Collision Avoidance in Hierarchical Peer-to-Peer Systems

Yong Meng TEO
Marian MIHAILESCU

Asia Pacific Science and Technology Center
SUN Microsystems Inc.

November 2007
Singapore

Collision Avoidance in Hierarchical Peer-to-Peer Systems

Yong Meng TEO
Marian MIHAILESCU

TECHNICAL REPORT
APSTC-TR-2007-03

Abstract

In a two-level Chord-based hierarchical peer-to-peer system, nodes from the first-level overlay network, called supernodes, act as gateways to peers organized in the second-level overlay. Collision occurs when peer-to-peer operations create more than one supernode with the same node identifier in the first-level overlay. Collisions are reported to enlarge first-level overlay by more than 12 times the ideal size and thus increase the lookup path length. Other consequences of collisions include lookup failures even though the lookup resources exist in the network and reduced scalability of the system. In contrast to collision detection and correction techniques, this paper proposes: (i) a collision-free approach for the join and leave operations, and (ii) a new stabilization scheme to recover from second-level overlay node failures. However, avoiding collision due to supernode failures is complex and remains a challenge. Simulation results show that collisions can be avoided and our algorithms improve the overall system performance.

Keywords: collision avoidance, peer-to-peer, chord, hierarchical, distributed hash tables

Email:
teoym@comp.nus.edu.sg

Asia Pacific Science and Technology Center
50 Nanyang Avenue, N3-01-C10
Singapore 639798

Collision Avoidance in Hierarchical Peer-to-Peer Systems*

Yong Meng Teo and Marian Mihailescu

Department of Computing Science
National University of Singapore
Computing 1, Law Link
Singapore 117590

[teoym,marianmi]@comp.nus.edu.sg

Abstract

In a two-level Chord-based hierarchical peer-to-peer system, nodes from the first-level overlay network, called supernodes, act as gateways to peers organized in the second-level overlay. Collision occurs when peer-to-peer operations create more than one supernode with the same node identifier in the first-level overlay. Collisions are reported to enlarge first-level overlay by more than 12 times the ideal size and thus increase the lookup path length. Other consequences of collisions include lookup failures even though the lookup resources exist in the network and reduced scalability of the system. In contrast to collision detection and correction techniques, this paper proposes: (i) a collision-free approach for the join and leave operations, and (ii) a new stabilization scheme to recover from second-level overlay node failures. However, avoiding collision due to supernode failures is complex and remains a challenge. Simulation results show that collisions can be avoided and our algorithms improve the overall system performance.

Keywords: collision avoidance, peer-to-peer, chord, hierarchical, distributed hash tables

1. Introduction

Peer-to-peer (P2P) networks are a class of distributed systems that support the sharing of computer resources and services. A key requirement in peer-to-peer applications is an efficient lookup service to locate the node that stores a particular resource. Current generation of P2P systems adopts a decentralized approach, where the lookup service does not rely on a central directory, as in the centralized approach, to answer queries. A decentralized system, with no single point of failure, is more robust and scalable. However, searching an unstructured decentralized network is less efficient and there is no guarantee of finding a resource, even if it exists in the network [3].

*A version of this technical report is published in *Proceedings of the 7th International Conference on Networking (ICN08)*, pp. 336-341, IEEE Computer Society Press, Cancun, Mexico, April 13-18, 2008.

Structured overlay networks using distributed hash tables (DHT) guarantee better lookup results by using a mechanism that maintains the structural properties of the network while peers are dynamically joining and leaving the network [1, 5]. More efficient discovery methods can be implemented using DHTs, where both node and data are hashed into the overlay [8, 9, 11, 13]. A lookup query is also hashed to a key, and nodes can determine, based on that key, how the query should be forwarded. DHT-based schemes such as Chord [11] adopt a one-level overlay topology with a lookup performance of $O(logN)$, for a system with N peers. Recent developments in hierarchical overlays has the advantages of faster lookup times, less messages exchanged between nodes, and scalability [2]. In two-level hierarchical overlay networks, peers are grouped according to a common property such as resource type for a lookup service used in resource indexing and discovery [12]. However, concurrent operations in a decentralized and dynamic peer-to-peer system may create multiple groups for the same resource type, before the inconsistent views of the overlay network topology are updated in a process called stabilization. The creation and existence of more than one group with the same identifier is called a *collision*. Collisions result in lookup operations returning incorrect results, increased stabilization costs to resolve inconsistent states of the overlay network and reduced scalability of the system. As reported in [6], in a two-level hierarchical overlay network collisions increase the size of the top-level overlay by as much as 12 times the ideal size.

