking into consideration the number of requests served off this node, and how many requests for a particular document have been forwarded by this node, on the spatial ring.

The exact protocol is described below.

Each node keeps track of

- the documents for which it forwards requests in the spatial ring, and the number of times it does so for each document
- the number of requests it serves per document, on the plain and spatial ring
- the estimated average load for all the nodes in the system (this is used by the current node to decide whether it is overloaded itself and therefore if it can afford to serve requests from its cache and to what extent).

Initially, the caches of the documents are empty i.e. no requests are served by the spatial ring. The replication protocol is then periodically run at every node, by executing the following steps:

1. The document cache is deactivated i.e. all the documents currently in the cache are marked as not active. This means that this node will not be serving requests for this document from its cache
2. A candidate set which is a set of documents this node should cache, is formed. This set consists of all documents for which it forwards requests on the spatial ring as well as the set of documents it is currently caching.

3. This complete set of documents is sorted on the basis of the number of times it was accessed in the spatial ring (either forwarded or served).

4. A threshold that is based on the estimated average load in the system and the number of requests this node services on the plain ring, is calculated. This gives us an estimate of the number of requests this node can service on the spatial ring, without becoming overloaded.

5. A linear scan of this set is performed and the nodes are either activated from the current cache or explicit replicas created and added to the cache. This is done till the excess processing capacity of the node is exhausted. Since sometimes the threshold allows replication of a large number of documents, this creates an unnecessarily large number of replicas and increases the load on the system. To avoid this from happening, we limit the number of replicas being created explicitly in the following manner.

   (a) Each time the replication protocol is run at a node, there can be a maximum of m replicas on that node. This number includes replicas made using explicit calls and also the ones that are *activated* while running the replication protocol.

   (b) Only the first n ($n < m$) entries from the candidate set are considered for making replicas explicitly. This technique ensures that spurious replicas do not get created, once the replica set on a node has more or less settled down. Replicas from the current cache may continue to be activated as long as the processing capacity of the node is not exhausted and the limit on the total number of replicas on the node, m is not reached..

   For example in a system with $m = 100$ and $n = 10$ the working of the replication protocol at a node will be as follows. An explicit call to make a new replica

will be made only if that document is among the top 10 documents from the candidate set. Replicas may be activated from among those that are already being stored on this node as long as the node decides it can serve requests for that document and the limit for the total number of replicas that can be used to serve requests off that node, which is 100, has not been passed. Documents created are not deleted, since we assume that a node can store any number of documents.

V Controlling the Replication

As is suggested by the earlier discussion, replication of documents will be used to improve performance of the system and also help amortize the load due to popular documents. In the previous chapter we mention a heuristic that will create replicas of documents along the path to the successor. The placement of these replicas is decided entirely by requests for the original document. The intuition behind the placement of the replicas is very simple: if a node forwards a lot of requests for a document and it is lightly loaded, it will attempt to create a replica of this document.

This replication mechanism needs to be regulated. Failure to do so will lead to indiscriminate creation of replicas all over the system. The worst possible scenario is that, given sufficient time, there will be a replica of each document in the system on every node.

Specifically there are two parameters that need to be controlled to regulate the replication of documents in our system.

Cache Size (γ): Every node has limited storage. In addition to the documents that are already placed on a node we now are adding replicas to this node. The storage that is allocated to the replicas needs to be limited, in order to ensure that

the size of the cache does not grow without bound.

Number of replicas per document (β): A popular document will be requested throughout the system and requested for very frequently. Given the scheme we discuss in the previous chapter, this may lead to replicas of such documents to be created everywhere within the system. Although our system is a read-only, for it to be scalable to a system that allows both reads and writes to the documents it holds, we need to control the number of replicas we create for each document.

Consider the following:

The total number of nodes in the system $= n$

The total number of documents in the system $= m$

The cache size (documents/node) $= \gamma$

The maximum number of replicas per document $= \beta$

Therefore, the total size of the cache in the system is $\gamma * n$ and the total number of replicas that can be created is $\beta * m$. In order to to ensure that the system will have storage to hold all the replicas created while the system is in operation,

$$\gamma * n >= \beta * m$$

To ensure that the performance improvement is significant to justify the new system most (at least 80%) of the requests for docuemnts must be answered from the cache. Therefore the cahe hit ratio, $r = 0.8$.

