DYNAMIC TREE-LIKE STRUCTURES IN P2P-NETWORKS

Herwig Unger
Markus Wulff
Department of Computer Science
University of Rostock
D-18051 Rostock, Germany
{hunger,mwulff}@informatik.uni-rostock.de

KEYWORDS

P2P, Search, Stochastic Algorithms, Dynamic Structures

ABSTRACT

The search in P2P systems is still a problem because of the dynamic nature of these systems and the lack of central catalogs. In order to improve the search in P2P systems, a sorted structure is created from the content of all nodes in the system. Different from other approaches, a tree like structure is built by tokens which are constantly moving between the nodes and carrying all structural information.

INTRODUCTION

Peer-to-Peer (P2P) systems are decentralized systems built by a set of peers forming a loosely coupled network through the knowledge of some other peers of the system kept in a local neighborhood warehouse. Such systems, mostly known from file sharing applications like Gnutella or KaZaa, propose to solve some problems of classical client/server systems due to the following reasons. In a P2P system, every peer (normally composed of a user and a certain machine represented by its IP address) can request and provide services. However, there is no central or hierarchical control which results in the following advantages of P2P systems:

- no node is burdened with the work of the entire system;
- the system can maximize its performance by aggregating the computation and storage resources of the participating nodes;
- the system can continue functioning correctly when failures occur.

Generally speaking, the functionality of every participant (node) is equivalent in such a completely decentralized P2P system. Since the presence of a centralized coordinator is eliminated, the problem of lookup can be encountered, which is stated by Balakrishnan et al.: "given a data item X stored at some dynamic set of nodes in the system, find it". This is a critically ordinary problem in P2P systems (Balakrishnan et al., 2003).

Based on the used lookup algorithms, the P2P systems are categorized into unstructured and structured systems. Unstructured P2P systems, such as Gnutella (Gnutella, 2004), KaZaa (Kazaa, 2004) and Freenet (Clarke et al., 2000), define no constraint of connection between nodes. In general, their lookup algorithms are simple (and mostly based on an expensive broadcast of the search queries) and the network's dynamism involves a low cost of maintenance. However, the lookup algorithms of these systems are non deterministic, i. e., lookup for a resource may fail even if the resource exists somewhere in the network. Structured P2P systems, also called Distributed Hash Table (DHT) systems, define constraints of connection between nodes. Such systems can be regarded as a hash table distributed on the network's nodes, each of which is responsible for a chunk of the table. Examples of DHT systems are Tapestry (Zhao et al., 2001), Chord (Stoica et al., 2001), Pastry (Rowstron and Druschel, 2001), CAN (Ratnasamy et al., 2001), Viceroy (Malkhi et al., 2002), Koorde (Kaashoek and Karger, 2003). They provide efficient and deterministic lookup algorithms that locate a certain resource in a small number of routing hops (e. g., O(log N) where N is the number of live nodes in the network). However, DHT systems require a high cost to maintain their consistent structure in dynamic environments.

IDEA FOR A NEW ALGORITHM

Nevertheless, real P2P systems are not static: nodes are frequently added and removed from the system (mostly according to users interest). Therefore, unstructured systems need cost-intensive *Ping-Pong*-mechanism to ensure the connectivity of the system, which will generate even more network load than the expensive broadcast based search and reduce the scalability of the system once more. Structured P2P-systems need a relatively complex recovering algorithm and time to be adapted to such changes and to rebuild the respective tree-like overlay structures. The frequent inclusion of new nodes also seems to be problematic in case the system is very big (No publication about these systems has considered more than 10000 nodes.). Finally, it must be noticed that there

Proceedings 12th International Conference ASMTA 2005
Khalid Al-Begain, Gunter Bolch, Miklos Telek © ECMS, 2005
ISBN 1-84233-112-4 (Set) / ISBN 1-84233-113-2 (CD)

are frequently some key nodes in the system which play a more important role to keep the DHT structure.

The idea of the current contribution is to replace fixed peer based tree structures with mobile tokens running through the network. These tokens keep some sensitive structural information. Each of these tokens is generated and controlled by a smaller group of nodes in the system where the data and interests of these nodes are represented by this token. The amount of the data transported by the token is limited and the token can be recovered as long as one node of the group is still in the system. Therefore, the fault tolerance can be significantly increased compared with classical DHT systems.

