A Multi-Ring Method for Efficient Multi-Dimensional Data Lookup in P2P Networks

Nick Antonopoulos
Department of Computing
University of Surrey
Guildford, Surrey, UK

James Salter
Department of Computing
University of Surrey
Guildford, Surrey, UK

Roger Peel
Department of Computing
University of Surrey
Guildford, Surrey, UK

Abstract – In this paper, we describe a multi-ring approach to building multi-tiered P2P networks for efficient lookup of multi-dimensional data, utilising an alternative strategy for building the network overlay designed to reduce the hops required to route lookups and improve fault tolerance by allowing for the selection of high quality nodes with which to build subrings. We provide a case study showing how the method could be used to support 2-dimensional data in the form of keyword-value queries. Our calculations indicate that the presented method yields improvements in the average query hop count while reducing the amount of state stored on each node. The use of Preference Lists can further reduce the average hop count through bypassing previously traversed segments of the structure.

Keywords: peer-to-peer, distributed hash tables, multi-dimensional data, Chord

1.0 Introduction

The use of Distributed Hash Tables (DHTs) for resource discovery in peer-to-peer networks has become a highly studied research area. More recently, systems utilising multiple DHT structures have been explored. We show how a multi-layered multi-ring architecture based on Chord [1] can support multi-dimensional data lookup, describing our method by means of a case-study showing how it can be applied to a network supporting keyword-value pair queries. For the majority of topology configurations we demonstrate a reduction in messages required to route a query in worst-case scenarios compared with a single-layered Chord ring and other multi-ring architectures.

Our ring creation strategy follows a novel approach we call Lazy Ring Creation, creating sub-rings only when necessary and using optimal numbers of nodes by utilising processes defined in the ROME protocol [2]. Lazy Ring Creation decouples the ‘new keyword’ event from the ‘ring-building’ event and leads to smaller rings than in other systems. A keyword ring is only created once the number of queries or registered resources for the keyword rises above a threshold. Nodes self-monitor their workload and mechanisms are provided for overloaded nodes to recruit new machines to share their workload. We will show how this improves hop count and fault tolerance compared to other systems.

2.0 Related Work

Several systems based on Distributed Hash Tables have been proposed to substantially reduce query and update message traffic. Chord [1] organises nodes into a ring and can route a query in O(log n) messages while maintaining routing information about only $\log_2(n)$ nodes on every node in the network. Updates following a node joining or leaving the network require $O(\log^2 n)$ messages. Hop count (or messages consumed) is used as a standard metric to measure the performance and efficiency of P2P networks. Other widely used examples of DHTs in P2P networks include CAN and Pastry [3].

Multi-layered DHT structures contain multiple sub-rings, using a primary ring to route queries between different sub-rings. Various techniques are used to organise the network, but all assume secondary-level rings are either already built or are created based on node locality. Representative nodes are added to the primary ring, effectively building the network upwards. The system itself has no control over how many sub-rings exist or how many nodes each contains.

PLANES [4] partitions a single Chord ring into multiple smaller rings, but little regard is given to the information stored on each node in individual clusters. Although the top layer of the PLANES topology allows the cluster holding the target key to be located quickly, there are no guarantees that the same cluster will also contain similar keys to the one originally searched for.

Garcés-Erice et al [5] present Hierarchical Peer-to-Peer Systems and give a case study instantiation of a modified version of Chord. The lower level overlay is split into a number of sub-rings. One or more nodes from each sub-ring are inserted into the top-level overlay. A reduction in routing messages is demonstrated by assuming powerful machines with greater availability and connectivity are used in the top-level overlay, thus providing a smaller probability of node failure. However, ignoring probabilities of failure, reduction in messages is not possible in this approach.

In HIERAS [6] and Brocade [7], a single overlay contains all nodes. HIERAS rings co-exist in different layers, but each node is a member of a ring in each layer. Although average routing latency measured in presented experiments is less than Chord, HIERAS requires a larger average number of routing hops. In Brocade, a second overlay layer is used to improve performance of long range routing. However, nodes are grouped only by administrative domain rather than by the information contained within them.

In Coral [8], searches begin in a sub-group containing nodes in the same geographical region, continuing on a cluster with continental coverage before eventually switching to a single planet-wide cluster if the lookup has not been resolved in the intermediate stages. While searches that are resolved within the regional cluster may be faster than those in our system, more messages will be consumed to route a query in the worst-case.