This gives us an indication of the minimum total cache size that is required to achieve the required hit-rate.

$$\gamma * n = r * \beta * m$$

Now, not all the documents stored in the system will be requested for with the same probability. Some documents that are popular will be requested for very

frequently and some others will not be requested for at all. Since the replica creation is a direct result of the requests for documents in the system, not all documents will need up to β replicas. We can reasonably assume that the requests in our system follow the 80-20 rule. According to this rule, 20% of the documents are popular and are requested for frequently and the remaining 80% are rarely queried. It would therefore be sufficient to create only a small number of replicas for the unpopular documents and allow the popular documents be replicated to up to the maximum possible limit designated for the number of replicas per document. Here we assume unpopular documents do not need more than 2 replicas.

Therefore, the new total number of replicas in the system $= m*(2*0.8+\beta*0.2)$ and, the new cache size that needs to be maintained, $\gamma*n = r*m*(2*0.8+\beta*0.2)$.

For a system with 500 nodes and a 1000 documents: to achieve our target hit-rate of 80%, and allowing a maximum of 15 replicas per document the average size of the cache at each node should be about 8 documents per node.

We perform experiments with the values of its parameters in the vicinity of those mentioned above and discuss the results in the next chapter.

Chapter 7

Experiments

I Experimental Setup

We ran a series of experiments, to understand the behavior of the system with two rings and the algorithms for replication. The experiments were run using a Java simulation, which we developed.

For all our experiments, the nodes in the system are stationary and are assumed to reside in a square with dimensions of 5000×5000. There are 500 such nodes in the system. We distribute 1000 documents among these 500 nodes. We then generate a trace of 50000 requests for these documents.

The coordinates are generated with the help of BRITE [**?**, **?**], which is a universal topology generator.

A. BRITE

BRITE (Boston University Representative Internet topology generator) is one of the more popular topology generators which is used in simulations. The main reason of using BRITE as was because of its stated design principle of *representativeness*: the topologies generated accurately reflect many aspects of actual Internet topology. We use the BRITE system to generate a topology for our system (using the system

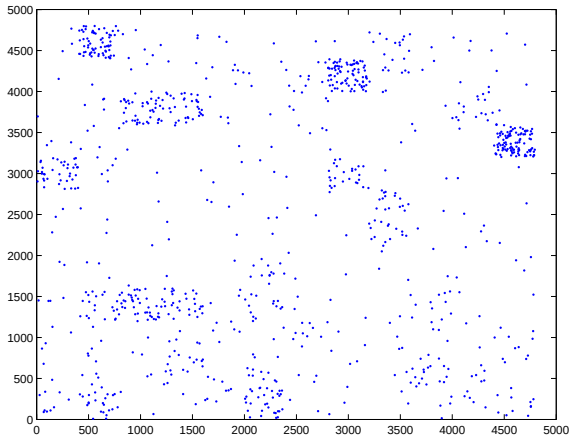

FIG. 7.1. The distribution of all the nodes in a plane

parameters stated above) and also derive the exact co-ordinates for the position of each node.

This location information is used to generate the identifier for this node in the spatial ring. As mentioned earlier, this transformation is done using bit-interleaving which is one of the techniques that can be used to map the two dimensional coordinates to a scalar value.

The physical distance traveled by a request or reply packet, which is the principal criteria of judging the performance of the system is calculated by adding the Euclidean distance between the two nodes at the endpoints of each hop, for that packet. The actual positions of all the nodes on a plane is shown in Figure 7.1.

B. **The Trace**

In this section, we have a small generalized description of the trace we use for all our simulations. To start the simulation, a series of join events are executed, for nodes joining the system. Each node that joins the system has two identifiers - one for each ring it participates in. All join events have to be initiated at some node,

and this node is picked randomly from the set of nodes that already exist within the system.