It will be shown that the tokens can be connected in an intelligent manner to build a flexible, adaptive and tree-like structure on top of a given P2P community structure. This structure will support a classically sorted tree-like search and therefore any key may be found during a fixed time without the need of a search broadcast.

To explain the idea above, the article first describes the algorithm. Then, the necessary adaptations to run the algorithm in a dynamic environment where nodes frequently join and leave the system are discussed. Finally, the achieved simulation results for the static as well as for simulated dynamic small world environment are presented.

THE TOKEN BASED LOCAL ALGORITHM

Basic Principles

Since the P2P systems are decentrally organized, there is no centralized control which tells peers what to do. Therefore, each peer can only execute a local algorithm by using the locally-stored information. Any cooperation needs and possibilities will represent itself in global, self-appearing structures from that activities.

For simplicity, it should be assumed that each peer carry one data set described by one key item. (In many other file sharing systems, the data (files) are represented by its file name which may consist of several main keywords that are important for the search.) Each node initiates messages (tokens) corresponding to its key (content), representing that data-peer issue in the community. Then this token is sent as a random walker to one of its neighbours while the node itself is waiting for processing incoming messages (in an infinite loop).

The main rules in the token processing behaviour should be:

1. Each token must periodically visit all peers whose data are beeing represented by it.
2. Each token may represent a certain number of peers which are part of its plan to describe the elements of its (periodically repeated) visits.
3. The token will be forwarded randomly if the current position of the plan is free.
4. Meeting tokens may be merged (if they meet special criterion) and then represent the group of nodes they are derived from.
5. Stable tokens, i. e., tokens which have not changed for a longer time, represent more than one data-peer issue. These tokens may generate a new token of a higher level as representative, playing with tokens of its level after the same rules.
6. Tokens of different levels may communicate with each other by using commonly visited nodes as their communication places.

Due to the last rule, a tree-like structure appears, in which the nodes are built by the tokens moving through the network. Two nodes of the tree are connected if one token of level $N+1$ represents the respective level N token (In this case, both tokens will visit exactly the same node).

The Algorithm in Details

According to the current situation (mostly the number of tokens in the queue of a node), the following functions are executed on each node or are applied to each incoming token on every node:

1. **Initiate:**
   For every keyword (document name) *lex* on a location p a level-0-message (token) is generated initially. It contains the following data:
   - The level of the message (token) N, initially set to zero.
   - The information about a data range interval $[lex_{min}, lex_{max}]$. At the beginning is $lex_{min} = lex_{max} = lex$.
   - A list of locations L and corresponding data, initially containing $L = \{(p, [lex, lex])\}$. $|L|$ denotes the number of elements in L.
   - The actual number of messages currently merged in this message Z_a. In addition, the list L contains R entries for randomly selected and free changeable visiting positions.
   - A (constant) maximal number of elements Z, which can be merged in this message. Note that $Z = Z_a + R$.
   - A counter T, showing how long there have been no changes in the message and a system wide constant value T_{max} indicating after how much cycles without changes a new, higher level message (token) may be generated.
   - Information m with $m \in L$ indicating at which location a $N+1$ message (token) was generated (zero-address).

2. **Forward:**
   After processing on a node, a message (token) is forwarded. This is carried out in a cyclic manner. The Z positions of this cycle contain the Z_a locations of the list L and $Z - Z_a$ randomly selected forward positions.

   Some delay time may be introduced on every node to avoid high network traffic.

3. **Merge:**
   Two messages (tokens) 1 and 2 will be merged if

   - they meet at the same location and
   - they have the same message level.

   In this case, the merged message contains the following information

   (a) $L := L_1 \cup L_2$
   (b) $lex_{min} = MIN(lex_{min,1}, lex_{min,2})$
   (c) $lex_{max} = MAX(lex_{max,1}, lex_{max,2})$
   (d) $Z_a = Z_{a,1} + Z_{a,2}$
   (e) $Z = MIN(Z_1, Z_2)$
   (f) $T = 0$
   (g) The level information N remain unchanged.

   In case $Z - R > Z_a$, the *Divide*-operation is called.