In [9], a multi ring architecture is used to provide content locality at an organisational level, where nodes are added to an organisational ring based on their locality. Additionally, all nodes in the network join a global ring, unless they are connected behind a firewall or NAT. This increases the cost of traversing the global ring, compared with a smaller global ring containing only a small number of representative nodes.

3.0 Case Study System Model

In this section we present a case study demonstrating how our method could be used with multi-dimensional data comprising keyword-value pairs. Keyword-value queries are composed of one or more sets of keyword-value pairs. A single pair could be “linuxversion=9.1”, where “linuxversion” is the keyword and “9.1” is the value.

Our framework is based on Chord [1]. Unlike in Chord, where every node is organised into a single ring, we have multiple rings. Each ring is responsible for a single keyword. Each node in the keyword ring holds a list of IP addresses of machines hosting resources matching the value(s) and keyword for which the node has responsibility. In our example, the keyword is “linuxversion”, thus this is also the name of the keyword ring. “9.1” would be a value hosted on one of the nodes in the keyword ring. To enable

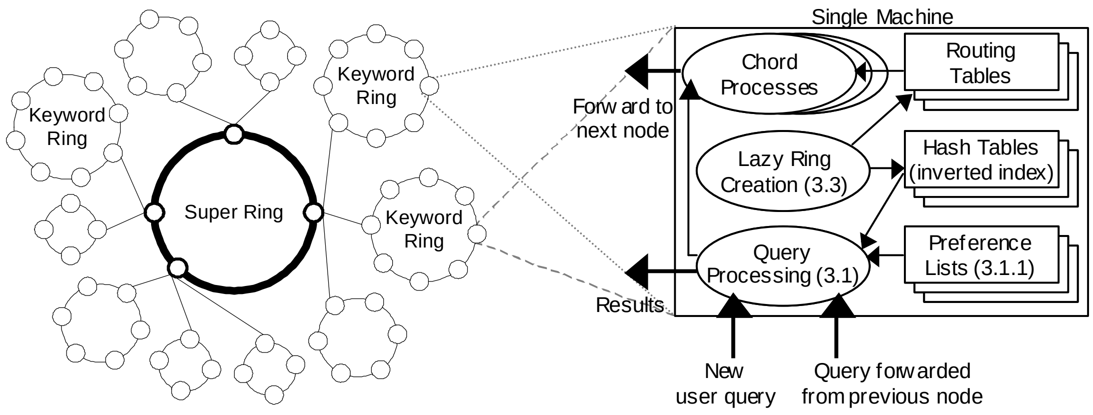

Figure 1. System model showing ring architecture and processes and data structures of a typical machine being a member of multiple rings.

keyword rings to be found, a Super Ring is used to host a ring of nodes that contain pointers to the other rings (Figure 1).

The system selects nodes as it requires them, so not every machine will necessarily be involved in the indexing part of the system. Nodes exhibiting only client behaviour (nodes that do not host any resources) are free to query the system, but do not need to register with the structure or take part in any rings. This allows for increased dynamicity in session times of clients without affecting the stability of the rest of the system.

3.1 Querying

Queries originate on machines which may or may not be members of our structure, but at minimum must know the address of at least one supernode. This is used as an entry point into the super ring. These machines may be acting as gateways to the peer-to-peer network for other computers on, for example, the same local area network. Before any traversing of the structure is undertaken, the machine is first searched to determine whether it can provide the required resource itself, thereby saving the need to transmit messages across the peer-to-peer network when an answer can be provided more quickly in the local network.

Once a query has been generated, the query keyword is hashed to give the ID of the supernode responsible for that keyword. The query is routed to this supernode using the standard Chord routing algorithm. Addresses of multiple nodes that are members of the required keyword ring are listed on the supernode. One of these is selected and the query is forwarded into the keyword ring.

The value of the keyword is hashed to give the ID of a node in the keyword ring and the query is sent to it using the standard Chord routing algorithm. The node hosts a list of providers of resources matching the keyword and value specified. This list is added to the query and returned to the query instigator. The providers are then contacted to determine whether they will provide access to the resource. We assume invocation of the resource to be an external process.

