An Architecture to build Spatially Aware Peer to Peer System[1,2]

Konstantinos Kalpakis, Koustuv Dasgupta and Prashant Punjabi

Abstract

Peer to peer networks 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 the most effective mechanism to implement such systems. There are several protocols to implement a peer to peer system as a distribted 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 orientation or 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 will considerably reduce the total distance travelled 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 with using Chord as a base protocol and we suggest changes to make the system spatially aware.

We then go on to analyse our scheme thoroughly, using a simulation. We address all issues that may arise because of the changes we describe. We provide detailed experimental results demonstrating that our algorithm can offer significant improvements in system performance, when compared with sytems that do not incorporate spatial information and are completely virtual. And all of the above is achieved without introducing unnecessary load in the system.

I Introduction

The peer to peer paradigm is 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. The numerous advantages of such systems such as scalability, robustness and the lack of a need for administration have resulted in them being used for numerous applications such as distributed storage, content delivery etc.

There are several issues that need to be considered while designing an architecture for a peer to peer system. The principal among these are how a node wishing to participate in the system actually join the system, how does it learn about other nodes that are already present in the network, how to decide which nodes to connect to, which nodes to contact first in case of query and last but not the least, how to search the peer to peer network for something. This last point is easily the most common operation in a public peer to peer network such as Gnutella or Napster [4, 5]. In such networks users typically share resources such as documents, images, media files. A user that needs some resource can get the same from some other peer in the network that has the said resource. This obviously involves a search for the resource throughout the network of peers. The search in Napster is centralized and so carries with is the baggage of such an

[1]Supported in part by NASA under Cooperative Agreement NCC5–315 and Contract NAS5-32337.

[2]Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250. E-mail: {kalpakis,dasgupta,pras1}@csee.umbc.edu. Phone: 410-455-3143. Fax: 410-455-3969. **Please send all correspondence to Prof. Kalpakis.**

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 comprmises the result of the search. [7, 6] describe how a Gnutella type network can be constructed with a low diameter. Also they go on to describe a protocol that ensures the degree of any node in the system to within constant bounds and so the lower and upper bound on the connectivity of the network.

So far the more successful attempts to provide an architecture for peer to peer systems have focussed on implementing them as *distributed hash tables*. In this paradigm for building peer to peer 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 [8], Chord [10] and Tapestry [11].

Terradir [2] is one architecture that attempts to create the connections in the peer to peer network as to reflect the way the data is arranged. This according to them facilitates query routing. They also argue that virtualized overlay networks do not take advantage of locality in the manner in which data is accessed. For example, if the data that is accessed by a particular node is on the same remote node, the peer to peer systems will at most store the route to that remote node. There will be no prefetching of data due to spatial locality that would occur if these two nodes were part of a file system in an enterprise network. They also mention that the nodes do not do any caching of the data, but this can be easily incorporated into the existing architectures.

With the wide scale use of peer to peer applications, increasing portions of network traffic generated will be due to the use of such applications. Obviously, there is a need to minimize such traffic. And because peer to peer applications are typically implemented on top of an overlay network [1], 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. [6, 7] 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 logarthmically with the size of the network.

In the above aproaches, the nodes that one node connects to has nothing to do with "where" those nodes are. This means that for a public peer to peer network that uses the Internet, for example, there can be nodes that are "adjacent" to each other according to the overlay network which connects all the peers in the system, which, in fact may be really distant from each other as far as the true network distance is concerned. This will lead to inefficiency in the sense that if every query proceeds by contacting a set of "neighbors", and then fanning out from there, these neighbors may take a long time to get the request and to finally reply. Consider a situation where the documents are replicated, it might be beneficial for the system performance to always manage to find and return the closest copy, closest in terms of real network distance. This introduces a case for proximity awareness in the method for constructing a peer to peer network.

Our contribution in this paper is a scheme to incorporate proximity information in Chord, which is one of the architectures to build peer to peer 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 also describe in detail the mechanism we employ to counter the loss of uniform distribution of node identifiers over the identifier space.

The rest of this paper is organized as follows. ...

