Computer Communications 32 (2009) 1062–1071

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Dynamic nodeID based heterogeneity aware p2p system

Kyungbaek Kim *

Computer Science Department, University of California, Irvine, CA 92697, USA

ARTICLE INFO

Article history:
Received 23 August 2007
Received in revised form 23 December 2008
Accepted 24 December 2008
Available online 14 January 2009

Keywords:
Distributed systems
Peer-to-peer
Heterogeneity

ABSTRACT

A lot of research papers discussed Distributed Hash Table (DHT) based p2p systems to promise that idle resources may be efficiently harvested. However, p2p systems are composed of components with extremely heterogeneous availabilities and they will generate heavy information maintenance traffic to keep the efficiency of DHT based p2p systems under churn. In this paper, we suggest a dynamic nodeID based heterogeneity aware p2p system to reduce the overhead by exploiting the heterogeneity of participant nodes efficiently. Unlike the DHT based p2p systems, the nodeID of a node changes on the fly according to its characteristic and each node takes different responsibility in accordance with its nodeID to support p2p systems efficiently. A nodeID is composed of Load-Balanced ID (LBID) which balances the loads of reliable nodes and Load-Free ID (LFID) which reduces the responsibility of normal nodes and eliminates compulsory maintenance overhead. We conduct an event-driven simulation and show that our p2p system reduces data maintenance traffic and makes routing process more efficient and more reliable.

1. Introduction

In these days, a peer-to-peer system has become an extremely popular platform for large-scale content sharing. Unlike client/server model based storage systems, which centralize 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 systems (CHORD, PASTRY, TAPESTRY and CAN) [3–6].

In contrast to traditional systems, peer-to-peer systems are composed of components with extremely heterogeneous availabilities. Individually administered hosts may be turned on and off at their will, have intermittent connectivity, and are constructed from low-cost low-reliable components. For example, one recent study [9] of a popular peer-to-peer file sharing system found that the majority of peers had application-level availability rates of under 20% and only 20% 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 available hosts may stop servicing requests soon.

A big issue in current DHT based p2p systems is the high overhead of maintaining DHT routing data structure and the stored data. When a node joins or leaves the system, the affected routing data structure of some existing nodes must be updated accordingly in order to reflect the change of the membership of nodes. Moreover, as most p2p systems employ some form of the data redundancy to cope with failure, when the membership of nodes changes, these systems generate huge overhead of compulsory copies for the data availability [7,8,2]. Especially, for nodes which join and leave the systems frequently, the p2p system will generate a lot of routing information update traffic and data copy traffic. It does not only increase the consumption of the network bandwidth, but also affects the efficiency of DHT based routing. Until now, DHT based p2p systems are not widely used in commercial systems yet, but most p2p file sharing systems are still using unstructured p2p mechanisms[15].

In this paper, we suggest a dynamic nodeID based heterogeneity aware p2p system to reduce the information maintenance overhead by exploiting the heterogeneity of participant nodes efficiently. The node heterogeneity makes the maintenance overhead heavy, but it also gives us the chance to improve the performance. That is, the more reliable and more stable nodes can handle much more jobs than normal nodes. If we can sort out the reliable nodes from all participant nodes during the system running, we can get the chance to reduce the management overhead. In this case, the nodeID of a node is coupled to its role of the p2p systems; if we use the static nodeID which is predefined by a node's unique identifier like the previous DHT based p2p systems, it is hard to change its position on the ID space as well as its role of p2p systems. In order to give the flexibility in the role of a node, in our p2p system the nodeID of a node changes on the fly based on its characteristic. This dynamic nodeID supports that each node takes the different responsibility in accordance with its nodeID to help the p2p system efficiently.

* Tel.: +1 949 812 2980.
E-mail addresses: kyungbak@uci.edu, kyungbaekkim@gmail.com

0140-3664/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2008.12.044

The nodeID consists of *Load-Balanced ID (LBID)* and *Load-Free ID (LFID)*. The nodeID is dynamically assigned based on nodes' behavior and this nodeID assignment is fully distributed without any central manager. According to the nodeID, every node is classified into two types, representative nodes and leaf nodes. The representative nodes are more stable and more reliable nodes servicing routing and replication for wider region on the ID space. On the other hand, the leaf nodes are the normal nodes which join and leave very frequently the system, and the majority of the participant nodes are these leaf nodes taking simple roles such as servicing the request and helping representative nodes. According to this classification, LBID used mainly to identify the representative nodes should be evenly distributed to balance their workloads which might be heavy, and LFID used mainly to identify the leaf nodes should make the ID region of a leaf node as small as possible to reduce the effect of their dynamic membership change. This nodeID assignment helps our p2p system reducing the overhead of data management as well as achieving more efficient routing without frequent updates.

Moreover, we exploit the plentiful information of the availabilities of nodes and reduce the data management traffic for the dynamic membership. That is, even if many nodes join and leave frequently, as more available nodes replicate data, more data traffic we can reduce.

This paper is organized as follows. In Section 2, we describe the DHT based p2p systems and the other researches which try to reduce the overhead. Section 3 introduces the detail of the dynamic nodeID based heterogeneity aware p2p system. The simulation environment and performance evaluation are given in Section 4. Finally we conclude in Section 5.