4. **Divide:**
   This operation divides a message into two new messages 1 and 2 of almost the same size but with disjunct data range intervals. Division is done such that

   (a) $Z_1 = Z_2 = Z$.
   (b) $Z_{a,1} \simeq Z_{a,2}$.
   (c) If $[lex_{min,1}, lex_{max,1}]$ and $[lex_{min,2}, lex_{max,2}]$ are the respective data range intervals of the new messages 1 and 2 $lex_{max,1} < lex_{min,2}$.
   (d) L_1 and L_2 are built corresponding to the assignment of the locations to the data range intervals defined above.
   (e) $T = 0$.
   (f) The message (token) level information N remain unchanged.

   Division should be done so that the structure of existing sub-message (sub-token) are not changed and their re-organization is not necessary.

5. **Generate:**
   If for a level-N-message ($T > T_{max}$ and $Z_a > 1$) or ($Z_a = Z - R$), one level-$N+1$-token is generated at the first passed location x of L.

   This token inherits all data from the original message (token) except the level and the location list L. If x is the node generating the new message (token) $L = \{(x, [MIN|_i(lex_{min,i}), MAX|_i(lex_{max,i})])\}$, where $i = 1(1)|L|$. The new level N is one level higher than that of the generating message (token).

   It should be emphasized that the rules for processing are the same for all messages/tokens at all levels.

It is easy to notice that the structure obtained from the algorithms above is a tree discussed in the previous section. Furthermore, the structural changes should converge to zero after some time if the community has not changed and the tree is a sorted one.

Of course, the tree structures can be used to support a more efficient search as is known from classical tree search algorithms. However, two cases must be distinguished.

1. **Search in the sorted tree** A search for a key lex_s appears on a location p. This request (lex_s, p) is given to the next passing token. If the nodes attached to that token can fulfill the request, the result will be returned after one period of the token. Otherwise, the request is given on the contact point to message (tokens) of the next higher level. This will be repeated until the data interval of the respective tokens fits to the request. Then it will guide the request down to the node hosting the searched data. This node can now response to the request directly.

2. **Search in a non-sorted, changing tree** The search request is processed in the same manner as the one described above. However, since the tree is not sorted, the data intervals of the level $N-1$ tokens of a considered level N token may overlap. Therefore,

   - In case of overlapping intervals, the token must be copied and one copy must be given to every fitting level $N-1$ token and
   - the search request must be forwarded to the token of the highest level (root token).

   It is important to mention that even the non-sorted tree can improve the search. However, not every key-entry in the system may be found.

DYNAMIC BEHAVIOUR

The section above has introduced the algorithm for communities with a static membership of peers. However, for real applications, the dynamic behaviour must be considered since nodes may join and leave a P2P community without prior notification. In addition, messages (tokens) can be lost due to transmission mistakes or computer breakdowns.

Therefore, the following situations may appear and should be solved by the corresponding recovery mechanisms:

1. Nodes are no more available, but the existence of messages (tokens) has not been affected.

In this case, a message (token) cannot be forwarded to a position in L, because the node is not available. Therefore, the message is forwarded to the next node after the one unavailable in L. Unsuccessful forward operations should be counted in a special time-out counter so that these nodes can be removed from L after a certain number of unsuccessful transmission requests.

2. Contact nodes to a token (message) of level N are no more available for level $(N+1)$-tokens (messages). In this case, the whole subtree below that message is disconnected from the remaining structure.

   Therefore, it is suggested that some locations form the $(N+1)$-token (message) be propagated to the nodes in the list L of the level N- messages (tokens) to realize a re-connect in this case.

3. Lost messages (tokens) must be re-initiated and re-connected to the remaining part of a structure. In this case, it may be suitable to leave copies of the message (token) on all nodes that have been visited. In this way, the nodes which are still active may re-initiate the message after some waiting time before the structure is destroyed.

   This will definitely reduce the restructuring effort compared with the method of using a static locally elected server (cluster center) to manage attached sub-nodes.

4. New nodes must be included in an existing structure. New nodes automatically generate new messages (tokens) which are sent through the network. If all nodes execute the algorithm above, such new nodes will be automatically included in a respective place in a structure which has already existed.