II Motivation

A. Introduction to Chord

Chord [10] presents a distributed look up scheme that can be the backbone of a peer to peer system. It is a scheme based on hashing where each node and each instance of a resource in the system is assigned an identifier which is arrived at using *consistent hashing.* A resource is stored at a node which is called its *successor.* A successor of an object is that node whose identifier value is greater than or equal to the identifier value of the object, and is closest to it than that of any other node. The nodes are visualized as arranged along the circumference of a ring. Each node has a pointer to its *successor.* A successor of a node is the node whose identifier value is least among all nodes whose identifier values are greater than its own identifier value. Also each node maintains a list of pointers into the ring called *fingers.* These are pointers to successors of nodes that have identifier values that are more than the identifier value of the current node that are successive powers of 2. These successor pointers and fingers are used for routing a query for a resource, given its identifier, and to find successors of other nodes in the system. Chord is therefore a relatively simple scheme that has reasonably good performance in terms of hops taken to satisfy a query, which is $O(log(N))$, where N is the number of possible nodes in the system. The protocols defined for its maintainance ensure a mostly error free operation and reasonably good performance for the system.

B. 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 hashing which generates a virtual identifier.

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 where queries (requests) 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 travelled by the request packet is of the order of $D * logN$. The figure below shows clearly the benefit possible if the nodes are arranged along the ring in a spatially-aware fashion. The total distance travelled by a request will be proportional to the total distance travelled by the request along the ring, which on average will be of the order of D. This will definitely lead to an improvement as far as the total distance travelled by the request packet is concerned.

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

III Design

In the system we are proposing, the positions of the peer nodes along the circumference of the Chord ring are dependant on their actual physical locations. Consequently, the distance between two nodes along the ring is indicative of the real distance between the two peers in Euclidean space. The placement of nodes

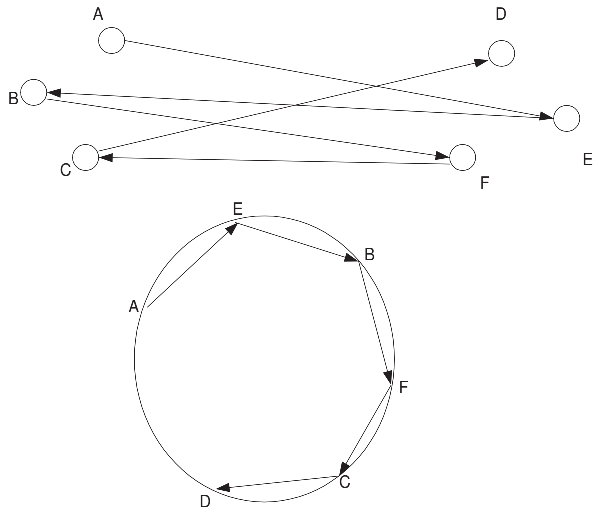

Figure 1: PLAIN RING

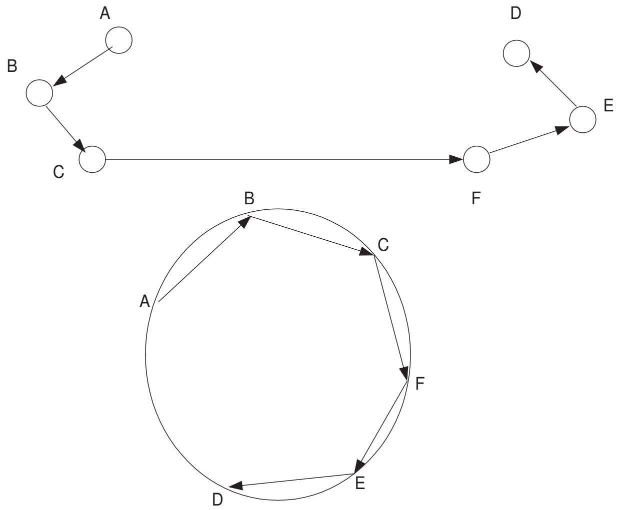

Figure 2: SPATIAL RING

along the circumference of the ring is done according to the identifier value for that node. Therefore, any method to make the placement of the nodes on the ring according to the physical locations of the nodes would have to incorporate this location information into the identifier values for these nodes.

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. 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. The Problem

But this approach by itself will not be enough for the system to be complete. 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. 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 [9]. Thus, there will have to be some controls and load balancing 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*.

C. Solutions based on Hashing: Collission-avoidance

Since Chord is based on a hashing scheme, the collission avoidance/ resloution schemes used normally in hash functions would seem to offer promising solutions. But this is not the case. One would imagine schemes such as double hashing, or quadractic 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 collissions during hashing. And if a query arrives at a node, which has moved some documents off to some other node, it will have to find a way to forward the request to the node that actually stores that document.

But these techniques give rise to a host of other problems. The first among these being 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 colission 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 impossible 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 reasonaly good balance of the load becaues if its taking advantage of the consistent hashing function to generate node and document identifiers.