2. Background

There are many DHT based p2p systems such as CHORD, PASTRY, TAPESTRY and CAN [3–6]. Each node has a DHT which is a small routing table and any node can be reached in about $O(logN)$ routing hops where N is the total number of nodes in the system. To achieve this efficient and bounded routing, there are some rules for the organizing of the participant nodes. First of all, each node has a unique nodeID which is taken by hashing any unique identifier of a node, and according to its nodeID it maps on the ID space where nodes and objects are co-located with nodeIDs and keys which are the hashed values of nodes and objects, respectively. In turn, each node manages its DHT based on its own p2p system. In CHORD [3], for the ID space defined as a sequence of m bits, a DHT keeps at most m pointers to nodes which follow it in the ID space by 2^1, 2^2, and so on, up to 2^{m-1}. The ith entry in node n's DHT contains the first node that succeeds n by least 2^{i-1} in the ID space. With this DHT, CHORD routes to the target ID within $O(logN)$ hops. In PASTRY [4], a DHT is organized into $log_{2^b}N$ rows with 2^b-1 columns. The 2^b-1 columns in row i refer to a node whose nodeID matches the present node's nodeID in the first i digits, but whose $i+1$th digit has one of the 2^b-1 possible values other than the $i+1$th digit in the present node's nodeID. According to this DHT, PASTRY achieves $O(log_{2^b}N)$ routing.

Though these well-organized rules make the routing of DHT based p2p systems efficient and bounded, a big issue in the current DHT based p2p systems is the high information maintenance overhead of maintaining the DHT routing data structure and the stored data. Because its nodeID is already given by the hashing function with the unique and fixed identifier and its position on ID space is already fixed, when a node joins or leaves the p2p system, the ID regions being served by its neighbor nodes change. For the p2p system to support the correct and reliable services, the affected neighbor nodes should copy the responsible data for the new ID region and update their DHTs too. Moreover, many p2p systems use a simple replication method to cope with massive node failure and keep high data availability [2,7,8,10]. That is, if a p2p system needs N replicas to achieve acceptable data availability, each node has N or more replicas for its responsible ID range among its sequentially closest neighbor nodes such as successor list of CHORD and leafset of PASTRY. According to this simple replication method, when a node changes, N replicas should be updated to keep high data availability. In this case, one recent research [9] of a popular p2p file sharing system found that 80% of total nodes of a p2p system join and leave very frequently and the majority of nodes have the application-level availability rate of under 20%. 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 paper [11], they manage the DHT which is a certain amount of system routing information with the availability of each node which is evaluated during the time it joins the system. They prefer to add stable nodes into routing data structures instead of normal nodes which join and leave frequently. Paper [10] tries to reduce the compulsory data copies for the churn of nodes with the node availability. They manage the availability of each data by evaluating the availability of each node which stores 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 systems. However, in these approaches, though the stable nodes get too much workload, they lack for the explicit method which balances the workload of each stable node. Because each stable node already has the fixed nodeID and the space between any two stable nodes is unbalanced, each node gets an unfair workload [16]. Moreover, the fixed nodeID still affects the ID region of a node and the churn of a node makes the compulsory copies too. In our solution, the dynamic nodeID based heterogeneity aware p2p system, each stable node gets the balanced ID region and workload, and normal nodes which join and leave very frequently affects the information maintenance overhead little.

Some approaches [12,13] provide a hierarchical method to use the most reliable node as the supernode. However, because these methods assume that powerful supernodes already exist, they lack explicit methods to sort out the reliable nodes from the whole participant nodes. This makes the system inflexible and the problems of the fixed nodeID also exist. Moreover, in the paper [17], they explore how they manage superpeer nodes. But, their target system is the unstructured p2p systems and their method cannot apply to the DHT based p2p system directly. They cannot show how efficient the lookup process performs either, because of the nature of the unstructured p2p system. Our p2p system assigns a nodeId to a node dynamically and each node manages its routing table which is a kind of DHT tables. Each node can change its nodeID easily on the fly according to its characteristics and moves to the proper position easily. This behavior can help organizing the superpeer structure for the DHT based p2p systems.

3. Dynamic nodeID based heterogeneity aware p2p

3.1. Overview

Previous DHT based p2p systems lack explicit methods for exploiting the heterogeneous characteristics of participant nodes. The main reason of this lack is the static nodeID which fixes a location of a node on the ID space and it is hard to organize the p2p system flexibly. We address this problem with the dynamic nodeID.

When a node joins the p2p system, a nodeID is assigned to this node. After some time, this nodeID can be changed to relocate a node on the ID space. Stable nodes are relocated on more important position which is mainly responsible for routing and replication, and normal nodes are moved to trivial position mitigating the maintenance overhead induced by their joining or leaving.

Fig. 1 shows the stable state of our p2p system using the dynamic nodeID. Participant nodes are on the 2^5 ID space and the number of bits for a nodeID is 5. The general systems use 2^{128} ID space, but in this example we use only few bits for the easy explanation. To distribute participant nodes efficiently, we should divide the ID space into many *sub-regions* which are the balanced ID regions. Each sub-region has one representative node which represents this region and many leaf nodes which assist the representative node. That is, the representative node is mainly responsible for routing and servicing the objects for the sub-region and the leaf nodes service the objects for the small ID region which is the part of the sub-region allocated to each leaf node in accordance with its nodeID. Consequently, stable nodes and normal nodes are relocated on the representative nodes and the leaf nodes, respectively.