In contrast with the collision detection and correction scheme described in [6], this paper proposes a collision avoidance approach. Using the example of two-level Chord-based hierarchical system in [12], we discuss how join and leave operation can be made collision free. We argue that avoiding collisions has a number of advantages:

- *no overhead cost* - runaway collisions are very costly, as shown in [6]: detecting and resolving collisions is highly difficult in a decentralized and dynamic peer-to-peer system with high churn rate;
- *reduced bootstrap time* - new peers join the network at a faster rate because the time between the join event and the update of the underlying overlay network states is reduced;
- *improved lookup performance* - without collision, the top-level overlay is maintained at the ideal size;
- *faster resource availability* - costly resolution is not necessary and resources are available once the nodes join the network.

The remainder of the paper is organized as follows. section 2 presents related work, and section 3 presents our collision avoidance scheme. Simulation methodology and results are discussed in section 4. Finally, section 5 concludes the paper.

2. Related Work

Garcés-Erice et. al propose a hierarchical framework that makes use of DHTs in [2]. The framework has several advantages when compared to a flat structure: transparency (moving a key from one peer to another inside the same group is transparent to the top-level), administrative autonomy (each group can choose its own intra-lookup protocol) and faster lookups (the average number of peer hops in a lookup is reduced using hierarchical routing). In the two-level hierarchy described by [2], the nodes that are contained in both level overlays are called *superpeers*. However, these peers are assumed to be the most powerful and stable and thus unlikely to fail, therefore the extent of collisions is not studied.

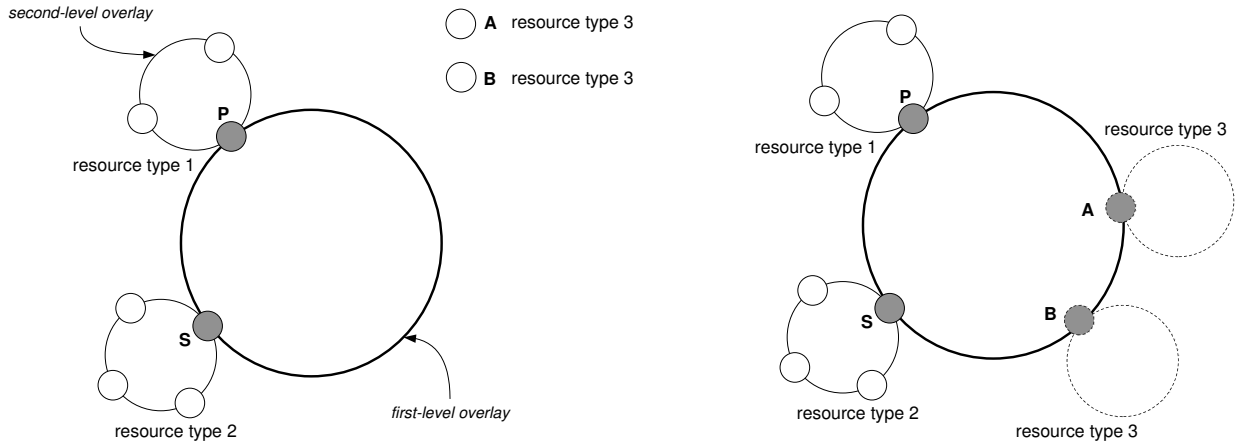
A framework for a peer-to-peer lookup service used for indexing and discovery is described in [12]. The problem of collisions is discussed in an additional paper [6], where a collision detection and correction algorithm is proposed to discover and merge groups of nodes with the same resource type. However, this approach incurs a high stabilization overhead to reduce collisions, while not entirely eliminating them.