SIMULATION RESULTS AND DISCUSSION

Real world peer-to-peer systems are often very complex. Many parameters, such as different bandwidths of the connections between the nodes, different computing power of the nodes and the unknown size and structure of the network, make it hard to predict the runtime behaviour of the algorithm. Furthermore, the high dynamic of these systems makes the development of a mathematical model for these systems a difficult task. Too many parameters are to be considered.

Therefore, the algorithm was examined by means of a computer simulation which aimed to show the actual behaviour of the algorithm and whether the algorithm is working properly. As a result, the simulation setup was stripped down to the essential mechanisms described in the algorithm above. In this way, the delay of messages caused by the network, the different bandwidths of the connections between the nodes and the processing time for the messages at every node were disregarded.

The underlying network for the simulation setup is a small-world graph which provides a good approximation of a real peer-to-peer network (Deo and Gupta, 2001). It was created by using the edge reassignment algorithm. The simulation is running synchronously, i. e., in every (time) step of the simulation, every node is considered. The respective messages are processed and eventually passed to another node. This setup can only give a rough idea of the function of the algorithm, but it is sufficiently exact to get an idea of the overall behaviour of the system.

Parameters for the measurement are the network size which ranges from 500 to 3000 nodes in the network. More nodes would cause too long simulation time. Furthermore, different sizes of the neighborhood-warehouse, which is the number of known neighbours of a node, and the number of random steps which a message can do have been varied.

The algorithm should be able to work on a huge number of nodes. Therefore, most interesting parameters which were determined are the time (number of steps) needed to create the described structure and the influence of the network size. Figure 1 shows the number of simulation steps which are necessary to reach a stable state for a different number of random steps by using networks of different size. For a different number of random steps, the results are graphed in the figure. It clearly shows that an increasing number of random steps will speed up the generation of the final structure. This is obvious because the possibility that messages will be merged becomes higher. However, the drawback of a large number of random steps is that the interval of a message visiting 'its' nodes becomes bigger.

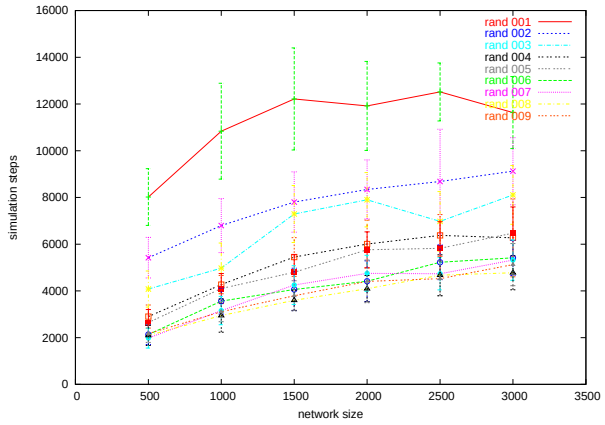

Figure 1: Number of simulation steps to stabilize the structure over network size for a different number of random steps

The impact of a different warehouse size is shown in figure 2. The creation time for the structure is almost the same for every warehouse size.

Figure 3 and 4 show the number of merge operations over time. A merging operation is the fusion of two messages which meet at the same node. Here, a network size of 3000 nodes was used for the simulation. Again,

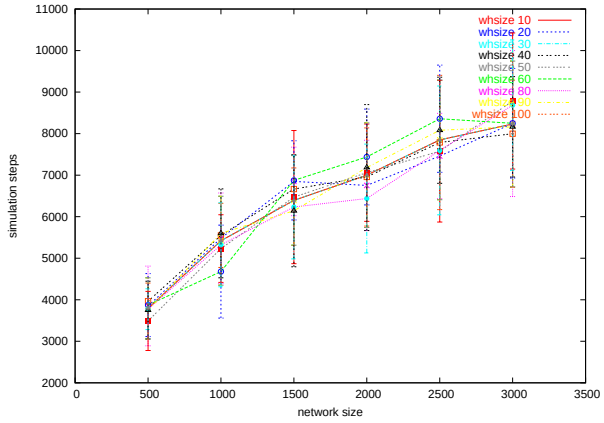

Figure 2: Number of simulation steps to stabilize the structure over network size for a different warehouse size

the simulations were done by varying the share of random steps and the warehouse size respectively. In both figures, the number of merge operations falls rapidly during the first 20 simulation steps. After this period, most messages have reached their maximum size and the number of merge operations keeps falling very slowly.