The nodeID consists of the *Load-Balanced ID (LBID)* and the *Load-Free ID (LFID)*. The LBID is the identifier for a sub-region and all nodes on the same sub-region have the same LBID. Each node has the LBID table consisting of m entries where m is the number of bits of the LBID. The ith entry of a node n's LBID table points a representative node whose ith bit of the LBID is only different from n's LBID. By using this LBID table, a node routes a request to a representative node which is responsible for the requested sub-region. The LFID is the identifier for a node in a sub-region. All the bits of a LFID of a representative node are set to 1. Except this representative node, other leaf nodes get different LFIDs related to their location. Each node has the LFID table composed of many LFID entries which are evenly distributed in a sub-region by the LFID prefix. The location of each entry is predefined by its LFID prefix and a new node gets a new LFID from one of empty entries. In Fig. 1, the first 2 bits of a nodeID is the LBID and the other 3 bits is the LFID. Nodes B(00111), R(00011) and M(00101) are included in the same sub-region(00^{***}) and their LBIDs are same(00). The LFID of the representative node B is 111 and the other leaf nodes get different LFIDs based on the LFID table.

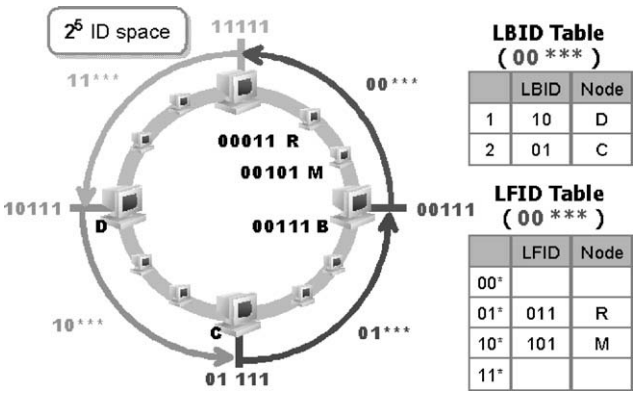


Fig. 1. Overview of dynamic nodeID based heterogeneity aware P2P system and LBID/LFID tables for node B(00111).

When a node tries to lookup an object, it sends a lookup request with the object key to any other participant node. This request is forwarded to the representative node for the sub-region which is responsible for the object key by referring the LBID tables, and finally forwarded to the node whose ID region is responsible for the object key by referring the LFID tables. A representative node is responsible for whole represented sub-region as its ID region, but a leaf node is responsible for the relatively short ID region which is defined by the LFID table. In Fig. 1, when a node wants to get an object whose key is 00001, the representative node B for the sub-region 00^{***} handles this request and finds a leaf node being responsible for this object key by referring the LFID table. But, there is no leaf node for the 00^{*} entry of the LFID table and the representative node B takes this request. However, when a node tries to get an object whose key is 00010, there is the leaf node R mapped to 01^{*} entry of the LFID table whose ID region is responsible for the key between 00010 and 00011, and R takes this request to assist the representative node B.

During the initial state of our p2p system, we allocate new nodes for representative nodes to be responsible for the sub-regions. In this case, we cannot use the static ID at all and every nodeID is assigned dynamically to distribute the load of each sub-region evenly. After this state, when a node joins, we get its static nodeID by hashing its identifier like the DHT based p2p systems. We only use this hashed value for a new node to find its sub-region, and according to this, the number of leaf nodes are distributed evenly to all sub-regions. To route to the sub-region, a node which gets a join request forwards it to the next node which is on the most prefix matched entry of the LBID table. After finding the sub-region, the LFID table assigns the right LFID to the new node. In the next section, we show the detail of the whole join process.

3.2. Dynamic nodeID

3.2.1. NodeID transformation

Fig. 2 shows the three phases of nodeID assignment as well as the nodeID transformation. At the first time, when there are few nodes in the p2p system and any one of sub-regions needs a representative node, this phase is called the *bootstrap phase*. In this phase, the LBID assignment performs to divide sub-regions evenly and to make up the LBID routing tables. When a new node joins, it

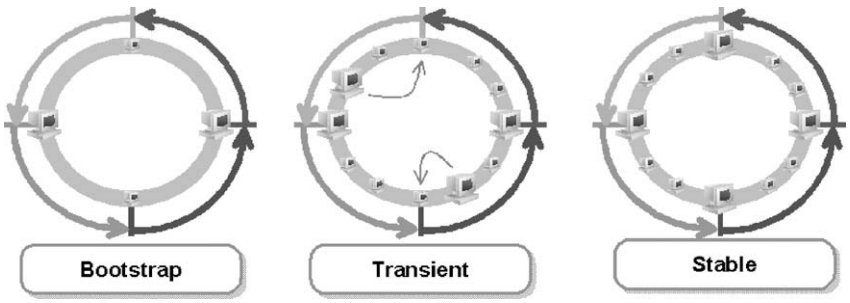


Fig. 2. Three phases of nodeID assignment.

should get a nodeID to represent a sub-region as a representative node regardless of its behavior and characteristic. According to this, in this phase some representative nodes are not really reliable and available nodes.

After the bootstrap phase, that is, when every sub-region has its representative node, the *transient phase* starts. Joining the system as leaf nodes to assist the representative nodes, new nodes get nodeIDs by performing the LFID assignment. When a new node joins, a join message with the node identifier is forwarded to the representative node whose LBID is same to the node identifier. This representative node picks up a new nodeID which is composed of a same LBID of the representative node and a new LFID from the LFID table. The new node gets this new nodeID and all nodes in the same sub-region updates their LFID tables with this new node. In this phase, when the representative node fails, the most reliable leaf node in the same sub-region substitutes it. That is, a representative node which is not reliable is displaced by a new reliable node which is in the same sub-region.