3. Collision Avoidance Scheme

Network dynamics that has consequences on the overlays is determined by user-operations such as join, leave and also by node failures. Our algorithms are designed to work in a dynamic environment, where peers from both levels of the overlay can join, leave and fail at any time and no node is assumed to be stable. Figure 1 shows a two-level Chord-based overlay network with the first-level overlay consisting of groups of peers with the same resource type. Because of the good results shown by Chord as a flat structure [11], we use it as the top overlay network. For the second-level overlay we used a simplified schema, where the supernode keeps track of all the nodes in the group. This gives the best results for small groups (tens, hundreds of nodes), with lookup complexity of $O(1)$. For larger groups, overlay topologies such as Chord, Pastry [9] or Tapestry [13] that give better results can be used. However, efficient organization of the second-level overlay is beyond the scope of this paper.

Figure 1 shows two resources, A and B, of the same type (denoted as *resource type 3*) but belonging to different peers, joining a two-level hierarchical Chord system. When A and B concurrently join the overlay network at different network entry points, both detect that *resource type 3* does not exist in the network and each create separately a supernode for the same resource type. This phenomenon is called a *collision*.

The following sections discuss our proposed collision-free join and leave operations, and collisions due to node failures.



(d) Network state before A and B join

(e) A, B create 2 groups for the same resource type

Figure 1. Collision in a Hierarchical Chord System

3.1. Join

Before a resource joins one of the overlays, it is assigned a unique identifier based on its type and administrative domain[1]. This operation is called virtualization [12]. A node sharing more than one type of resource will have a number of virtual identifiers equal to the number of resource types and will join the corresponding groups.

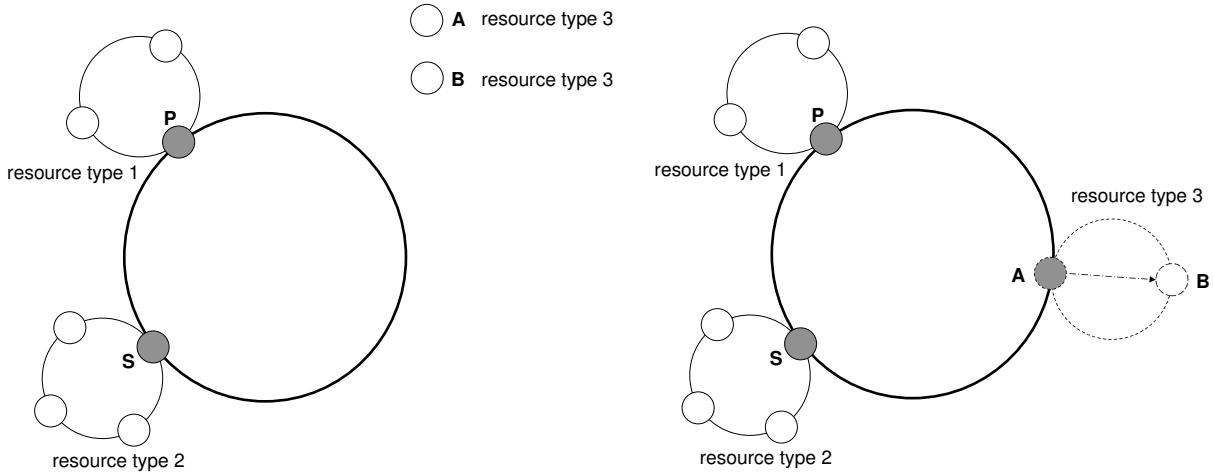
Resources are grouped based on type for ease of discovery in resource allocation in applications such as computational grids. Thus, each group is identified by a unique key equal to the resource type identifier. Each virtual node created starts the joining process by searching the group containing resources of the same type (using the resource type identifier as the key). Independent of the *bootstrap node* where the peer joins the overlay, the group lookup will be routed across the first-level overlay, according to the Chord protocol, to the supernode responsible for the key. In one case, the joining virtual node finds a supernode with the same resource type, and it enters the second-level overlay. In the other case, a supernode with the same resource type as the joining virtual node does not exist. The query is answered by the supernode of the successor group, and the joining virtual node becomes supernode in a new group for the resource type. Predecessor and successor pointers of the affected supernodes are corrected by the Chord stabilization protocol.

