Mobile NodeID based P2P Algorithm for the Heterogeneous Network

Kyungbaek Kim and Daeyeon Park
Department of Electrical Engineering & Computer Science,
Division of Electrical Engineering,
Korea Advanced Institute of Science and Technology (KAIST),
373-1 Guseong-dong, Yuseong-gu, Daejeon, 305-701, Republic of Korea

E-mail: kbkim@sslab.kaist.ac.kr

Abstract

A lot of research papers discussed the Distributed Hash Table (DHT) based p2p algorithms to promise that idle resources may be efficiently harvested. However, p2p systems are composed of components with extremely heterogeneous availabilities and to handle churn, the system will generate the heavy information maintenance traffic to keep the efficiency of the DHT based p2p algorithms. In this paper, we suggest the mobile ID based p2p algorithm to reduce the overhead by exploiting the heterogeneity of participant nodes efficiently. Unlike the DHT based p2p algorithms, the node ID of a node changes according to its characteristic to support the p2p system efficiency and each nodes takes the different responsibility in accordance with its node ID. We classify nodes into the two types according to the characteristics of nodes : the reliable nodes and the leaf nodes. The reliable node has the load-balanced ID to balance the loads and the leaf nodes has the load-free ID to reduce the responsibility. We examine the efficiency of our p2p algorithm via a event driven simulation and show that the information maintenance traffic reduces and the routing process is more efficient.

1. Introduction

In these days, peer-to-peer systems have become an extremely popular platform for large-scale content sharing. Unlike client/server model based storage systems, which centralized the management of data in a few highly reliable servers, peer-to-peer storage systems distribute the burden of data storage and communications among tens of thousands of clients. The wide-spread attraction of this model arises from the promise that idle resources may be efficiently harvested to provide scalable storage services. A lot of research papers discussed the Distributed Hash Table (DHT) based p2p routing algorithms (Chord, Pastry, Tapestry and CAN) [5][3][7][2].

In contrast to traditional systems, peer-to-peer systems are composed of components with extremely heterogeneous availabilities - individually administered host PC's may be turned on and off, join and leave the system, have intermittent connectivity, and are constructed from low-cost low reliability components. For example, one recent study[4] of a popular peer-to-peer file sharing system found that the majority of peers had application-level availability rates of under 20 percent and only 20 percent nodes have server-like profiles. In such an environment, failure is no longer an exceptional event, but is a pervasive condition. At any point in time the majority of hosts in the system are unavailable and those hosts that are available may soon stop servicing requests.

A big issue in current DHT based p2p algorithms is the high overhead of maintaining DHT routing data structure and the stored data. When a node joins/leaves the system, the affected routing data structure on some existing nodes must be updated accordingly to reflect the change. Moreover, most p2p systems employ some form of the data redundancy to cope with failure and when the membership of nodes changes, these systems generate huge overhead of compulsory copies for the data availability. Especially, for nodes which join/leave the systems frequently, the p2p system will generate a lot of routing information update traffic and data copy traffic. It is not only increase the consumption of the network bandwidth, but also affect the efficiency of DHT based routing algorithms. Until now, DHT algorithms are not widely used in commercial systems yet, most p2p file sharing systems are still using non structured p2p mechanisms.

In this paper, we suggest the mobile ID based p2p algorithm to reduce the information maintenance overhead by exploiting the heterogeneity of participant nodes efficiently. Unlike the DHT based p2p algorithms, the node ID of a node changes according to its characteristic to support the p2p system efficiency and each nodes takes the different responsibility in accordance with its node ID. We classify nodes into the two types according to the characteristics of nodes : the reliable nodes and the leaf nodes. The reliable node which is the more stable and more reliable node acts as the more important role of the routing and the replication. The leaf node which joins/leaves very frequently acts as the simple role to minimize the information maintenance traffic. The reliable node has the load-balanced ID to balance the loads and the leaf nodes has the load-free ID to reduce the responsibility.

The reliable nodes are more stable and more reliable nodes and these nodes act as more important roles such as routing and replication. The reliable nodes have Load Balanced ID (LBID) which is evenly distributed and balances the workload of each reliable node. This LBID is dynamically assigned and the LBID routing table which help for routing to any reliable nodes is also organized when the LBID is assigned. The leaf nodes join/leave very frequently on the system and the majority of the participant nodes are these leaf nodes. These nodes act as simple roles such as servicing the request and helping the reliable nodes. The leaf nodes have Load Free ID (LFID) which makes the ID region of a leaf node as small as possible and reduces the effect of the dynamic membership change which increases the information maintenance overhead. According to these basic behaviors, because the frequently joining/leaving of nodes occurs as the leaf nodes which make little overhead, we can reduce the overhead of the whole p2p system and achieve more efficient routing without the frequent updates.