3.1.1 Preference Lists

Further reduction in the number of hops across the network is realized by introducing preference lists [10]. The list of providers returned in response to a query is cached on the machine originating the query, so that future queries for the same keyword-value pair can be resolved more quickly. As the query is repeated, the providers are ordered by those most likely to supply the required resource, based on how often they have done so in the past. Several criteria affect the likelihood a provider will actually allow access to the resource (or whether it is desirable to access the resource from the provider), such as compatible security policies, workload or connectivity between it and the consumer. Preference lists provide a direct mapping between requestor and resource providers. The entire indexing structure is bypassed, reducing the hop count to zero. The size of preference lists is constrained to prevent them from growing infinitely large. When the maximum available capacity to store preference list information on a node (a user-defined parameter) is exceeded, preference lists for older less-frequently used values and keywords are removed.

Preference lists can also be used as shortcuts to bypass the super ring for queries where the keyword has been previously searched for but the value is different. In addition to the list of providers, the addresses of the keyword ring node that hosted the list and supernode that pointed to it is stored in the preference list. The keyword ring node can be used as an entry point into the keyword ring for the repeated keyword, eliminating the need to route the query through the super ring to find a keyword ring entry point.

Xie and O'Hallaron [11] have shown that query frequency follows a Zipf distribution, where many queries are repeated multiple times. Therefore, our approaches for reducing the number of messages required for answering repeated keyword and repeated keyword-value queries are likely to significantly reduce the overall message traffic in the system.

3.2 Network Construction

Network construction is controlled by ROME [2]. To join the system, a machine contacts the ROME bootstrap server to find the address of a node already in the network. The machine inherits the preference list of the existing node, which should list the address of at least one supernode (which is the minimum information the new machine needs to obtain an entry point into the super ring for query processing). The actual lists of resource providers are not inherited, as they will almost certainly be different or inaccurate for the new machine.

If the machine does not wish to provide any resources, it need not do anything else. There is no expectation that it will register within any structure, which allows for dynamicity in session times for nodes acting exclusively as consumers.

3.3 Lazy Ring Creation

Resources are described using one or more keyword-value pairs, which can subsequently be searched for as described above. In addition to a list of keyword-value pairs, each resource has an owner (the IP address of the machine providing access to it) and an identifier to uniquely identify the resource.

When new resources are registered in the system, update queries are submitted for each keyword-value pair the resource will be registered against. These queries are similar to standard queries but, instead of returning a list of results, they record the resource provider's address on the relevant keyword ring node. Machines of new resource providers register with the ROME bootstrap server, which stores details of the machine's capacity in its node pool. A machine is not automatically assigned to a keyword ring, but may eventually become a node in the super ring or a keyword ring.

A resource can be removed from the system in a similar manner to addition, where a query is performed for each keyword-value pair associated with the resource, this time removing references to it from the relevant keyword ring nodes. No attempt is made to remove references to the resource from preference lists. As no information is available about which preference lists reference the resource, a broadcast throughout the network would be necessary. We assume it is less costly to remove references to the resource from a preference list when its removal is discovered following a query compared to a network-wide broadcast.

For new keywords for which a keyword ring is not yet built, resource information is held on the supernode responsible for the keyword. If the supernode does not have capacity (processing cycles and storage) to cope with expected query demand for the new keyword, a new supernode will be selected and inserted into the super ring using ROME. Once the trigger level for creating a keyword ring is met, the supernode responsible for the keyword will determine the number of nodes needed for the keyword ring and the capacity (primarily in terms of query processing load) each one will need. This calculation is based on the number of values for the keyword and the query workload experienced for each value to date, with the required capacity per node being approximately $z = \bar{l} + 2\sigma$ and number of nodes being approximately $\frac{totalkeywordworkload}{z}$, where $\bar{l}$ is mean workload per value and σ is standard deviation. Assuming the workload is normally distributed, the mean plus two standard deviations is used to account for 95% of the values. Additional nodes are selected to satisfy values with individual workload higher than *z*. Once the number of nodes required is finalised, the ROME bootstrap server's node pool is utilised to find the required number of machines with spare capacity, with the additional requirement that each machine must have a probability of failure less than a specified maximum (to limit probability of query routing failure).