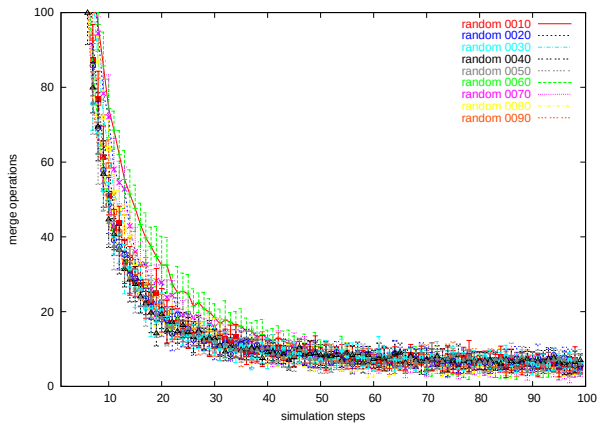

Figure 3: Merge operations over time by varying the number of random steps with a network size of 3000 nodes

The token population is very high at the beginning of the simulation (see Fig. 5). Every node is generating its own agent. This is obviously a problem of the simulation. In a real world environment, not all of the nodes would appear at the same time, but it shows that the situation will normalize very soon. Due to the merge operations, the population decreases fast and stabilizes at a low level. The number of agents remaining in the system depends on the size of the plan used by the agents. In the figure, the population for a network size of 3000 nodes is shown as an example.

To get an impression of the dynamic behavior of the algorithm, it was assumed that nodes join and leave the network. under this circumstances, the time to recreate the structure after removing an agent was measured in the simulation. Figure 6 shows the results. If an agent is

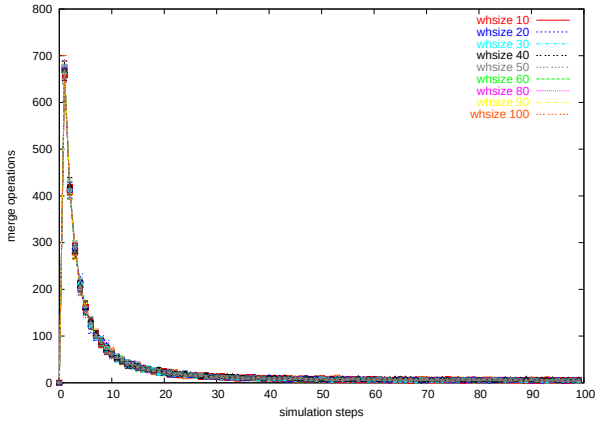

Figure 4: Merge operations over time by varying warehouse size with a network size of 3000 nodes

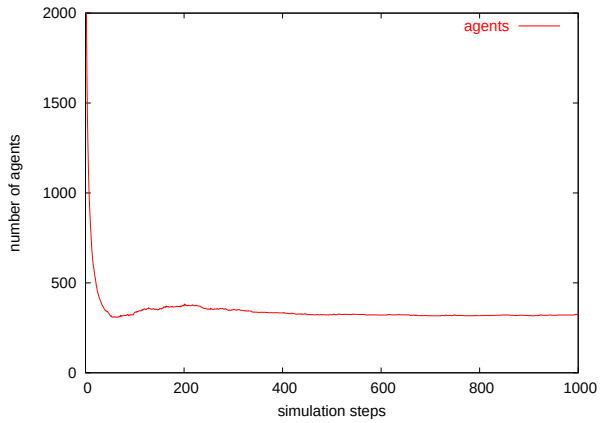

Figure 5: Agent population during the simulation for a network size of 3000 nodes

removed at a low level, the time to rebuild the structure is higher as if a higher level token was removed. This is the expected behaviour because the higher level tokens have to be recreated if a lower level agent fails.

It was mentioned before, that the content of every token is kept as a backup copy on each node. This enables a fast recovery if a token is lost. This backup copy was not used in the dynamic simulation. The goal was to show the recovering time without any additional feature. The time needed for fast recovery is shown in the bottom curve of Fig. 6. It is a constant time because after a node was not visited by the wanderer for a certain time, it will recreate the wanderer using its backup copy. This is independent of the level of the wanderer. In the figure, the timeout until a new wanderer is created was set to three times of the normal visiting interval which depends on the length of the plan L of the wanderer (16 in this example). Only in the rare case that all k nodes fail which are visited by this token, the recovery using the backup data is not possible.