Moreover, we exploit the plentiful information of the availabilities and reduce the data management traffic for the dynamic membership. That is, more available nodes replicates data, more data traffic we can reduce when the nodes frequently join/leave.

This paper is organized as follow. In section 2, we describe the DHT based p2p algorithm and the other researches which try to reduce the overhead. Section 3 introduces the detail of the mobile ID based p2p algorithm. The simulation environment and performance evaluation are given in section 4. Finally conclude in section 5.

2. Background

There are many DHT based p2p algorithms such as Chord, Pastry, Tapestry and CAN [5][3][7][2]. Each node has a DHT which is a small routing table and any node can be reached in about $O(logN)$ routing hops where the N is the total number of nodes in the system. To achieve this efficient and bounded routing, there is some rules for the

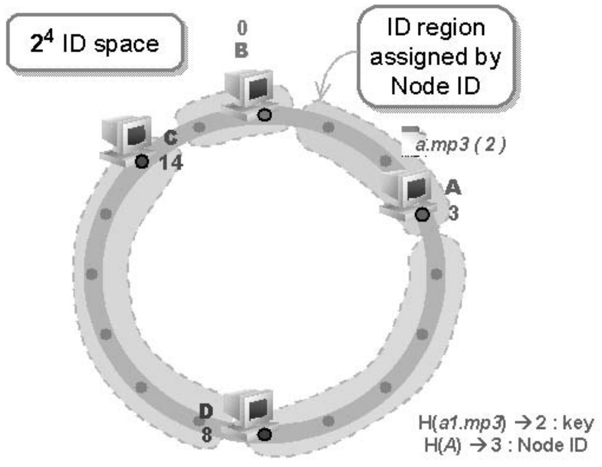


Figure 1. Overview for the general DHT based P2P algorithm

organizing the participant nodes. First of all, each node has a unique node ID which is taken by hashing any identifier of a node, and according to its node ID it maps on the ID space where the nodes and the objects are co-located with the node IDs or the keys which are the hashed value of the nodes or the objects. In the figure 1, the node id of node A is 3 and it maps on the position for 3. After the mapping of the node id, each node knows its ID region from the next position of its previous node ID to its node ID and each node should store and service the objects for its ID region. In the figure 1, node A takes its ID region from 1 to 3 because its node id is 3 and its previous node B locates on 0 and when any node want to get a.mp3 whose key is 2, node A gets the request for it.

Though these well-organized rules make the routing of the p2p system efficient and bounded, a big issue in current DHT based p2p algorithms is the high information maintenance overhead of maintaining DHT routing data structure and the stored data. Because its node id is already given by the hashing function and its position on ID space is already fixed, when a node joins/leaves the p2p system, the ID region of its neighbor nodes changes and the stored data should be copies for the new ID region to service the right and reliable object, and the update of the routing table is also needed. In figure 1, if node A leaves, the ID region of node D changes and the object from 1 to 3 should be copied from node A to node D. Moreover, the affected routing table which has the entry with node A must be updated. In this case, one recent research[4] of a popular p2p file sharing system found that 80 percent of total nodes of a p2p system join/leave very frequently and the majority of nodes has the application-level availability rate of under 20 percent.

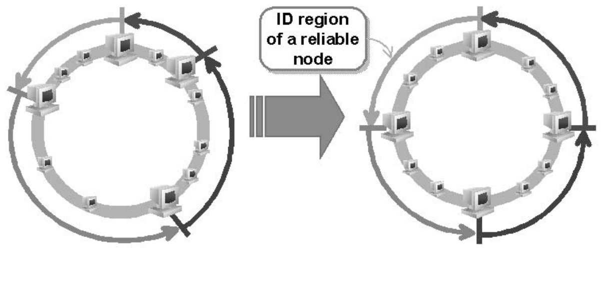


Figure 2. Ideal state

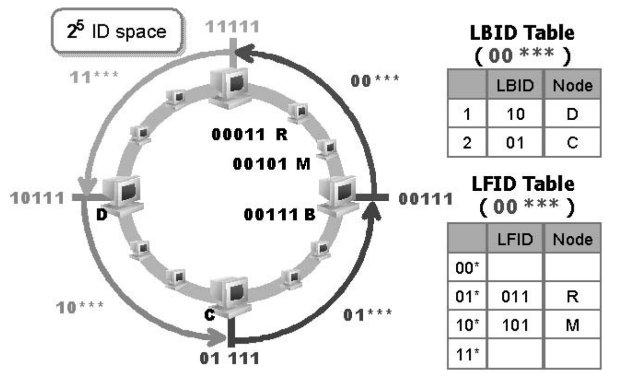
In such an environment, the information maintenance overhead is getting worse and this overhead discourages that the DHT based p2p systems are deployed to the real world.