Since the ROME layer runs above Chord on each node, keyword ring nodes and supernodes monitor their workload and use ROME to insert another node into their ring to share their workload if they become overloaded. This provides a mechanism for dealing with nodes unable to cope with an increased workload placed upon them (something often left unaddressed in other systems).

Selecting superpeers from within sub-rings as in [5] and [9] could lead to situations where a sub-ring comprises only unstable nodes which would be promoted to the top-level overlay, reducing fault tolerance of the primary ring. These systems also fail to address problems where a promoted node cannot handle the traffic requirements placed on it in the primary ring. Supernodes in our system are selected from a global set, therefore avoiding these conditions.

[5] select their superpeers from a list of local candidate nodes. This could lead to situations where a group comprises only low powered nodes, one or more of which would be promoted to the top-level overlay, thereby reducing performance of the part of the topology shared system-wide. In addition, this method involves a monitoring process and the periodic broadcast of an ordered list of candidate nodes to all the peer nodes in the group. The authors do mention that the broadcast would not be cost-effective in large groups and peers could learn "lazily" about network changes. However, the monitoring process must continually take place, generating an unbounded number of messages over time. If nodes actively send updates to superpeers in their group, the message cost for every node to send an update is $O(n)$ every update cycle where n is the number of nodes in the network. In a dynamic environment such as a P2P network, this monitoring must take place frequently to maintain accurate information, generating a large and unscalable volume of messages.

In comparison to the push strategy outlined in [5], our pull strategy does not generate any update messages. Instead, messages are only used when an available node is searched for on the ROME bootstrap server. As the message cost for the push strategy is at least n every update period, it is clear the pull strategy should generate less messages in all scenarios apart from where many nodes need extra capacity simultaneously and there are very few nodes with spare capacity available (the network is close to collapse).

Additionally, using Lazy Ring Creation maximises the use of supernode capacity and makes the selection of new nodes a rarer event than in an eager scheme where keyword rings are automatically created and, as in [5], a supernode must be created for *each* keyword ring. This also has the side-effect of reducing the size of the super ring, meaning less hops are required to traverse it during query processing.

4.0 Message Costs

As we use the standard Chord routing algorithm, our system uses the same volume of messages to route a query inside our rings. [1] has shown the worst case number of messages to route a query through a standard Chord ring is $\log_2(n)$ and $½\log_2(n)$ in the average case (assuming all routing information is correct), where n is the number of nodes in the ring. In the worst case, it follows that the number of messages required to route a query through our structure would be $\log_2(s) + \log_2(k) + 1$ where s is the number of super ring nodes and k is the number of nodes in the keyword ring. The extra message is needed to jump between the two rings, because each is independent unlike in systems such as [5] where a node in the top-level topology is also a member of the sub-ring to which it is related. We assume here that all finger table entries are correct. If not, the absolute worst-case in approaches relying on an underlying Chord ring becomes linear to the number of nodes in the ring, with queries being forwarded between neighbouring nodes.

We show a reduction in hops required to route a query versus a standard Chord implementation: Given a multi-ring network C with s nodes in the super ring and k nodes in each keyword ring, then the comparable single layered Chord ring C' has one node for each node in all keyword rings, that is wsk nodes where w is the number of keyword rings for which each supernode is responsible. Given a ring C (or C') denoted by $WC(C)$ being the worst case number of hops required to route a query and $AC(C)$ being the average case number of hops required to route a query, then from [1] for a single Chord ring $WC(C')=\log_2(wsk)$ and $AC(C')=½\log_2(wsk)$. For a multi-ring network:

$WC(C) = WC(\text{super ring hops}) + 1 + WC(\text{keyword ring hops}) = \log_2(s) + \log_2(k) + 1 = \log_2(sk) + 1$ and $AC(C) = AC(\text{super ring hops}) + 1 + AC(\text{keyword ring hops}) = ½\log_2(s) + ½\log_2(k) + 1 = ½\log_2(sk) + 1$.