If new nodes are added to a stable network, it might take some time to rebuild the stucture. If a node joins the system, it generates a new wanderer which will then

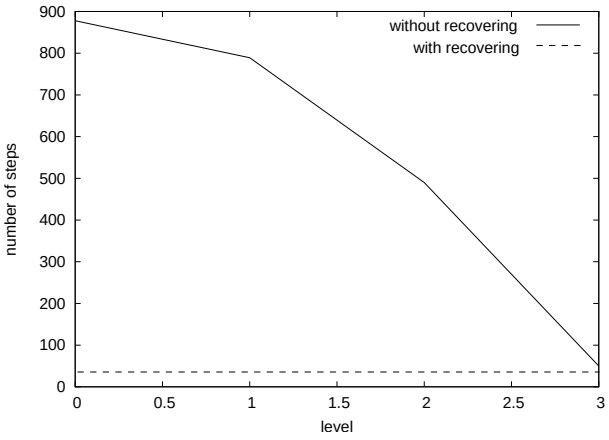

Figure 6: The time to recreate the structure if an agent was removed from a level. The upper graph shows the time without a recovery algorithm and the bottom curve shows the time needed by using the recovering algorithm.

be integrated after some time as described above. Figure 7 shows the number of necessary steps, if ten nodes are added to the network. The number of steps seems not to depend on the network size.

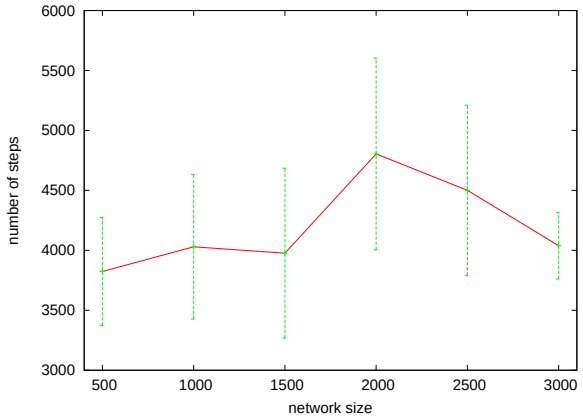

Figure 7: Number of steps to stabilize the structure if 10 nodes are added to a network of 500 . . . 3000 nodes

It could be shown by running the simulations that the creation of the sorted structure from a peer-to-peer network is possible by using the algorithm described above. The simulation was run about ten times to achieve meaningful results. More replications would have been even better but the simulation was very time-consuming. Nevertheless, the simulation did not take some problems caused by the dynamic nature of real world peer-to-peer networks into account.

SUMMARY

In the paper, an algorithm is presented which allows to build a sorted structure on top of a P2P-network in order to support the search in these systems. The number of messages and the time to find some information could be significantly reduced.

Similar to DHT and other approaches an additional overhead is introduced. However, the overhead is limited and the fault tolerance is higher than in other systems. In future publications, the behaviour the classical DHT system will be compared with this approach.

REFERENCES

Balakrishnan, H., Kaashoek, M., Karger, D., Morris, R., and Stoica, I. (2003). Looking up data in p2p systems.

Clarke, I., Sandberg, O., Wiley, B., and Hong, T. (2000). Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *ICSI Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA.

Deo, N. and Gupta, P. (2001). Graph-Theoretic Web Algorithms: An Overview. In *Innovative Internet Computing Systems (IICS 2001)*, pages 91–102, Ilmenau, Germany.

Gnutella (2004). www.gnutella.com.

Kaashoek, M. and Karger, D. (2003). Koorde: A simple degree-optimal distributed hash table. In *2nd IPTPS*, Berkeley, CA.

Kazaa (2004). www.kazaa.com.

Malkhi, D., Naor, M., and Ratajczak, D. (2002). Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st annual ACM symposium on Principles of distributed computing*. ACM Press.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. (2001). A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*.

Rowstron, A. and Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218.

Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H. (2001). Chord: A scalable Peer-To-Peer lookup service for internet applications. In *ACM SIGCOMM 2001*.

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