Some researches emerged to prevent the information maintenance overhead by using the heterogeneity of participant nodes. In the paper [6], they manage the DHT which is certain amount of system routing information with the availability of each node which is evaluated during the time it joins the system. They proffer to add stable nodes into routing data structures instead of frequently join/leave nodes. The paper [1] tries to reduce the compulsory data copies for joining/leaving nodes with the node availability. They manage the availability of each data by evaluating the availability of each node which store the data. The common feature of these approaches is that the stable nodes take most system workload and this reduces the information maintenance overhead of the DHT based p2p algorithms. However, in these approaches, though the stable nodes get too much workload, they lack the explicit method which balances the workload of each stable node. In figure 2, the large sized computer means the stable and reliable node and the small sized computer presents the normal node which frequently joins/leaves. Because each reliable node already has the fixed node ID and the space between any two reliable nodes is unbalanced, each node get the unfair workload. Moreover, the fixed node id still affects the ID region of a node and the joining/leaving for a node makes the compulsory copies too. In our solution, the mobile ID based p2p algorithm, each reliable node gets the balanced ID region and workload and normal nodes which join/leave very frequently affects the information maintenance overhead little like the right side of the figure 2.

3. Mobile ID based P2P Algorithm

3.1 Overview

Previous DHT based p2p algorithms lack the explicit methods for exploiting the heterogeneous characteristics of participant nodes. The main reason of this lack is the static ID which makes the location of a node fixed on the ID space, and the system with the static ID is not flexible. We address this problem with the mobile ID which changes according to the characteristics of a node. We classify the participant nodes into two types : reliable nodes and leaf nodes. The reliable nodes are more reliable and more stable nodes and the leaf nodes join/leave very frequently. The node ID of a reliable node is well distributed on the ID space and makes that the each reliable nodes gets fair ID region and balanced loads. The leaf node gets the node id which makes its ID region as small as possible to minimize the information maintenance overhead for joining/leaving of it.



LBID Table
(00 ***)

	LBID	Node
1	10	D
2	01	C

LFID Table
(00 ***)

	LFID	Node
00*		
01*	011	R
10*	101	M
11*		

Figure 3. Overview of Mobile ID based P2P Algorithm

Figure 3 shows the overview of the p2p system that uses the mobile ID based p2p algorithm. The participant nodes are on the 2^5 ID space and the number of bits for a node ID is 5. The node ID is consist of the *Load-Balanced ID (LBID)* and the *Load-Free ID (LFID)* and ,in this example, the first 2 bits of a node ID means the LBID and the other 3 bits is for the LFID. In this figure, the large sized computer means the reliable node and the small sized computer means the leaf node. To distribute the participant nodes efficiently, we divide the ID space into many *sub-regions* which are the balanced ID regions. Each sub-region has one reliable node which represents this region and many leaf nodes which assist the reliable node. That is, the reliable node is mainly responsible for the objects for the sub-region and the leaf nodes service the objects for the small ID region which is assigned by both of their node id and the LFID table on behalf of the reliable node. For example, when any node wants to get an object whose key is 00001, the reliable node B takes the request, however when any node tries to get an object with 00010, the leaf node R takes the request to assist the reliable node, because the ID region of node R is from 00010, the start of the LFID slot to 00011, its node ID. All nodes on the same sub-region have the same LBID and they are identified by the LFID. The all bits of LFID for the re-

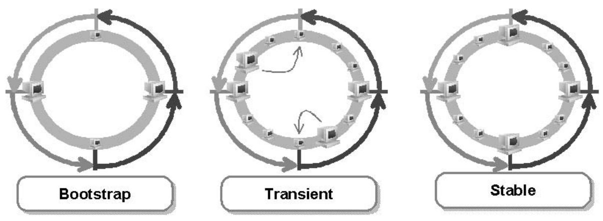


Figure 4. Three phases of LBID assignment

liable node are set to 1 and the LFIDs of other leaf nodes change according to the behaviors of them.