Figure 1 shows a network with two groups, where P is the supernode of the *resource type 1* group and S is the supernode of the *resource type 2* group. A and B are resources (virtual nodes) of the same type, but from different peers, that join concurrently using different bootstrap nodes. However, Chord routes their requests to the same virtual node, S, the successor of *resource type 3*. Supernode S returns to both A and B that no group exists for *resource type 3*; consequently, A and B creates separate groups, with each virtual

[1] We assume that for each administrative domain only the index server will join the network.

node as the supernode of a group (Figure 1b). Collision occurs and it is not corrected by the current Chord stabilization scheme.

Figure 2 illustrates our collision-free join operation. A group query for a resource type that does not exist in the top-level overlay is resolved by the predecessor supernode, instead of the successor. The predecessor node also updates its pointers to reflect the changes in the network. This modification of the Chord protocol is very important in our algorithm, as the new group is reflected immediately by the network and can be used in the next lookup. Consequently, virtual node P responds to the first query that arrives (e.g. initiated by A), and it also modifies its successor pointer to the peer that shares resource A, as the new supernode for *resource type 3* group. Because requests are sequentialized at each node, the next request that is processed (initiated by B) can now use P's successor pointer to reach the supernode for *resource type 3*. This is not possible in the previous algorithm because the successor pointer is not updated until the join process of resource A is completed. Joining with different bootstrap nodes makes no difference, as the requests are routed by the overlay network to the predecessor, which have a correct successor node. Collision no longer occurs and B joins the second-level overlay.



(a) Network state before A, B join

(b) B joins second-level overlay

Figure 2. Concurrent Node Joins without Collision

The detailed join algorithm in Figure 3 is divided into the following steps:

1. A joining virtual node performs a group lookup, which is routed in the top overlay to the supernode responsible for the key (i.e. resource type);
2. If a group for the resource type exists, a supernode is already created for the resource type and the joining node becomes a member of the second-level overlay;
3. If a group for the resource type does not exist, the supernode that responds updates its successor

pointer to the joining node. The joining node becomes the supernode of the newly created group, and set its predecessor pointer to the supernode that answered the lookup query, and the successor pointer to the old successor of its predecessor;

4. Stabilization is used by the created supernode to build a finger-table.

```
n.join(n')
    g = group(n); // unique id based on res.  type
    g' = find_group(n);
    if  g is g'
    then // join the second-level overlay
        sn = supernode(g');
        n.join_group(sn);
    else
        predecessor(g) = g';
        successor(g) = successor(g');
    notify (successor(g), UPDATE_PREDECESSOR);
        n. stabilize ();

n.find_group(n')
    // use find_predecessor from Chord
    x = find_predecessor (n');
    if  group(x.successor) is group(n)
        return  group(x.successor);
    return  group(x);
```

Figure 3. Collision Free Join Operation

For the second-level overlay, we assume for simplicity that the supernode maintains a linked list with all the peers in the group. For each new peer that joins, an entry is added in the list.

3.2. Leave

A node leaving the system can be a supernode or a member of a second-level overlay. Collision does not arise when a member node leaves the second-level overlay.