Next, events to insert documents into the system are generated. The document identifiers are generated randomly, and are in the same range as the node identifiers. The node that initiates the insert request is picked randomly from among the nodes existing in the system. No replicas of the documents are made at the time of their insertion. The insert request for a document is routed to the *successor* of that document in the plain (consistently hashed ring). Replicas of documents are created later, as need for them arises.

Finally, events that represent requests (queries) for documents in the system are generated. Again, the nodes where these look-ups originate are randomly chosen from among the nodes in the system. The document identifiers, for the documents that need to be looked up are generated such that they match a popularity distribution. This is to reflect the fact that some documents are more popular than others and so, in a fixed period of time, they will be accessed more frequently as compared to the rest of the documents.

C. Zipf-like Behavior of Requests

Zipf's law applied to web-documents can be stated as follows: The relative probability of a request for the i^{th} most popular document is proportional to $1/i$. In [**?**] one of the issues addressed by Breslau et al is the applicability of Zipf's law to web requests. Using six traces from proxies at academic institutions, ISPs and corporations they conclude that the distribution of page requests generally follow a Zipf like distribution where the relative probability of a request for the i^{th} most popular document is proportional to $1/i^{\alpha}$, and α is a value less than unity. That is, the request distribution does not follow the strict Zipf's law (for which $\alpha = 1$) but a zipf-like

Case	Performance - Mean values		Load		
	Dist/Req	Hops/Req	Avg	Median	Max
Plain Chord	12044	4.5	100	59	1372
Spatial Chord	5605	5.03	100	0	4286

Table 7.1. Comparison of performance and load for plain and spatial Chord

distribution with varying α. The values for α for the traces analyzed by them ranged from 0.64 to 0.83. We assume a popularity distribution similar to that of web documents for the documents in our peer to peer system. For our experiments, we use a variation of Zipf's law with an α value of 0.72 to generate traces for our simulations.

II Results

The first set of results (Table 7.1) point to the initial experiments we conducted to explore the notion of using geographical information within Chord. Here we have a single ring where identifiers for a node are computed using bit interleaving on the coordinates of that node in the real network. We compare performance and load results of this system with the original Chord system by observing these properties for the same trace of events.

The results show a considerable improvement in the performance. The mean latency (in terms of distance covered per request) of a request in the system which uses spatial information is less than half that of plain Chord. In the above system there are no measures taken to control the load on any node in the system, nor are any load-balancing techniques in place. As a result a some of the nodes have to bear loads that far exceed the average load in the system. Also, most of the nodes do not have to respond to single request (since they do not store any documents).

Nevertheless the results clearly point to the fact that using location information to speed up Chord has a lot of potential. If we can retain most of the performance gain while introducing better load balancing and fault tolerance mechanisms, this

Case	Performance (Mean)		Load			Replicas Created		
	Dist/Req	Hops/Req	Avg	Med	Max	Avg	Med	Max
Plain Chord	12044	4.5	100	59	1372	-	-	-
Two Rings - single replica	11640	7.2	100	22	2427	1	1	1
Two Rings - multiple replicas	7055	4.1	100	51	923	5.7	5	16

Table 7.2. Comparison of performance, load and replication overhead for the different cases

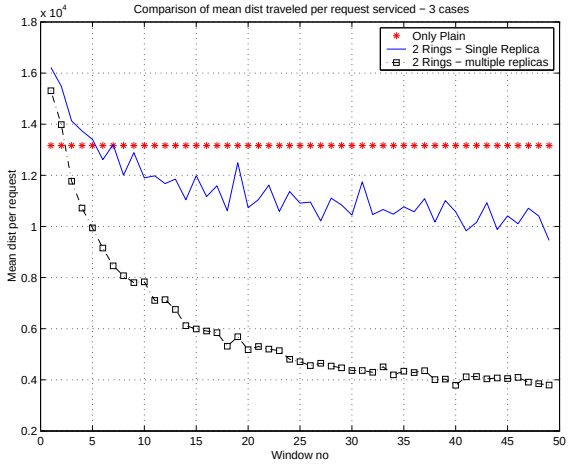

FIG. 7.2. Comparison of the distance traveled per request, across the 3 systems

technique might well be worth the trouble of implementing it.