Lemma 4.1. *With C and C' as above then if w>2 then WC(C') > WC(C) and if w>4 then AC(C') > AC(C).*

Proof. If $w>2$ then $WC(C') = \log_2(wsk) > \log_2(2sk) = \log_2(sk)+\log_2(2) = \log_2(sk)+1 = WC(C)$. If $w>4$ then $AC(C') = ½\log_2(wsk) > ½\log_2(4sk) = ½\log_2(sk)+½\log_2(4) = ½\log_2(sk)+1 = AC(C)$. □

Other systems that add one or more supernodes to the primary ring for each sub-ring use more hops to route a query than our approach, because we create a smaller super ring by allowing each supernode to host pointers to multiple sub-rings. This is possible because of our detachment of the two topological layers, at a cost of one extra hop between the super ring and selected keyword ring. This detachment is unattainable if supernodes are selected from sub-rings such as in [5] and [9]. In [5] nodes in their top layer are also members of the bottom layer, meaning detachment would be impossible unless bottom layer nodes were permitted to be members of more than one group. [5]'s structure cannot reduce hop count in isolation. They only achieve reduction by assuming more powerful machines with greater availability and less probability of failure are inserted into the super ring.

We can also show improved performance for queries containing repeated keywords. Since an entry point into the keyword ring is already known, the worst case number of messages becomes $\log_2(k)$, with $\frac{1}{2}\log_2(k)$ in the average case.

Improvement in message cost for repeat keyword queries is not possible in standard Chord, since there is no logical ordering of the nodes into keyword groups and there is no way to determine a segment of the ring that contains possible answers. Comparing with our system in the worst case gives $\log_2(wsk) > \log_2(k)$. By using shortcuts provided by preference lists, we can realise savings of $\log_2(ws)$ messages in the worst-case for every repeated keyword query.

Comparing with the Chord instantiation of a Hierarchical Peer-to-Peer system shown in [5], to create a topology containing the same number of keyword rings and nodes within each keyword ring as in our system, the number of superpeers required in their system would be *ws* since each superpeer can only be responsible for a single subgroup. In the worst-case (assuming a stable network without node failure) the number of messages required to route a query in their instantiation would be $\log_2(wsk)$, which is greater than in our system for $w>2$.

For repeated-keyword queries, we again show improvement. Hierarchical Peer-to-Peer does not contain a mechanism similar to our shortcuts, so queries will always require $\log_2(wsk)$ messages in the worst case. In comparison with our system, we can reduce the required worst-case messages by $\log_2(ws)$: $\log_2(wsk) > \log_2(k)$.

Depending on how many keywords each supernode holds and whether a query is for a new keyword-value pair, a repeated keyword or a repeated keyword-value, our system can give varying degrees of message savings in comparison with both Chord and multi-ring topologies.

4.2 Effect of Ring Size on Query Failure Rate

So far we have shown the benefits in message costs of building smaller rings than in other systems. However, it is also important to understand the impact of smaller rings on the ability to resolve queries under failure conditions to ensure that, whilst we show improvement in messaging costs, we are not adversely affecting the fault tolerance of the system.

To analyse this we have used p2psim [12], a freely available simulator for peer-to-peer protocols, to simulate lookups in different sized Chord rings constructed of nodes with varying mean lifetimes. Each simulation was run for 4000 seconds, with a mean of 10 seconds between lookups. The stabilisation process was run approximately every 200 seconds. In each case, the percentage of correct lookups was averaged over ten simulation runs. The results are plotted in Figure 2.

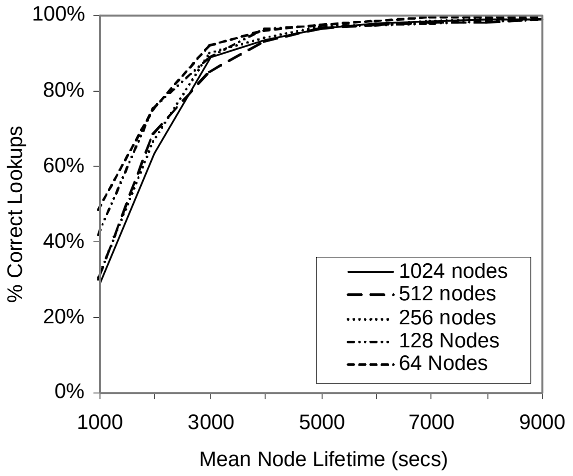

Figure 2: Effect of Ring Size on Lookups under failure conditions

It can be seen that the size of a ring has minimal effect on the number of successful lookups. Small differences can be observed, showing smaller rings exhibit slightly better fault tolerance. This may be because a small ring will take less time to re-stabilise than a larger ring following a node failure, since there will be fewer distinct finger table entries which must be corrected.