When a supernode leaves, a joining peer may create a new group for that resource type, before a new supernode for the group is determined. To prevent this type of collisions, the supernode notifies its successor and predecessor in the first-level overlay to update their finger tables, and a new peer within the group is then elected as the new supernode. Since the join operation is collision free, we adopted a simple (but costly) scheme to determine the new supernode. All the peers in the group are first disbanded. The nodes forced out of the group rejoin the system, and the first node to rejoin becomes the new supernode. The collision free leave algorithm is shown in Figure 4.

```
n.leave()
    g = group(n); // unique id based on res.  type
    if n is supernode(g)
    then
        notify (predecessor(g), UPDATE_SUCCESSOR);
        notify (successor(g), UPDATE_PREDECESSOR);
        while not empty (g) do
            n' = g.pop();
        n'.join();
else // leave the second-level overlay
        notify (supernode(g), NODE_LEAVE);
```

Figure 4. Collision Free Leave Operation

3.3. Fail

A more complex case that leads to collisions is when nodes fail. A node failure affects both successor pointers and the consistency of the finger tables of other peers in the network. Without consistent successor pointer, joining virtual nodes may create groups for resource types that already exist. Thus, accurate successor pointers are essential for the correctness of the topology. Fingers assure that lookup performance is $O(\log N)$, where N is the number of peers. The correctness of the top-level overlay is maintained by the Chord stabilization protocol. But when a supernode fails, stabilization may be performed after collisions have occurred. Avoiding this type of collision remains a challenge. For this case, our stabilization protocol included collision detection and correction, as in [6]. Each node runs this protocol at a predefined period of time, p.

Node failures at the second-level overlay are detected using an overlay-specific stabilization protocol. In our scheme, the supernode periodically checks the peers within its group for liveness, and peers that have failed are removed from the group. Also, each peer in the second-level overlay checks the supernode; if the supernode failed, the peer rejoins the network. The same stabilization period p as in the top-level overlay is used for the second level.

4. Simulation Results

To evaluate the performance of our algorithms, we modified the Chord simulator [10] to model a hierarchical Chord system with N nodes organized in two-level overlays. We compared our results with collision detection and resolution [6]. Nodes are grouped based on its resource type. Each supernode maintains one successor pointer, one predecessor pointer, and $O(\log G)$ fingers, where G is the total number of groups (or

resource types). Stabilization performed by each supernode at periodic intervals of p seconds corrects the successor pointer and one finger. As this paper focuses on collisions in the top-level overlay, the topology and performance of the second-level overlay is not discussed here. A supernode maintains a list consisting of all member nodes in its group. During stabilization, each node within a group periodically checks the supernode, and vice-versa. The time to process each request (join/leave/lookup) is assumed to be uniformly distributed between 5 and 15ms, and link latency is exponentially distributed with a mean of 50ms.

4.1. Join and Leave

We study the impact of collisions due to join and leave. The first set of experiments study the join operation using the same configuration and settings as in [6]. Assume a peer-to-peer system with 100,000 nodes (N), 1,000 resource types (G) and a workload of 100,000 joins with inter-arrival time exponentially distributed with a mean of 1 second. Simulations results confirm that there is no collision due to join using our approach.

Figure 5 shows the number of groups created in the top-level overlay over the simulation duration. For comparison, we plotted three main cases: "with collisions" to show the extend of collisions, "detect and correct" as in [6], and our proposed "collision avoidance" scheme. Two stabilization intervals (p) of 30 and 480 seconds are used. The results show that with collision avoidance the size of the top-level overlay is the same as the ideal size, G. In contrast, the size of the first-level overlay in the presence of collisions is more than ten times the ideal size. Also, collision detection and correction creates a greater number of groups.