Table 7.2 shows the performance of our system as compared to the original Chord system. The results are as expected. The systems with two rings out-perform the plain chord system comprehensively. Among the systems with two rings, the system where we aggressively create replicas in the spatial ring has a far better performance than the system where only one replica exists on the successor of the document in the spatial ring. This is because, as replicas of popular documents get created along the path to the successor of the document, the distance that needs to be covered and also the number of hops that need to be taken before a request is answered will decrease. The newly created replicas will be used to answer some of the queries for

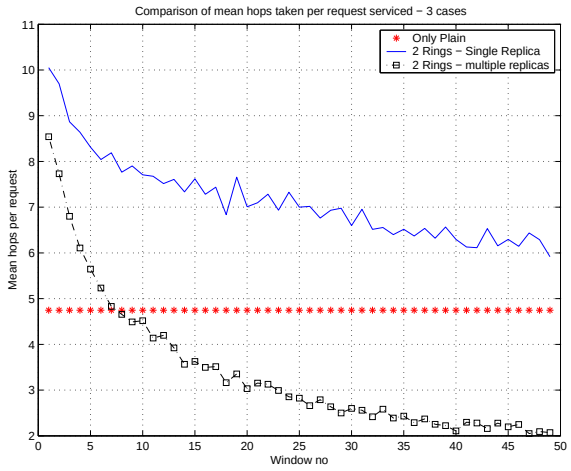

FIG. 7.3. Comparison of the hops taken per request, across the 3 systems

the documents. Replication of documents in the spatial ring also affords a better distribution of the load in the system. Because of replication, the successor of a document is not the only node capable of answering a request for a document. All the nodes that store a replica for a document share the load for that document.

To get a better idea of the performance of the system, consider Figures 7.2 and 7.3.

Here we see the performance of the system in terms of mean hops per request and mean distance traveled per request, over time periods of 1000 requests. In the systems with 2 rings, initially they start out with performance worse than plain Chord. This is inevitable because in the beginning, there are no replicas in the spatial ring and all requests have to traverse all the way to the successor of the document in the plain ring in order to be served. As time progresses the systems with 2 rings consistently out perform plain chord. As more replicas get created in the spatial ring and more requests get serviced from the same, the performance of the system improves considerably. A similar effect is observed with the hops per request metric.

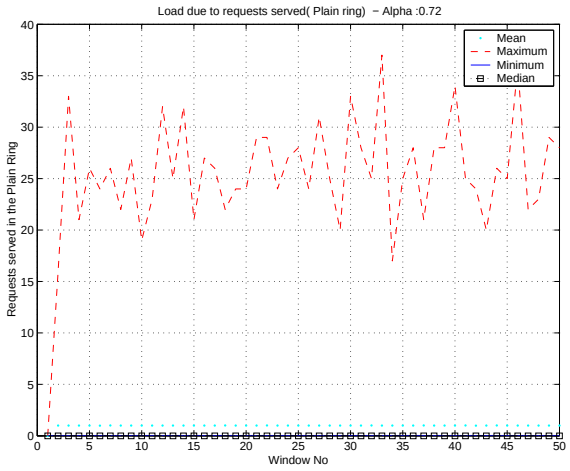

FIG. 7.4. Load in the system for plain Chord

Also worth mentioning in both the above results is that the system with multiple replicas has a much better performance when compared to the system where we just place a single replica of a document in the spatial ring.

The behavior of the system with respect to the load is shown in Fig 7.4, Fig 7.5 and Fig 7.6. The figures show the mean, median, maximum and minimum values for requests serviced (load) by a node, measured within windows which have a period of 1000 requests. The load for the systems with 2 rings includes the load due to the creation of replicas. From these figures we can see that the load on the nodes in the system, even in the configurations where we are using the spatially aware system, is not too different from the behavior of the original Chord system. But the load balancing in the system with multiple replicas of a document in the spatial ring is better than the system with just a single replica, because in the latter case, the load is distributed among the replicas of a document.