According to this assignment and the failure recovery, the p2p system becomes stable. That is, in the *stable phase*, more reliable nodes acts as representative nodes and the other nodes assist the representative nodes as leaf nodes. Like the transient phase, in the stable phase, a new nodeID is assigned by the LFID assignment and the same failure recovery is used.

3.2.2. NodeID assignment – LBID

During the LBID assignment, our p2p system cannot use the static nodeID which is generally used by other DHT p2p systems, but assigns the proper nodeID to each new node in order to act as a representative node without any help of any servers which manage proper nodeIDs. The nodeID of a new node is assigned by a representative node which gets a join message. The LBID of the new nodeID is generated by the representative node and all of the bits for the LFID of the new nodeID are set to 1 because it will become a representative node. In this LBID assignment, the new node also creates the LBID routing table according to its LBID.

Each representative node has the state information such as Join, Level, Full and Leaf. When the Join bit sets to 1, this node can process join requests and create new LBID for new nodes. The Level indicates the depth value which means how many join requests are processed in this node, that is, how many routing entries are filled. The Full bit sets to 1 after the enough representative nodes join the p2p system and they are ready to get the leaf nodes. The Leaf means the number of leaf nodes which is connected to a representative node. According to these state information, LBID is assigned automatically and correctly.

The basic algorithm for the LBID assignment is in Fig. 3. When a new node joins the p2p system and there is no representative node, it has the new LBID whose all bits set to 1. Otherwise, when any representative node gets a join request, it creates new LBID based on its LBID and its Level bit. To make a new LBID, the $Level_{th}$ leftmost bit of its LBID sets to the exclusive bit. This simple rule makes the difference of LBID of any two closest representative nodes even and each representative node takes the balanced and fair sub-region.

The LBID routing table which is used for routing to any representative node is also organized when a new LBID is created. The basic rule is the *i*th entry of a routing table has information of a node whose *i*th leftmost bit of LBID is exclusive to the owner's LBID. These bit-wise exclusive entries constitute the LBID routing table and a node can reach any other nodes through these LBID routing tables. In Fig. 4, the node whose LBID is 000 has routing entries 100, 010, 001. In this case, the routing entry 100 is the next routing point for LBID 1^{**} and the routing entry 010 is the next routing point for LBID 01^{*}. Like Fig. 4, this LBID routing table has *logm* of routing entries and the maximum routing hops are limited to *logm*, where *m* is the number of LBID bits.

When there is not proper node information for a routing entry, this entry becomes a *temporal routing entry*. In Fig. 5, the first entry of node B is C(101) which is the right one. However the second entry of node B is G(110) which does not fit to this entry and this entry is called a temporal routing entry. This temporal routing entry has the node information which does not matched, but this unmatched node is closer to the right node information. When a node has a temporal routing entry getting a join request, it forwards the join request to the temporal node which is ready to process a join request. After this forwarding process, the new node replaces the temporal routing entry with right routing entry and the LBID routing table is composed completely.

Fig. 5 shows a simple example of the LBID assignment. When the new node A joins and the target node B gets this join request, the node B makes a decision which it makes a new nodeID or forward to next nodes. The main point of the decision is the fulfillment of LBID routing table and the requested node tries to fill the LBID routing table with a new LBID. In this case, the node B can treat the join request, because its third routing entry is empty. According to the basic rule for the LBID routing table which is described in the previous section, the requested node B assigns a new nodeID, 000 to the new node A to fill the third routing entry. After the new node A gets the new nodeID, it also makes up the LBID routing table. To do this, when the requested node B responses the join request, it gives not only the new nodeID but also its new modified LBID routing table. The node A checks that each delivered routing entry is the right information for its LBID routing table according to the basic rule for the LBID routing table. If a routing entry is the right one, it is used, otherwise, the node tries to find the right information by requesting to the wrong one. For example, the third entry of the node A received from B is B(001) and it can be used by itself because of B(001) is the right one for the third entry of the node A. However, the second entry of the delivered information is G(110) which is the wrong one and the node A requests the proper information to the node G. Fortunately, the node G has information of the node D(010) which is the first routing entry of the node G and the node A can update its second routing entry with D(010). Moreover, unless one node finds the proper node information, it remains this entry and sets as the temporal routing entry. So, the first entry of the node A is the temporal routing entry, C(T)(101). After the new node updates its LBID routing table, it should advertise its information to every node of its LBID routing table to update the routing information.

We described the join process when the LBID routing table of target node is not full. If the LBID routing table is full, it should forward the request to the next node which can deal with the join request. First of all, if the requested node has any temporal routing entries, it forwards the join request to the node which is referred by this temporal routing entry. Otherwise, the join request is forwarded by referring to the LBID routing table with descending manner. At first, a node whose routing table is full forwards the join request through the first routing entry. If the next node which gets the join request also has the full routing table, it sends the join request to the node on the second routing entry because the first entry has the information of the requesting node. According to this descending routing mechanism, the join request is forward to a joinable representative node.