Also this scheme does not accomodate the presence of an extremely popular document on a node. The rate of requests for this doucment at a single node may be sufficient to overload that node.

IV Two Rings

The problem with the spatial appoach 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 loactions 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 which translates into better performance and fewer hops in the underlying network which reduces the load on the same.

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

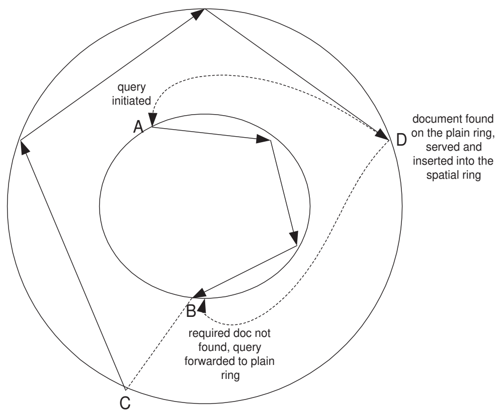

Figure 3: A *find* query switching rings, because of a *miss* in the plain ring

manner.

Instead of having a system with just one ring with node identifiers that are generated using a consistent hashing, we have that one ring and another that has the nodes arranged according to their actual locations, by using indetifiers that are spatially aware.

Every node in the system is a part of these two 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 and the other identifier is the one that will be used in the proximity-aware ring and is generated by using the actual location of the node. Each node will therefore have to maintain complete local information ie successor pointers, predecessor pointers and finger table and finger cache for 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.

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 to their new successors in both the rings and updating the successor and predecessor pointers. Then the set of documents to be retained at the new successor of the documents in the spatially arranged ring is decided using the algorithm described in the next section.

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 the same. If that document is not present on the node that will be its successor on this 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. If an access for a document is served from the plain Chord ring, that document is copied to the spatially arranged Chord ring. Using this strategy we make copies of the documents already in the plain chord ring in the ring that is spatially aware.

Data access patterns suggest that a very small portion of the data is accessed very frequently and most of the data is accessed very rarely. We imagine (with sufficient experimental evidence based on simulations [?]) that if we manage to keep the frequently accessed documents in the optimised spatially aware ring, there will be sufficient imrovement in the performance for most queries to overcome the occassional 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 queries.

If trends for accesses on documents stored in this system are based on *popularity of documents*, and most of the popular docuemnts have replicas in the spatially aware ring, there will be an improvement in the performance of the system, because most of the queries will now be served off the spatially aware component of the system. So basically the proximity-aware chord ring acts as a *cache* from where queries for documemnts can be served.

Now when the documents are moved to the spatially arranged ring, their presence on the node that they are headed for may cause that node to become overloaded. And so some controls must be introduced to limit the number of documents on a node in the spatially arranged ring.

Load balancing for the plain ring can be done as described in the original system, that is of having multiple virtual nodes per real node.

For a node in the spatially arranged ring, its view of the load in the system is the average load on all its fingers. This information is stored at each node and updated periodically by reading in the load seen by all the nodes at the fingers of the node. The load on a node is the load due to a request served for a document query.

V Document Replacement Algorithm

For the system that we are proposing to perform, popular documents need to be replicated on the spatial ring. At the same time, to keep the load at a node in the spatially aware ring about the same as the average load in the system it becomes necessary sometimes to deactivate some documents on a node in the spatially arranged ring.

Since this is a distributed system no single node can be expected to have a global view of the system to know the average system load. As in distributed system, each node gets the information to base its decisions on from other nodes that are reltively close to them. The decision making is therefore done in a *decentralized* manner and each node takes these decisions independently based on the local information available to it.

In the scheme we are proposing, each node decides for itself what the average system load is. Obviously, this cannot be done by polling *each* node for its load and taking the average load over all the nodes in the system. Each node therefore needs to arrive at a reasonable estimate of the load in the system. This is done by taking the average of the load on all the fingers of the node. This is a reasonable estimate because the fingers cover an appreciable part of the ring and so this estimate is better than one based on only local information from nodes physically close to that node. Also the accumulation of such information does not put too much extra strain on the system because the routing of messages that will carry the information about load is trivial because a node can directly communicate with the nodes at the other end of its fingers.

And so this load information is collected periodically at every node in the system and stored locally.

Since the aim is to maximize the performance of the system, we will attempt to keep those documents that have been accessed the most number of times, while deactivating those that have not been accessed as many times and at the same time the total load on the node due to the retained documents has to be within some range of the average system load.