Another, more obvious observation is the impact that low quality nodes (nodes with short lifetimes) can have on the ring. Using Lazy Ring Creation, we are able to select nodes to join our rings, giving the opportunity of selecting high quality nodes to maximise the number of correctly resolved queries. Since our rings are smaller than in other systems (due again to Lazy Ring Creation) there is a higher probability that high quality nodes can be picked. [13] shows that heterogeneity between nodes in current peer-to-peer networks can be extreme. Therefore, selection of a small number of high quality nodes should lead to a lower average probability of node failure within the ring as a whole, compared with a larger ring containing more heterogeneous nodes.

5.0 Conclusion

Lazy Ring Creation is a novel approach to ring building that reduces the number of supernodes and keyword nodes created compared with other multi-ring architectures, thereby increasing the scalability of the network and reducing further the number of hops required to route a query. Utilising a ROME bootstrap server and node pool allows nodes with low probability of failure to be selected from a global set, enabling creation of rings with small numbers of nodes whilst retaining probability of query routing failure comparable to or better than other systems. Since node quality is controlled and nodes do not automatically become members of rings upon machines joining the network, Lazy Ring Creation allows for less frequent runs of the stabilisation algorithm, thereby reducing ring maintenance costs.

6.0 References

[1] I Stoica, R Morris, D Liben-Nowell, D Karger, M F Kaashoek, F Dabek, and H Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications". *IEEE/ACM Transactions on Networking, Vol 11, No 1* (Feb 2003), pp.17-32.

[2] J Salter, and N Antonopoulos, "ROME: Optimising DHT-based peer-to-peer networks", in *5th International Network Conference (INC 2005)*, Samos Island, Greece, July 5-7 2005.

[3] M Kelaskar, V Matossian, P Mehra, D Paul and M Parashar, "A study of discovery mechanisms for peer-to-peer applications", in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid Workshop on Global and Peer-to-Peer on Large Scale Distributed Systems*, Berlin, Germany, May 2002, pp. 444-445.

[4] P Triantafillou, "PLANES: The next step", in *Peer-to-Peer Network Architectures, SIGCOMM Workshop on Future Directions in Network Architectures (FDNA-03)*, Karlsruhe, Germany, August 27 2003.

[5] L Garcés-Erice, E W Biersack, K W Ross, P A Felber, G Urvoy-Keller, "Hierarchical Peer-to-peer Systems". *Parallel Processing Letters, Vol 13, No 4* (Dec 2003), pp. 643-657.

[6] Z Xu, R Min and Y Hu, "HIERAS: A DHT based hierarchical P2P routing algorithm", in *2003 International Conference on Parallel Processing*, Kaohsiung, Taiwan, October 6-9 2003.

[7] B Y Zhao, Y Duan, L Huang, A D Joseph and J D Kubiatowicz, "Brocade: Landmark routing on overlay networks", in *1st International Workshop on Peer-to-Peer Systems (IPTPS 02)*, Cambridge, MA, USA, March 7-8 2002.

[8] M J Freedman and D Mazières, "Sloppy hashing and self-organising clusters", in *2nd International Workshop on Peer-to-Peer Systems (IPTPS 03)*, Berkeley, CA, February 20-21 2003.

[9] A Mislove and P Druschel, "Providing Administrative Control and Autonomy in Structured Peer-to-Peer Overlays", in *3rd International Workshop on Peer-to-Peer Systems (IPTPS 04)*, San Diego, CA, USA, February 26-27 2004.

[10] N Antonopoulos and J Salter, "Towards an Intelligent Agent Model for Resource Discovery in Grid Environments", in *International Conference on Applied Computing 2004 (AC 2004)*, Lisbon, Portugal, 23-26 Mar 2004, pp. II187-II191.

[11] Y Xie and D O'Hallaron, "Locality in Search Engine Queries and Its Implications for Caching" in *21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, New York City, NY, USA, June 23-27 2002.

[12] p2psim. *http://www.pdos.lcs.mit.edu/p2psim/*

[13] S Saroiu, K Gummadi and S Gribble, "A measurement study of peer-to-peer file sharing systems", in *Multimedia Conferencing and Networking*, San Jose, January 2002.