The LBID table and the LFID table is used to lookup the location of a node or an object. When a node joins, we get its static node ID by hashing its identifier. The first thing is to route to the right sub-region according to the LBID table. In this case, a node which gets a join request forwards it to the next node which is the node of the most prefix matched entry of the LBID table. After finding the sub-region, the LFID table assign the right LFID to the new node. This join process is only for the leaf nodes and in the next section, we show the detail of the whole join process. Moreover, when a node tries to lookup an object, it sends a lookup request with the object key which is its hashed value to any other participant node. Like the case of the join process, it forwards to the right sub-region by the LBID tables and find the node whose ID region is responsible for the key by the LFID tables. Basically, we use the hashed values of both of nodes and objects, but they are used only for finding the location of them. When the nodes join to the p2p system, their ids are newly assigned by the p2p system and change according to its characteristics for the nodes to be locate on fit places.

3.2 Mobile Node ID

Load Balanced ID is the identifier for the reliable node. Each reliable node is assigned this LBID and it is responsible for storing and servicing the request for the fair and balance ID region. For this, the LBID is well distributed and evenly divided. Moreover, because there is no routing information, we need LBID routing table which helps routing to any reliable nodes.

This LBID is assigned after a node finds the any reliable nodes. If there are not enough reliable nodes, new node is assigned the LBID and acts as the reliable node without its real characteristics. In this case, we call the bootstrap phase like figure 4. In this phase, a new reliable node is assigned the right LBID and create the LBID routing table according to its LBID. Each reliable node has the state information such as Join, Level, Full and Leaf. When the Join bit sets to 1, this node can process the join request and create new LBID for the new node. The Level bit is the depth value which means how many join requests is processed in this node, that is, how many routing entries are filled. The Full bit sets to 1 after the enough reliable nodes get the leaf nodes. The Leaf bit means the number of leaf nodes which is connected to a reliable node. According to these state information, LBID is assigned automatically and correctly.

```
If No Reliable Node
    $LBID_{new}$ = set all bits of LBID to 1
    Level = 1
Else
    If( Join ){
        If( Level < Levelthres ){
            // Accept Join
            $LBID_{new}$ = set exclusive bit of $Level_{th}$ bit of $LBID_{target}$
            LBID RT entry Update
            $Level_{th}$ RT entry = $LBID_{new}$
            Level++
        }
    }
    Else{
        If( Temporal Routing Entry )
            Forward Join request to this entry
        Else If( There is no temporal routing entry )
            If( sending node != last RT entry )
                Forward Join request to next RT entry of sending node
            Else
                // finalize
                Sending Notification of Finalization to all of RT entries
    }
```

Figure 5. Basic algorithm of the LBID assignment

The basic algorithm for the LBID assignment is in figure 5. When a node joins to the system and there is no reliable node, the new node has the new LBID whose all bits set to 1. Otherwise, when any reliable node gets a join request, it creates new LBID based on the two information which are its LBID and the Level bit. That is, the $level_{th}$ bit of LBID sets to the exclusive bit and this is the new LBID. This simple rule makes the difference of LBID of any two closest reliable nodes even and each reliable node gets the balanced and fair ID region. The LBID routing table which is used for routing to any reliable nodes is also organized when the new LBID is created. The basic rule is the N_{th} entry of the routing table has the node information whose N_{th} bit of LBID is exclusive to the owner's LBID. In figure 6, the node whose LBID is 0011 has the LBID routing table whose 1st entry has the information for the node C whose LBID is 1011 and 2nd entry has the information for the node B whose LBID is 0111. These bit-wise exclusive entries make the LBID routing table and any node can reach any other nodes. This LBID routing table have $logN$ of routing entries and the maximum routing hops are limited to $logN$, where N is the number of LBID bits. When there is not proper node information for a routing entry, we

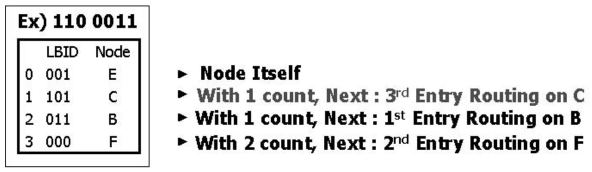

Ex) 110 0011
LBID Node
0 001 E
1 101 C
2 011 B
3 000 F
Node Itself
With 1 count, Next : 3rd Entry Routing on C
With 1 count, Next : 1st Entry Routing on B
With 2 count, Next : 2nd Entry Routing on F