The performance of the replication protocol directly depends on the quality of the replicas created. That is, if replicas of popular documents are created and these

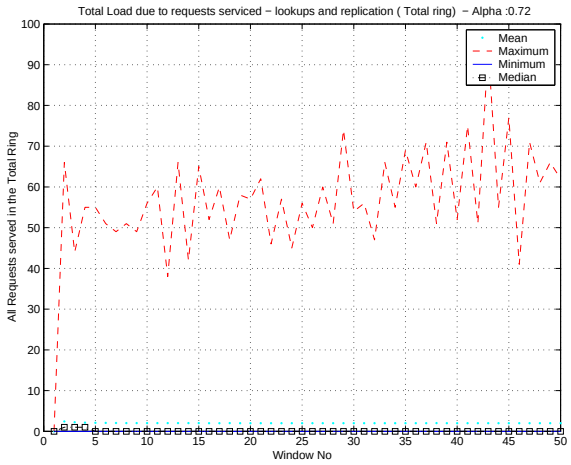

FIG. 7.5. Load in the system for Chord with 2 rings, single replica

documents are utilized in serving requests, it will lead to better performance and load balancing. If the replicas created are those of documents that are not looked up very often, this will render these replicas useless and just contribute to increasing the load on the other nodes in the system.

To be able to make replicas of documents that will be used the replication protocol has to *learn* the popularity distribution of the documents. To get a better understanding of this aspect, the experiments were run for different document counts, from 1000 to 10000 in steps of 1000, keeping the cache size the same and the replication period the same. As the document count is increased, the extent to which the replication protocol is able to learn the popularity distribution decreases. This is shown in Figure 7.7 which shows how the performance of the system deteriorates as the number of documents in the system is increased. This can be explained as follows. Each node in the system runs the replication protocol independently of other nodes and decides for itself which documents it should replicate. These decisions are based on how many requests for documents have passed through this node. To be

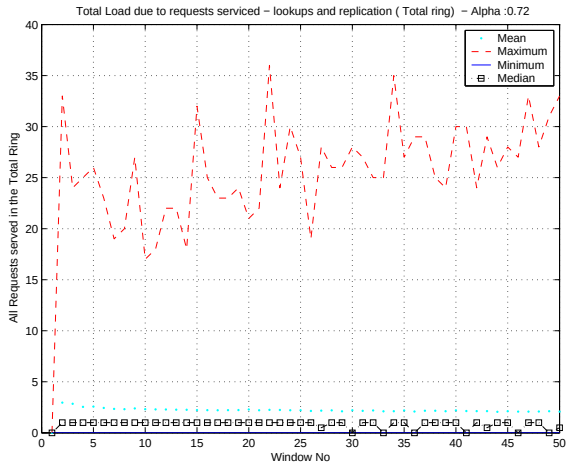

FIG. 7.6. Load in the system for Chord with 2 rings, multiple replicas, maximum 15 replicas per document, maximum 20 new replicas created on a node each time replication protocol is run

able to make a good decision, requests should be generated for a sufficient number of documents in the system. For example, for $\alpha = 0.72$, experiments showed that if there are n documents in the system, approximately n requests go by, before half of the documents have been requested for at least once.

The number of replicas created are a direct measure of the overhead in the system, and so it is desirable to keep this number at a minimum. This means that whenever a replica is created we need to be sure that it is a useful replica, and its being created will serve its purpose of reducing the load on the original document. When the system is being bootstrapped, documents are inserted and initial requests generated, there will be a large number of replicas created i.e. each node will make as many replicas as it is allowed to. But then, the system should settle down to a so called *steady state*, in which the replica cache of each of its nodes is more or less stationary, and there are very few requests to create new replica. This is the expected behavior, if the protocol is performing well. This is illustrated in Fig 7.8, which shows