If the join request is forwarded by referring the last routing entry of any node, this node tries to find any node which can treat the join request among the whole representative nodes. In this case, each node does not know the whole representative nodes, but only knows the *m* routing entries where *m* is the number of the LBID bits. According to this, one nodes cannot notify to all other nodes by itself and the p2p system needs the efficient and systematic method to visit the whole representative nodes [14]. To achieve this, a node sends messages with TTL count value to every nodes

```
If No Representative Node
   LBIDnew = set all bits of LBID to 1
   Level++
Else
   If( !Full ){
      // Bootstrap phase
      If( Temporal Routing Entry )
          Forward Join request to this temporal entry
      If( Join ){
         // Accept Join
         LBIDnew = set exclusive bit of Levelth bit of LBIDtarget
         Levelth RT entry = LBIDnew
         Level++
         If( Level > Level_Thres) Join = 0
      }
      Else{
         If( sending node != last RT entry )
            Forward Join request to next RT entry of sending node
         Else
            If( Find_Joinable_Representative_Node() )
               Forward Join request to this representative node
            else
               // finalize
               Sending Notification of Finalization (Full = 1) to all nodes
      }
   }
   Else{
      // Transient & stable phase
      Find most prefix matched LBID RT entry with the static NodeID
      If( LBID match )
         LFID assignment with LFID Table
      Else
         Forward Join Request to the RT entry
   }
```

Fig. 3. Basic algorithm of the nodeID assignment.

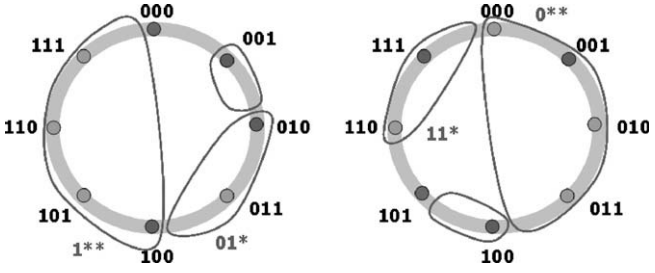
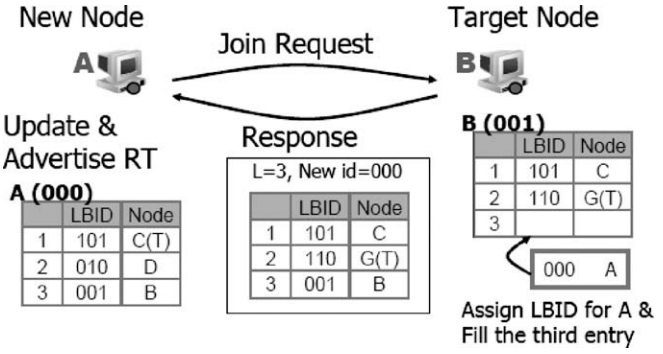


Fig. 4. The basic concept of LBID routing table (left:LBID 000, right:LBID 101).



	LBID	Node
1	101	C(T)
2	010	D
3	001	B

	LBID	Node
1	101	C
2	110	G(T)
3	001	B

	LBID	Node
1	101	C
2	110	G(T)
3		

000 A

Fig. 5. Simple example of the LBID assignment.

of the LBID routing table. Like Fig. 6, the message for the ith routing entry has $i-1$ TTL count value, except the 1st entry for whom the message has 1 TTL value. The node A which gets the message from the node B reduces the TTL value by 1 and the message is forwarded to the $L-1$th routing entry, where L is the position of the node A on the node B's LBID routing table. For example, if the node B is the 2nd routing entry of the node A, the node A sends the message to its first routing entry. However, if the node B is the first routing entry of the node A, it sends the message to its last routing entry. According to this mechanism, one node can visit the whole representative nodes. If there is no node which can process the join request, the LBID assignment finishes and the finalizing mechanism starts. This finalizing mechanism visits all of the representative nodes and sets the Full bit to 1 for every representative node to be ready to get the leaf nodes.

3.2.3. NodeID assignment – LFID

After the bootstrap phase, the transient phase starts. In this phase, each node joins in the system as leaf nodes. Because leaf

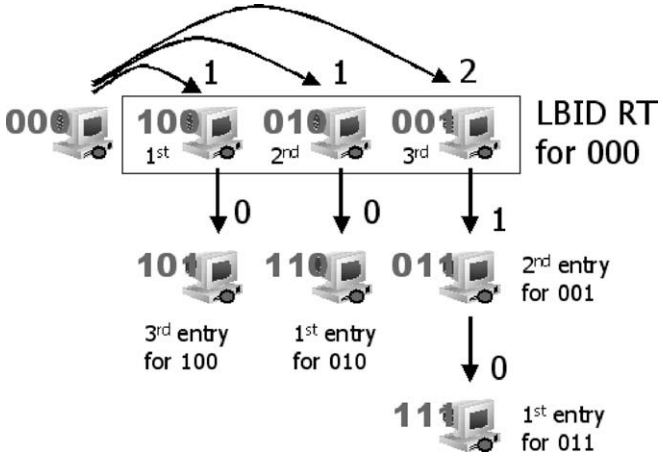


Fig. 6. Message flow to the whole of the representative nodes.

nodes are generally composed of the nodes which join and leave the p2p system frequently, our p2p system should minimize their responsibility to reduce the compulsory management cost which are due to the dynamic membership change such as updating the routing tables and copying the responsible data.