Figure 6. LBID Routing Table and finalization rule

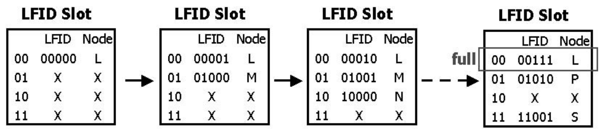


Figure 7. LFID slot and its change according to the lifetime

set the temporal routing entry. This temporal routing entry has the node information which does not matched but closest to the right node information. When the node which has the temporal routing entry get a join request, it forwards the join request to the temporal node which is ready to process join request. After this forwarding process, new node replaces the temporal routing entry and the LBID routing table is composed completely.

After the enough number of reliable nodes join, every reliable nodes set the Full bit to 1 and ready to get leaf nodes. To confirm that the all reliable nodes set Full bit to 1, we need finalize mechanism for the bootstrap phase. When a join request route to the reliable node by the LBID routing table and every node can not process the join request, the final node knows that the reliable nodes are assigned fully and the finalize mechanism should start. In this case, each node just does not know the whole of the reliable nodes, but only knows the $logN$ routing entries. According to this, one node can not notify to all node and needs the efficient and systematic notification. To achieve this notification, a node sends the notification with TTL count value to every nodes which is on the routing table. Like figure 6, the notification to the N_{th} routing entry has N-1 TTL count value, except the 1st entry whose notification has 1 TTL value. The target node which gets the notification reduces the TTL value by 1 and sends the notification to the $N - 1_{th}$ routing entry, where N is the position of the target node on the routing entry of the sending node. When the target node is 1st routing entry of the sending node, it sends the last notification message to the last routing entry.

Load Free ID is the identifier of the leaf nodes. Because each leaf nodes are unreliable nodes which join/leave very frequently, we should minimize the management cost for the effect of the dynamic membership change by assigning little load to leaf nodes. At first, LFID is close to 0 and when the access time of the leaf node increases, LFID also increases for the node to get more load and help the p2p system.

After the bootstrap phase, the transient phase starts. In this phase, each node joins in the system as the leaf nodes. A new node route to a reliable node by the LBID of the unique node key. When the Full bit of the reliable node is 1, this node processes the join request and increases the number of the Leaf bit by 1. To help assigning LFID, every reliable node has the LFID slot which divide the id space and each slot manage the leaf node information. The figure 7 shows the LFID slot and the change of them according to time. When node L joins, the first slot which is 00 slot assign the LFID 00000 to node L. After time pass, the LFID of node L change to 00001 and new node M gets 01000 for the second slot which is 01 slot, and so on. Each node is responsible for storing and requesting the data for the id space from the LFID 000 to the current LFID for each slot. When the LFID changes the data copy occurs, but this traffic is smaller than the management traffic of previous DHT, because these leaf nodes are free for data availability. Though each LFID increases according to the time, it can not increase more than the slot size.

3.3 Data Management

The general servers have very high availability and the data management on them is more simple and there is few overhead. However, in the p2p systems, unlike the general servers, the participants have very low availability. In this case, to preserve the availability of data, there are many replications for the data. These data are the basic p2p system information such as routing information and node information and the object data which is managed by each nodes which is responsible for some ID region. Previous DHT based p2p algorithms manage the replication by using the sequential node list such as the successor list for the chord and the leaf set for the pastry. This approaches take too much overhead to preserve the replications when the join/leave occurs frequently.

In our p2p algorithm, each reliable node knows not only the the availability of itself, but also the availabilities of the other nodes such as leaf nodes and LBID routing entry nodes which is managed by the reliable node. In this case, we exploit the plentiful information of the availabilities to preserve the data availability of each sub-region above the target availability and reduce the data management traffic according to the dynamic membership. That is, more available nodes replicates data, more data traffic we can reduce when the nodes frequently join/leave.