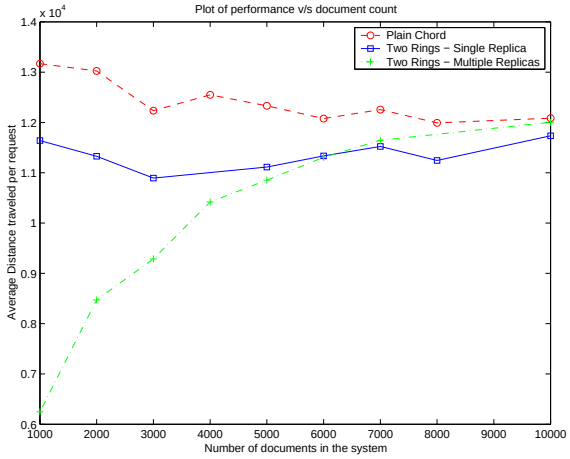

FIG. 7.7. The performance of the system deteriorates as document count is increased, if replication period is kept the same

the number of replicas created at each instance of the replication protocol, for the system with 1000 documents. Fig 7.9 illustrates the other case where there are 10000 documents in the system. Here, the number of replicas created at each iteration of the replication protocol does not decrease considerably throughout the simulation. This is because since the replication period is short, the number of requests generated for each document are not enough to judge its popularity.

The number of requests serviced by each node in the system, at the end of the simulation gives a fairly good indication about the distribution of the load in the system. Figures 7.10, 7.11 and 7.12 below show the histogram of the number of requests served by each node for the plain Chord system and the systems with 2 rings, one where the spatial ring has only one replica per document and the other in which there can be multiple replicas for each document.

The distribution is skewed because of the popularity of the documents makes requests for some documents much more frequent than the requests for some other documents. This will obviously mean that nodes that store these popular documents

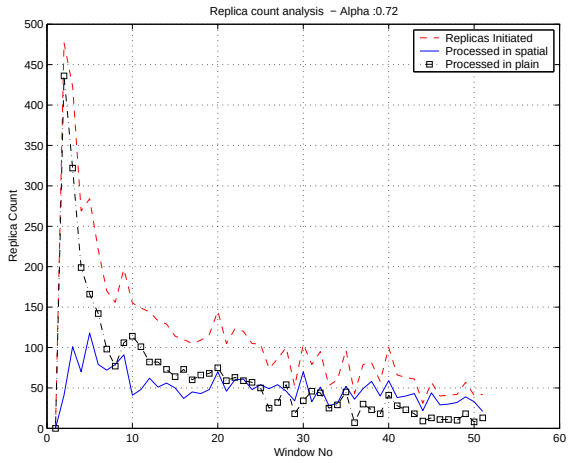

FIG. 7.8. The number of replicas created settles down when the system reaches a *steady state* - 1000 documents, 2 rings, multiple replicas

will be have to bear more of the load. In the systems with 2 rings, the one in which we allow multiple replicas achieves a better distribution of the load as compared to the on where there is just a single replica.

In the earlier chapter we had a small discussion on techniques for extracting a good performance from the system while keeping the load on the nodes in the system under control by controlling the replication. The replication is controlled by limiting the number of replicas per document and also total size of the cache. We had concluded that to make the extra effort expended in maintaining two rings and the replicas we would need a hit-rate of at least 80%. For our system it meant that for 15 replicas per document the size of the cache should be about 8 documents per node. We ran some experiments varying values for the total number of replicas per node and the size of the cache at a single node from 5 to 20 in steps of 5. The results are shown in the form of 3 dimensional graphs in the figures 7.13 and 7.14. The best performance is observed for the system with 20 replicas per document and cache size of 15 documents per node. The system with cache size of 15 documents per node and