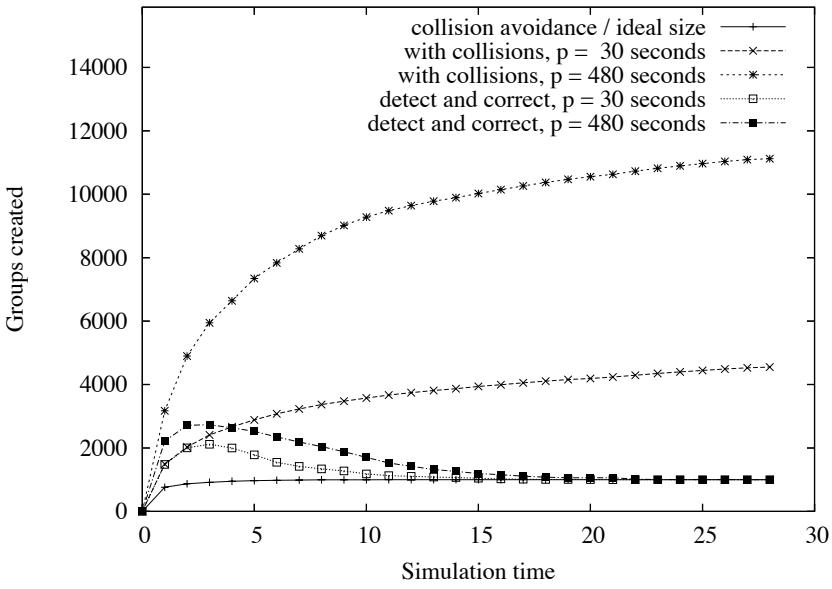


Figure 5. Comparison of Collision Algorithms for the Join Operation

A similar set of experiments varying different percentage of leave operations also reveals that collisions are avoided.

4.2. Bootstrap Time

In the presence of collisions, additional groups are created and lookups take more time to complete. Consequently, it takes more time for a node to join the system. This is not the case with a collision-free network, where the bootstrap time is smaller as the number of groups is the same as the number of resource types. A comparison of the average bootstrap time in which a node joins the overlay in the presence and absence of collisions is shown in Table 1.

p	***Detect and Resolve [6]***		***Collision avoidance***	
	G=**1000**	G=**2000**	G=**1000**	G=**2000**
30	0.8	0.8	0.5	0.55
60	0.9	0.9	0.5	0.6
120	1.0	1.0	0.6	0.6
240	1.0	1.0	0.6	0.7
480	1.0	1.2	0.7	0.8

(a) N = 50,000

p	***Detect and Resolve [6]***		***Collision avoidance***	
	G=**1000**	G=**2000**	G=**1000**	G=**2000**
30	0.8	0.8	0.5	0.6
60	0.9	0.9	0.5	0.6
120	0.9	1.1	0.5	0.6
240	0.9	1.0	0.5	0.6
480	0.9	1.2	0.6	0.7

(b) N = 100,000

Table 1. Average Bootstrap Time (seconds)

With less frequent stabilization (larger p), more stabilization rounds are required to correct the finger tables, and as a result the bootstrap time increases with p. An improvement of between 25% to 45% can be observed in the average time it takes a node to join the network by using our algorithms. Furthermore, upon entering the top-level overlay, the predecessor will have its successor pointer set to the new node. This assures instant availability of the resources, unlike [6], where one extra stabilization round is needed for that.

The worst effect of collisions arise when entire groups of peers with the same resource type are not visible to a lookup because a supernode cannot point to more than one group with the same identifier. This results in unsuccessful lookups, even if the resource exists in the network. By avoiding collisions, our algorithms also removes this effect.

4.3. Node Failures

Node failures are aspects of real peer-to-peer systems. In each experiment, we first simulated N join operations without collisions to set up an initial hierarchical Chord system. Next, we modeled the profile of peer-to-peer workload and varied the percentage of node failures. Assume P_j is the probability of join, P_f is the probability of node failure (both overlays) and the remaining are lookup operations.

p	P_j	P_f	***sn-fail***	***coll.***	***coll./sn-fail*** $(\%)$
30	0.3	0.5	71	2	0.03
30	0.3	0.1	152	13	0.09
30	0.3	0.2	49	78	1.59
120	0.3	0.5	37	41	1.10
120	0.3	0.1	66	76	1.15
120	0.3	0.2	45	46	1.02

Table 2. Collisions Due to Supernode Failures

Table 2 shows the results for a system initialized with 1,000 node joins, 500 resource types and 5,000 peer-to-peer operations with an arrival rate of one operation per second. The fourth and fifth columns in the table show the number of supernode failures (*sn-fail*) and the number of collisions (*coll.*) respectively. When a node from the second-level detected a supernode fail, it rejoined the network; the first peer to rejoin became the new supernode. As can be seen from Table 2, in the case of node failures, collisions are not eliminated. Still, the number of collisions is significantly smaller than the number of collisions in the "detect and resolve" scheme from [6].