Basically the leaf node acts as an assistant for the representative node, and the LFID assignment is composed of two parts: finding a proper sub-region of a new node and assigning a new nodeID. When a new node joins the p2p system, it sends the join request to any one of participant nodes. In this case, unlike the LBID assignment, the join request contains the static nodeID obtained by hashing unique identifier of the new node in order to balance the number of the leaf nodes for each sub-region. This join request routes to the representative node whose LBID is the same to the LBID of the static nodeID of the new node. This routing process is the most prefix matched method which is generally used by other DHT based p2p systems. That is, when a node gets a join request, it forwards this request to the LBID routing table entry whose LBID is most prefix matched to the LBID of the static nodeID of the requesting node unless the LBID of this static nodeID is perfectly matched to the LBID of its nodeID.

After the join request of a new node is forwarded to the proper sub-region, the representative node of this sub-region assigns a new nodeID to the new node. The LBID of this new nodeID is same to the representative node and its new LFID is assigned by referring the LFID table which identifies the responsible ID region of leaf nodes. The LFID is composed of many LFID entries which are evenly distributed in the sub-region and each leaf node fills each LFID entry. In Fig. 7, all nodes under the sub-region whose LBID is 110 have a LFID table like that. This LFID table has four entries which are identified by the prefix such as 00^*, 01^*, 10^* and 11^*. When a leaf node joins, an empty entry is selected to be filled with this new leaf node, and in order to create a new LFID for this new leaf node the LFID bits are set to 1 except the prefix bits of the selected LFID entry. According to this assignment, in Fig. 7, the node L's LFID is 00111 and the node M's LFID is 01111. However, there is one exception of generating a new LFID. That is, the LFID of the last entry whose prefix is 11^* is not 11111, but 11110. According to the LBID assignment, if all bits of LFID are set to 1, this nodeID refers to the representative node. So, to avoid the nodeID confliction, all bits of the LFID of the last entry are set to 1 except its last bit. When a leaf node joins and there is no empty entry, one of entries separates into two entries and the new entry is assigned to the new leaf node. After a new node gets its new nodeID, it updates its LBID table and LFID table which are delivered from the representative node. Also it announces its join to other leaf nodes on the same sub-region to update their LFID tables.

LFID Table (LBID 110)

	LFID	Node
00^*	00111	L
01^*	01111	M
10^*	10111	N
11^*	11110	P

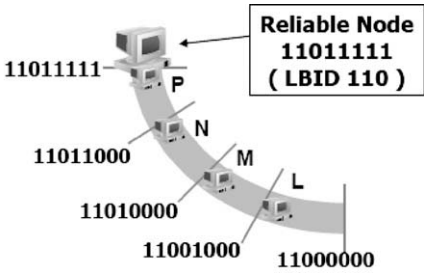


Fig. 7. LFID table and node assignment (LBID 110).

A leaf node is responsible for servicing the objects whose LFIDs have same prefix to its corresponding LFID prefix. For example, the node L is responsible for servicing the objects whose LFIDs are between 00000 and 00111. The LFID prefix is divided evenly and each ID region of each leaf node is balanced within a sub-region. As the number of leaf nodes increases, an ID region decreases. According to this, our system can mitigate the overhead which is due to the dynamic membership change of leaf nodes.

3.3. Lookup

The lookup process of our p2p system is similar to the LFID assignment. Fig. 8 shows the simple example of the lookup operation. The normal node Y wants to find "d3.avi" file and the static object key of d3.avi is 10000101. First of all, node Y checks its LBID routing table with the object key 10000101. If the LBID of the object key is matched to the node Y, node Y tries to find the right LFID entry from its LFID table. However, in this case, the LBID of the object key is different from node Y and node Y finds the next routing node from its LBID routing table. The node Y selects a next routing node as the most prefix matched LBID entry to the object key. In this figure, the next routing node is H and H also selects next routing node D by using most prefix matched method.

Finally, lookup request is forwarded to node D whose LBID is 100 which is same LBID of the object key. After finding right sub-region, the representative node D checks its LFID table. If the proper LFID entry is empty, the representative node handles this lookup request. Otherwise, the representative node forwards this lookup request to the leaf node allocated to the right LFID entry. In this figure, the object key 10000101 is managed by the first LFID entry, node L, whose prefix is 00^*. So, the lookup request is forwarded to the node L and node L returns the requested object "d3.avi" to node Y. According to this assistant, the representative nodes lessen their responsible loads.

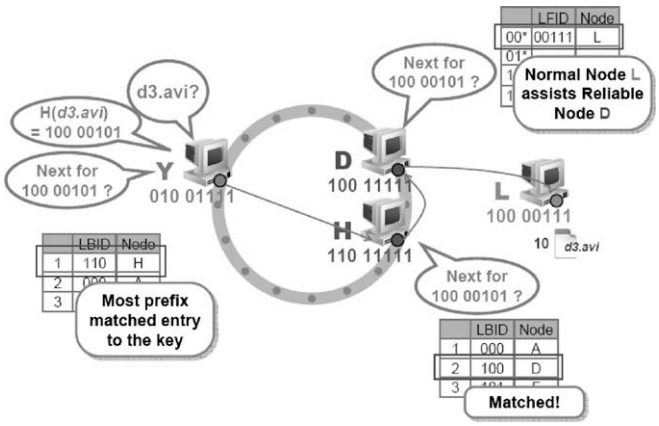


Fig. 8. Lookup operation.