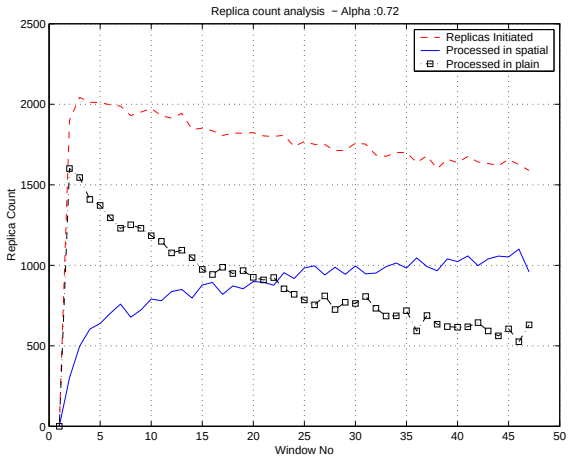

FIG. 7.9. The number of replicas created stays high throughout the simulation - 10000 documents, 2 rings, multiple replicas

a maximum of 20 replicas per document has the least load.The performance of this system is just marginally lesser than the best observed performance in the system. So, we use the system with these parameters (20 replicas per document, 15 replicas per node) as the representative system for comparisons with the plain chord system and the system where there is at most a single replica of all documents in the spatial ring.

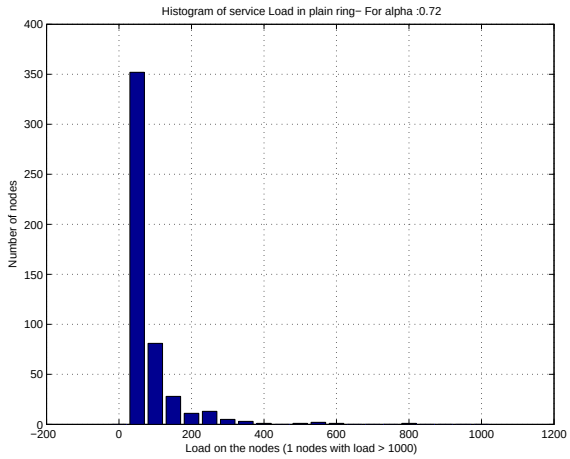

FIG. 7.10. Histogram for the load on each node, for the system with only the plain ring

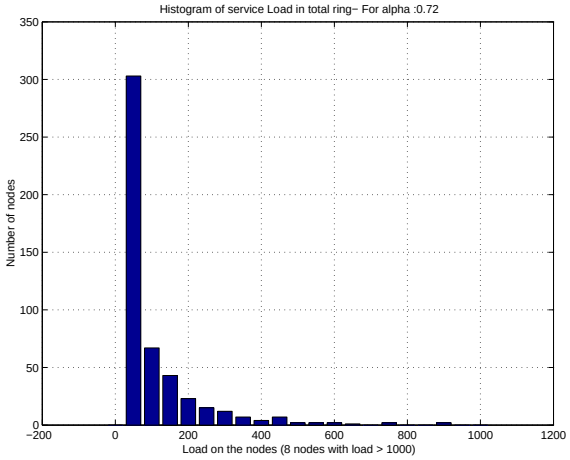

FIG. 7.11. Histogram for the load on each node, for the system with 2 rings and one replica per document

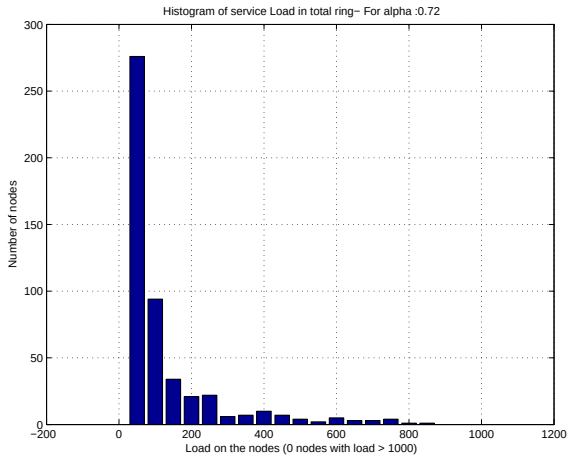

FIG. 7.12. Histogram for the load on each node, for the system with 2 rings and multiple replicas per document