The number of supernode failures is included in the total number of node failures. It is determined by comparing the generated number of join operations and the total number of join operations in the network (which included rejoin after supernode failures), with the number of generated node failures and the total number of virtual nodes that left the network (which included disbanded nodes). Comparing the number of supernode failures number with the number of collisions, we conclude that a failing supernode may produce, on average, at most two collisions.

5. Conclusions

Efficient lookup is an essential service in peer-to-peer applications. Lookup performance can be improved by using a hierarchical overlay network to organized shared resources. For example, to support resource allocation in a computational grid, resources can be grouped based on resource type in the overlay network for efficient resource discovery. Though the effect of collisions can be reduced by collision detection and correction [6], the message overhead cost is high.

We have presented a new collision avoidance approach where join and leave operations are collision free. Simulation experiments show that our collision-avoidance approach increases the lookup performance by maintaining the ideal size of the top-level overlay, and reduces the bootstrap time by up to 45%. Node failures are unplanned and collisions that arise due to node failure are therefore harder to address. Virtual nodes at the second-level overlay that fail do not lead to collisions, but collisions due to supernode failures still remain a challenge. However, our simulation results show that with more frequent stabilization, the probability of this type of collisions is small.

References

[1] Luc Onana Alima, Ali Ghodsi, and Seif Haridi, A framework for structured peer-to-peer overlay networks, Technical Report T2004-09, Swedish Institute of Computer Science, 2004.

[2] Luis Garcés-Erice, Ernst W. Biersack, Pascal Felber, Keith W. Ross, and Guillaume Urvoy-Keller, Hierarchical peer-to-peer systems, *Proceedings of the 9th International Euro-Par Conference,* LNCS 2790, pp. 1230-1239, Klagenfurt, Austria, 2003.

[3] Gnutella. http://www.gnutella.com/.

[4] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman, Skipnet: A scalable overlay network with practical locality properties, *Proceedings of USENIX Symposium on Internet Technologies and Systems*, pp. 113-126, Seattle, USA, 2003.

[5] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim, A survey and comparison of peer-to-peer overlay network schemes, *IEEE Communications Survey and Tutorials*, pp. 72-93, 2005.

[6] Verdi March, Yong Meng Teo, Hock-Beng Lim, Peter Eriksson, and Rassul Ayani, Collision detection and resolution in hierarchical peer-to-peer systems, *Proceedings of 30th IEEE Conference on Local Computer Networks,* IEEE Computer Society Press, pp. 2-9, Sydney, Australia, 2005.

[7] Alan Mislove, and Peter Druschel, Providing administrative control and autonomy in structured peer-to-peer overlays, *Proceedings of International Workshop on Peer-to-peer Systems,* Lecture Notes in Computer Science, vol. 3279, pp. 162-172, 2004.

[8] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, A scalable content-addressable network, *Computer Communication Review*, pp. 161-172, ACM Press, 2001.

[9] Antony Rowstron, and Peter Druschel, Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems, *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, pp. 329-350, Heidelberg, Germany, 2001.

[10] The Chord Simulator. http://pdos.csail.mit.edu/chord/sim.html.

[11] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, *Computer Communication Review*, Vol. 31(4), pp. 149-160, ACM Press, 2001.

[12] Yong Meng Teo, Verdi March, and Xianbing Wang, A DHT-based grid resource indexing and discovery scheme, Singapore-MIT Alliance Annual Symposium, 2005.

[13] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing, Technical Report UCB/CSD-1141, Department of Computer Science, University of California, Berkeley, 2001.