The lookup operation is composed of the LBID lookup and the LFID lookup. The LBID lookup takes $O(logN_r)$ cost where N_r is the number of the representative nodes and the LFID lookup takes $O(1)$ cost. Generally, N_r is about $0.2 * N$ where N is the number of total nodes because of the characteristics of participant nodes in the p2p systems. Consequently, the total lookup cost is proportional to $O(logN)$ where N is the number of total nodes. That is, this lookup operation is scalable.

3.4. Data management

The general servers have very high availability and the data management on them is more simple having little overhead. However, unlike the general servers, in the p2p systems 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 as well as the object data which is managed by each node which is responsible for some ID region. Previous DHT based p2p systems manage the replication by using the sequential node list such as the successor list of CHORD [8] and the leaf set of PASTRY [7,2]. This approaches take too much overhead to preserve the level of replications when nodes join and leave frequently.

In our p2p system, each representative node knows not only the availability of itself, but also the availabilities of the other nodes such as leaf nodes and LBID routing entry nodes which are managed by the representative node like Fig. 9. In this case, the plentiful information of the availabilities is exploited to reduce the data management traffic against the dynamic membership change. That is, more available nodes replicate data, more data traffic the p2p system can reduce when the other nodes frequently join and leave. Basically, more available than other leaf nodes, the representative nodes on LBID routing entries may replicate data with very high probability. However, being mainly responsible for each sub-region, each representative node takes too many jobs. Consequently, even though the representative nodes is highly available, the number of the representative nodes for replicating data is limited. Each representative node selects few other representative nodes and preferentially picks up the replicas among the leaf nodes. In this case, the leaf nodes having very low availability such as node R in Fig. 9 are not selected as replicas.

3.4.1. Availability prediction

According to other researches [9,11], the long-lived nodes generally have the large bandwidth as well as the high computing power, and we assume that the node availability is the prediction value how long a node is alive. The *Mean Time To Failure* and the *Mean Time To Recover* are used 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 offline after it leaves. To estimate these values, the *Time To Failure* and the *Time To Recover* are measured like Fig. 10. To do this, each online node stores the current time ($T_n^{current}$), the join time (T_n^{join}), the previous leave time (T_{n-1}^{leave}), and the previous join time (T_{n-1}^{join}). When joining the system, a node gets TTF_n by using the previous join time and the previous leave time, updating $MTTF_n$ with this TTF_n like the Eq. (2). One more thing for TTF_n is that having no chance to update its $MTTF_n$ until rejoining the system, a long-lived node updates TTF_n and $MTTF_n$ periodically by using the join time and the current time like Eq. (1).

$$TTF_n = \begin{cases} T_{n-1}^{leave} - T_{n-1}^{join} & \text{If measuring at join time} \\ T_n^{current} - T_n^{join} & \text{If measuring periodically} \end{cases} \quad (1)$$

$$MTTF_n = \alpha * TTF_n + (1-\alpha) * MTTF_{n-1}, \quad (0 < \alpha < 1). \quad (2)$$

To update $MTTR_n$, when joining the system, a node calculates TTR_n with the join time and the previous leave time like Eq. (3). Similar to $MTTF_n$, $MTTR_n$ is obtained by the weighted average of TTR_n like Eq. (4).

$$TTR_n = T_n^{join} - T_{n-1}^{leave} \quad (3)$$

$$MTTR_n = \beta * TTR_n + (1-\beta) * MTTR_{n-1}, \quad (0 < \beta < 1). \quad (4)$$

The node availability is computed with *MTTF* and *MTTR* like Eq. (5). Because of updating *MTTF*, a node obtains its node availability at join time or periodically. When getting its node availability, a leaf node notifies its node availability to the nodes on the same sub-region including the representative node and other leaf nodes. When a representative node obtains its node availability, it notifies its node availability to the leaf nodes on the same sub-region as well as the representative nodes on the LBID routing table. Each node exploits node availability to select the proper location for replicating data.

$$NodeAvailability(A_i) = \frac{MTTF}{MTTF + MTTR}. \quad (5)$$

3.4.2. Data replication set

In the p2p system, nodes join and leave at their will and some nodes leave the system without any notice. Even if this dynamic environment, the p2p system should support the high level of data availability by replicating data to multiple nodes. In this case, the data availability means the total availability when the multiple nodes restore the data. We call the set of multiple nodes restoring data the *replication set*. The data availability is calculated by Eq. (6), that is, the probability of simultaneous failure of all nodes in the replication set subtracted from 1. It means if only one node is alive, the data is available. To get this probability, the node availability of each node in the replication set is exploited.

$$DataAvailability(D_i) = 1 - \prod_{i \in R}(1 - Ai), \quad R \text{ is the replication set} \quad (6)$$

Each representative node is responsible for managing the LBID routing table as well as storing the objects which are mapped to the represented sub-region. To keep the data availability of the stored

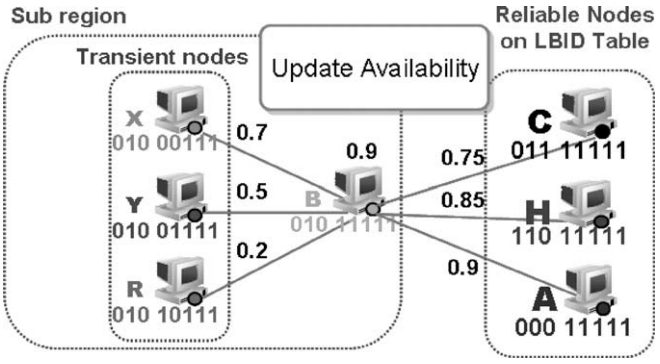


Fig. 9. Node availability update.