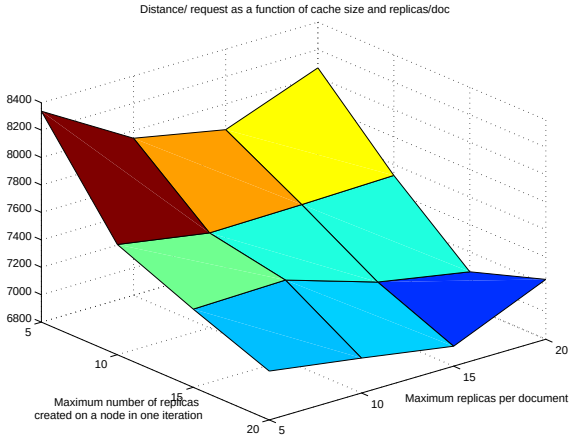

FIG. 7.13. Comparison of the performance of the system with two rings for different values of maximum number of replicas per document and number of replicas that can be created at a node at one time

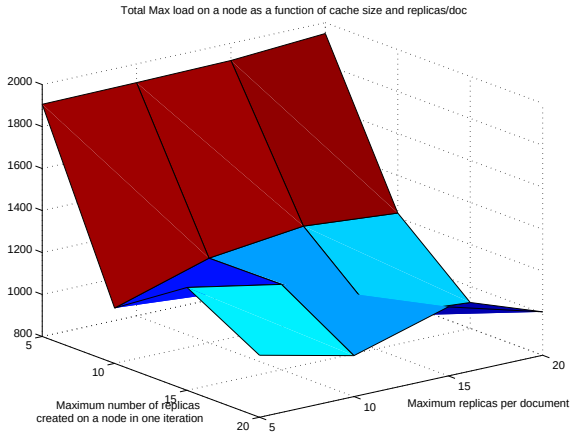

FIG. 7.14. Comparison of the load in the system with two rings for different values of maximum number of replicas per document and number of replicas that can be created at a node at one time

Chapter 8

Conclusion and Further Work

In this thesis we propose a novel architecture for a peer to peer system, which leverages information about the physical location of the nodes in the system to make the system more efficient. Our system is built on top of Chord, which is one of the architectures to build a completely virtual peer-to-peer system and is based on distributed hash tables. Using experiments based on simulations we show that our system can achieve a significant improvement in the performance of the system, which is measured as the distance traveled by a request in order to be satisfactorily serviced.

In our system we use two rings:one based on Chord and the other where the nodes are arranged along the ring based on their actual physical locations. The spatially arranged ring is used to accelerate the response times for popular documents and the plain chord ring is used as a safety net, which guarantees that all requests will be answered correctly. For analyzing the performance of the system we use a non-uniform distribution of requests, based on Zipf's law. Using replication along with the two ring system, we achieve a considerable improvement in performance and at the same time effectively control the load on the nodes in the system.

There are a number of areas in which this research can be extended. The first step would be to build an actual system and analyze the effectiveness of the protocols

that we propose. The measurements can be more detailed. The metric that we use to analyze the *distance* between two nodes can be more refined. Right now it is just the physical distance between the nodes. Information about the quality of the link between two nodes can be incorporated into the calculation of this metric. One more direction is to do more research is to get a more comprehensive analysis of the impact of replication on the two rings.

References

[1] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Symposium on Operating Systems Principles*, pages 131–145, 2001.

[2] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM (1)*, pages 126–134, 1999.

[3] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting network proximity in distributed hash tables, 2002.

[4] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Symposium on Operating Systems Principles*, pages 202–215, 2001.

[5] Gnutella.

[6] Kazaa.

[7] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[8] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: Universal topology generation from a user's perspective. Technical Report 2001-003, 1 2001.

[9] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in internet topologies. Technical Report 2000-004, 20, 2000.

[10] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building p2p networks with good topological properties.

[11] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter p2p networks. In *IEEE Symposium on Foundations of Computer Science*, pages 492–499, 2001.

[12] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.

[13] Project iris.

[14] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

[15] Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. Routing algorithms for dhts: Some open questions. 2002.

[16] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–??, 2001.

[17] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-topeer systems, 2003.

[18] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. pages 149–160.

[19] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.