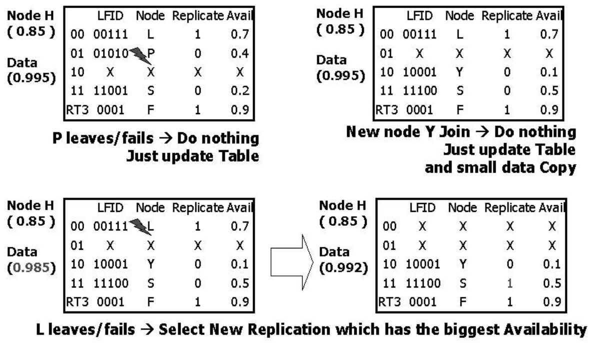
Availability is the prediction value how long a node is alive. We use the *Mean Time To Failure* and the *Mean Time To Recover* to estimate the node availability. MTTF is the average value how long a node is alive after it joins and MTTR is the average value how long a node is sleep after it leaves. We can get MTTF and MTTF by using the last join time and the last leave time. The average value of MTTF and MTTR is obtained by the sum of the weighted value estimation process. Any node gets its availability by using this process and notify the availability when it joins the p2p system. Moreover, after the node joins, it periodically estimates its availability and notifies the new value for the fresh information.

Data Availability means the total availability when the multiple nodes have the data. This availability is obtained by subtracting the probability of that all nodes which have the data leave from 1. That mean if only one node is alive, the data is available. The follow equation shows it.

Each reliable node manages the LBID routing table and the objects which is in the balanced ID region and knows the availability of the leaf nodes and routing entry nodes. To keep the data availability over the target availability, the reliable node manages the data replication set which is composed of more available nodes among the leaf nodes and routing entry nodes. That is, the reliable node replicates data to the more available leaf nodes or routing entry nodes. This behavior reduces the data management cost which is caused by the dynamic membership change. In the figure 8, when the leaf node P leaves, the reliable node just updates the LFID slot, and when the leaf node Y joins, the reliable node just updates the LFID slot and copies the data of new id range which the node Y is responsible for. That is, the joins/leaves of the leaf nodes make little management cost. However, when the node L which is the one of the data replication set and the total availability decrease below the threshold value, it needs too much cost such as data copies, LFID slot update and replication data update. But, this leave of reliable node occurs rarely and does not affect the management cost very much.

When the reliable node creates the data replication set, it selects more available nodes among the leaf nodes and routing entry nodes. In this case, most of the replicate nodes are the routing entry nodes which are the reliable nodes. However, these reliable nodes do too much jobs and we should prevent the additional overhead for them. Moreover, when the reliable nodes leave or fail, we need the alternative nodes which replace the left or failed reliable node with fast response time.

To achieve these, we select some leaf nodes for the data replication set. These more available leaf nodes is called the candidate nodes. These candidate nodes are included in the data replication set, and replicate the routing information such as the LBID routing table and the LFID slot and the


Node H (0.85)
Data (0.995)
P leaves/fails → Do nothing Just update Table
Node H (0.85)
Data (0.995)
New node Y Join → Do nothing Just update Table and small data Copy
Node H (0.85)
Data (0.985)
Node H (0.85)
Data (0.992)
L leaves/fails → Select New Replication which has the biggest Availability

	LFID	Node	Replicate	Avail
00	00111	L	1	0.7
01	01010	P	0	0.4
10	X	X	X	X
11	11001	S	0	0.2
RT3	0001	F	1	0.9

	LFID	Node	Replicate	Avail
00	00111	L	1	0.7
01	X	X	X	X
10	10001	Y	0	0.1
11	11100	S	0	0.5
RT3	0001	F	1	0.9

	LFID	Node	Replicate	Avail
00	00111	L	1	0.7
01	X	X	X	X
10	10001	Y	0	0.1
11	11100	S	0	0.5
RT3	0001	F	1	0.9

	LFID	Node	Replicate	Avail
00	X	X	X	X
01	X	X	X	X
10	10001	Y	0	0.1
11	11100	S	1	0.5
RT3	0001	F	1	0.9

Figure 8. Data Replication Set and its operation

data which is in the responsible range of the reliable node. When the reliable node leaves or fails, the most available candidate node replace it by changing leaf node id to reliable node id. This make the failure recovery process easier and faster.

4. Performance Evaluation

4.1 Simulation Setup

We make our p2p simulator which emulates the node behavior on the application layer. Each nodes has their own characteristics such as reliable nodes and leaf nodes. To make this dynamic characteristic, we use poison distribution whose average is 4, and to assign join/leave duration of a node, we use exponential distribution. We implement the DHT based p2p algorithms such as pastry, chord and our mobile ID based p2p algorithm. We use 160 bit ID space to identify nodes and the number of LBID bits change according to the number of the reliable nodes which are assigned by the total number of the participant nodes. In the next, DHT means the DHT based p2p algorithms and BA-N means the mobile ID based p2p algorithm which has the N percent of total nodes as the reliable nodes.