The algorithm that is used to decide which documents will be retained and which documents will be deactivated from the node is shown in the above figure. This algorithm is executed periodically at every node. The documents at that node are sorted in the decreasing order of their access counts. Then only those documents are retained, which according to their access counts in the last period will not overload this node.

```
Procedure Pack(node p)
Add documents d_1 ... d_n to the set D, of existing documents
Sort D in descending order of f, the number of accesses for a document
Set accumulator A = 0
for each d_i ∈ D do
    if A + f_i ≤ T // T is the threshhold
        retain d_i
    else
        deactivate d_i
return
```

Figure 4: Pseudocode for the method used to decide which documents are placed on the spatial ring.

VI Replication

A. Need for repliaction

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 probablity. Some documents are bound to be more popular than others. This behaviour 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 section we describe the scheme in which spatial awareness of the nodes in the system is used to make replicas of popular documents that considerably reduces the distance that a request for one of those documents has to follow. This is done by making replicas of the popular documents on the spatially aware ring. These replicas are made along the path to the successor of the popular documents on the spatial ring. This reduces the hop count and so also considerably reduces the distance a request has to traverse in order to be satisfied. A detailed discussion of this protocol follows.

B. 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 doucuments to be overwhelmed and break down. The most obvious way of alleviating the load on these possibly overloaded nodes is to make replicas of the popular documents that are stored on them.

Now that we have decided to make replicas, the next question is "where" these replicas must be placed. The placement of these replicas should be done sensibly. One good heuristic is to make replicas along the path leading upto the successor of a popular document. A sufficient number of replicas created on nodes that precede an overloaded node will lead to most of the requests for that document to be satisfied before that node itself is reached. There is an automatic benefit of a request for the document having to travel a lessed distance on the way to the sucessor of the document it is seeking in order to be satisfied.

C. Replicas on the spatial ring

We have established that the best place for the replicas is along the path to the successor of the document. We now introduce a case for utilizing a spatially arranged ring for making the replicas of very popular documents.

In the system that we have described earlier, we first scan the spatial ring for a document. If that document is not found on the spatial ring, we forward the request for that document to the plain chord ring. Results showed that for a set of requests, if a sufficient number were served off the spatial ring, there is a considerable improvement in the performance of the system. This is 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, in which case some of the nodes in the system will be considered as overloaded and so risk breaking down.

The improved performance was achieved in via shorter distances travelled along the spatially aware ring to reach the successor of the the document being looked up, in that ring.

Similarly if replicas are placed on the spatially aware ring, along the path to to the successor of a document in that node, intuitively this should result in a significant performance imporvement for requests served off 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.

D. 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 of this node, if need be

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 (cache)
- the 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 ie no requests are served off the spatial ring.

The replication protocol is then run at that node.

- The document cache is dactivated ie all the documents currently in the cache are marked as not active which means that this node will not be serving requests for this document from its cache
- A candidate set ie a set of documents that this node can serve cache and serve requests from it, is formed. The set of all documents for which it forwards requests on the spatial ring as well as the set of documents it is currently caching.
- 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).

Then 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 threshhold allows replication of a large number of documents, this creates an unecessarily 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.

- There can be a maximum of n replicas created during the course of running the replication protocol at a node.
- Only the first m ($m >= n$) entries from the candidate set are considered for making replicas explicitly.

This technique ensures that spurious replicas do not get created.

VII Experiments

A. Experimental Setup

For running experiments to test and analyse the performance of our system, we are using the following setup. Number of nodes : 500 Number of documents : 10000 Number of queries : 50000

To model popularity of documents, we use the zipf distribution ot generate document identifiers for the queries in the system. We use the alpha value 0.72, which generates a distribution of requests that matches a scenario with web documents most closely.

The topology for the system was generated using BRITE. Here also a kind of heavy tailed distribution was used so as to reflect a real physical distribution oof the nodes in a plane.

Comparison	Distance per hop	Improvement
for Plain Chord	12004.6	
for $\alpha = 0.72, m = 5, n = 5$	9779.8	19%
for $\alpha = 0.72, m = 5, n = 10$	9599.2	20%
for $\alpha = 0.72, m = 3, n = 5$	10362.1	14%

Preliminary Results comparing performance

VIII Conclusion

IX Further Work

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] Bobby Bhattacharjee, Pete Keleher, and Bujor Silaghi. The design of terradir.

[3] 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.

[4] Gnutella.

[5] Napster.

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

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

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

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

[10] 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.

[11] 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.