4.2 Dynamic Node Characteristics

Figure 9 shows the life distribution of participant. In this figure, the lifetime of 80% of total nodes is below 60% of total simulation time, that is, only 20% of total nodes have the reliable server-like profile. This characteristics of nodes are assigned when the nodes are created and never changed before the end of the simulation. This behavior decide both of the on-time which is the time for which the nodes are on the p2p system and the off-time which is the time for which the nodes are off. This distribution is similar to the

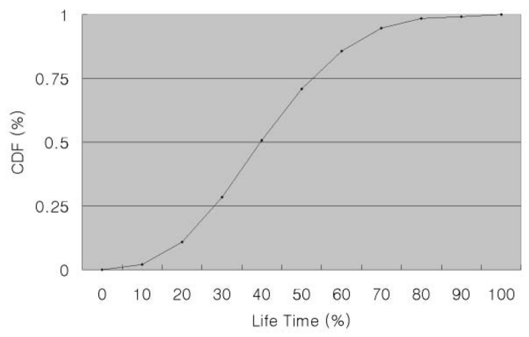


Figure 9. Life distribution of Nodes

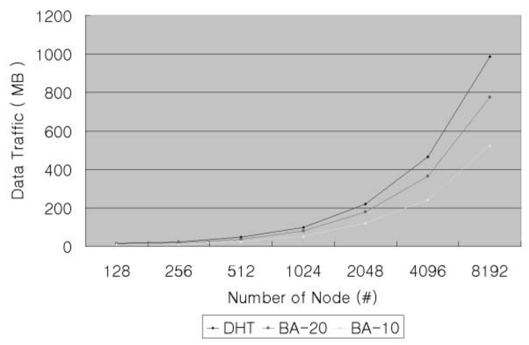


Figure 10. Comparison of Bandwidth Usage

real world. Recent researches measure the life distribution of the p2p nodes and show the similar graph like the figure 9. The next results are based on this characteristics.

4.3 Bandwidth Usage

The main problem of the current DHT p2p is the high management cost. In the figure 10, we show how the mobile ID based p2p algorithm reduces the management cost. To evaluate this cost, we assume that each node obtain same number of objects, that is, if the total number of nodes is 100 and the total number of objects is 10000, and if the total number of nodes is 200, the number of objects is 200000. In this case, the our p2p algorithm reduces the data management cost extremely. The main reason of this improvement is the behavior of leaf nodes. In DHT p2p, the frequent join/leave of leaf nodes cause the compulsory copies and update cost for routing information. However, in our p2p algorithm, the dynamic behavior of leaf nodes does not affect the data availability and the routing efficiency. According to these, the BA p2p can reduce more management traffic. Moreover BA-10 reduces more traffic than BA-20. On the same node characteristics, the BA-20 needs more reliable

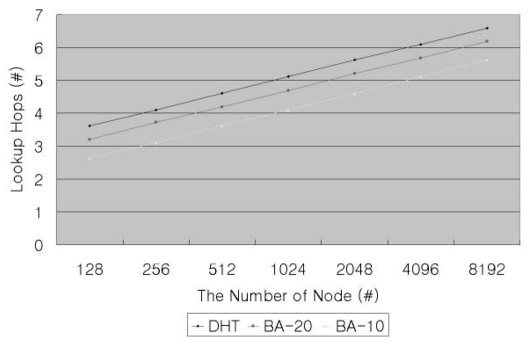


Figure 11. Comparison of Lookup Hops

nodes than BA-10, and the average availability of the reliable nodes of the BA-20 is less than the BA-10. In the BA-20, the transitions for the reliable nodes occur more than the BA-10 and BA-20 exhausts more network bandwidth than BA-10. To prevent this side effect, we need the adaptive method which manage the number of reliable nodes according to the state of the nodes and this work is our ongoing job.

4.4 Lookup Hops

In the p2p system, the lookup cost is also important parameter for the scalability because there are too many participants. Figure 11 shows the comparison of the lookup hops. For all algorithms, the lookup hops are proportion to the Log N, where N is the total number of nodes. The mobile ID based p2p algorithm performs more efficient lookup than normal DHT. The reason is that the our p2p algorithm uses the reliable nodes to route the lookup request and the number of these nodes are much less than the total nodes. These reliable nodes are more stable and more powerful than other nodes and they are durable nodes for the many routing requests. Additionally, the leaf nodes assist the reliable nodes to take the request for the ID region and the load of the reliable node are reasonable.

4.5 Load Balance

The figure 12 shows the load distribution for the total nodes. In this figure, we define the load as the number of lookup requests. As the nature of the previous DHT based p2p algorithm, the load are distributed to the whole of nodes by the shape of the normal distribution. This behavior causes the heavy information maintenance overhead because the nodes which join/leave very frequently can be responsible for the big ID region. On the other hand, in our mobile ID based p2p algorithm, the load distribution can be classified into the reliable nodes and the leaf nodes. About

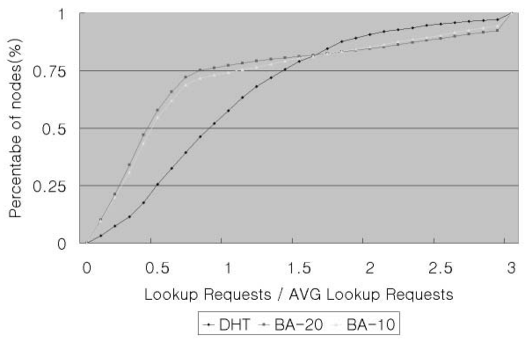


Figure 12. Load distribution for the total nodes

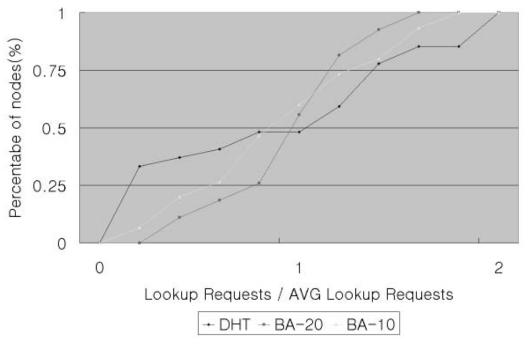


Figure 13. Load distribution for the reliable nodes

75 percent of nodes have less load than other nodes because these nodes act as a leaf node which join/leave frequently and they takes the responsible for small ID region which is assigned by the LFID. Otherwise, the reliable nodes take much more load because they are alive for a long time and represent for the sub-region. The reliable nodes gets more loads than the leaf nodes and we need to balance the load of the reliable nodes for the fairness. The figure 13 shows the load distribution for the reliable nodes. The mobile ID based p2p algorithm balances the load more than the previous DHT based p2p algorithm because the sub-region is distributed evenly and each reliable node is represent for a sub-region. Some jitters are appeared because the leaf nodes assist the reliable nodes and the transitions for the reliable nodes occurs. This feature which classifies the load according to nodes is very useful for the p2p system on the heterogeneous network which is consist of the various nodes such as servers, workstations and PCs. Some p2p approaches need the server-like components to increase the efficiency, and our algorithm can exploit these components easily and efficiently because the server-like nodes locates for the reliable nodes automatically.

5. Conclusions

In this paper, we suggest the mobile ID based p2p algorithm to reduce the information maintenance overhead by exploiting the heterogeneity of participant nodes efficiently. Unlike the DHT based p2p algorithms, the node ID of a node changes according to its characteristic to support the p2p system efficiency and each nodes takes the different responsibility in accordance with its node ID. The reliable node which is the more stable and more reliable node acts as the more important role of the routing and the replication. The leaf node which joins/leaves very frequently acts as the simple role to minimize the information maintenance traffic. The reliable node has the load-balanced ID to balance the loads and the leaf nodes has the load-free ID to reduce the responsibility. This algorithm is very good for the p2p system on the heterogeneous environment which is consist of the various kinds of nodes such as servers, workstations and PCs, because it locates the server-like nodes at the position for the reliable nodes and can exploit these nodes efficiently. However, our algorithm may over-provision for the reliable nodes and this may decreases the performance of our algorithm. The adaptive method for the whole state of nodes to keep the proper number of reliable nodes is our ongoing job.

References

[1] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker. Total recall: System support for automated availability management. In *NSDI*, pages 337–350, 2004.

[2] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.

[3] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.

[4] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *MMCN*, 2002.

[5] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[6] Z. Xu, R. Min, and Y. Hu. Reducing maintenance overhead in dht based peer-to-peer algorithms. In *Peer-to-Peer Computing*, pages 218–219, 2003.

[7] 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.