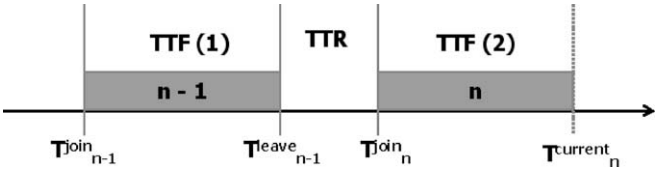


Fig. 10. Node availability prediction.

information over the target data availability with small maintenance cost, the representative node manages the data replication set indicating that which nodes replicate its data. When selecting the nodes for the replication set, a representative node chooses more available nodes among the leaf nodes and the other representative nodes of the LBID routing table.

Fig. 11 shows the operation of the data replication and its target data availability is 0.99. In this figure, Node H (0.85) means its node availability is 0.85 and Data (0.995) means its data availability is 0.995. When the leaf node P leaves, because this node is not a node replicating data, the representative node H just updates the LFID slot. When the leaf node Y joins, the representative node just update the LFID slot and copies the data of new id range for the node Y. That is, since the leaf nodes with relatively low node availability hardly become a member of the replication set, their frequent changes do not affect the management cost too much. However, when the node L, one of the nodes replicating data, leaves, the total data availability decreases below the target data availability ($0.985 < 0.99$). In this case, the representative node H should select a new node replicating data to keep the target data availability.

When selecting a new replica which is a new node replicating data, a node firstly check if one another representative node replicates the data. If not, it selects one representative node as a new replica. Otherwise, it does not select any other representative nodes as a new replica to prevent that replicas converge on few reliable nodes, because other representative nodes already have many jobs for other sub-regions. After that, it chooses the most available node among leaf nodes on the same sub-region as a new replica. This selecting process is performed until the new data availability is greater than the target availability. In Fig. 11, the node H selects the node S as a new replica, because there is already a representative node F for a replica and the node S is the most available node among leaf nodes in the sub-region 10^{***}.

3.5. Update messages

When churn occurs, affected nodes should update its routing tables such as DHT, LBID table and LFID table to ensure the correctness and the efficiency of the lookup process. In general DHT based p2p such as CHORD and PASTRY, when one node joins or leaves, each affected node updates its DHT. It takes $O(logN)^2$ cost because there are $O(logN)$ affected nodes and it takes $O(logN)$ cost to find one node.

However, in our p2p system, the update cost depends on the type of the leaving node. When a leaf node joins or leaves, the affected nodes are the representative node and the leaf nodes on the same sub-region. Because the leaf node is identified by the LFID table, each affected node updates its LFID table only. That is, the update cost is $O(N_t)$, where N_t is the average number of leaf nodes for a sub-region. When a representative node joins or leaves, it takes more cost than a leaf node. This change of a representative node affects the leaf nodes on the same sub-region and the other representative nodes of its LBID table. Moreover, it also affects the leaf nodes of other sub-region which is represented by the representative nodes of its LBID table, because every leaf nodes contain the same LBID table of its representative node. According to these, the update cost is $O(N_t) * (O(logN_r) + 1)$, where N_t is the average number of leaf nodes for a sub-region and N_r is the number of representative nodes.

The update cost for representative nodes are bigger than leaf nodes. However, the most of churn is caused by the leaf nodes. Consequently, the majority of update cost becomes about $O(N_t)$ for our p2p system. In this case, if $O(N_t)$ is smaller than $O(logN)^2$, our p2p system updates its routing table more efficiently than the DHT based p2p.

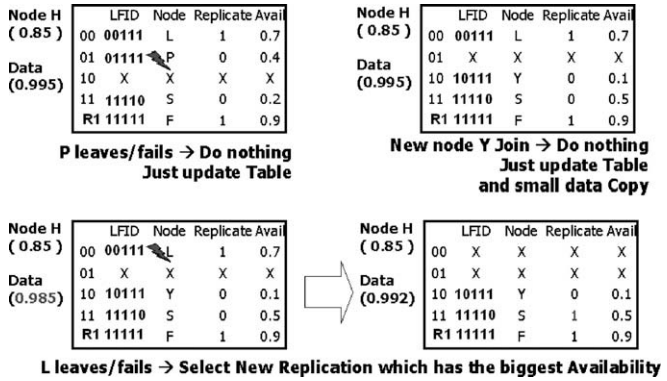


Fig. 11. Data replication set and its operation.

4. Performance evaluation

4.1. Simulation setup

We carried out event-driven simulations to evaluate our dynamic nodeID based heterogeneity aware p2p system and the results show that it outperforms the previous DHT based p2p systems such as CHORD and PASTRY. We made a p2p simulator which emulates behavior of nodes on the application layer by using C. This simulator contains PASTRY as a DHT based p2p system as well as our dynamic nodeID based heterogeneity aware p2p system. We brought the basic PASTRY module from the FreePastry [1]. 160 bit ID space is used to identify nodes and the total number of nodes is varied from 512 to 8192. The target data availability is 0.999 which is same to the 10 replicas whose average node availability is 0.5. The DHT based p2p system does not consider the behavior of nodes and sets the node availability of every node to the average value for total nodes.

The participant nodes join and leave the p2p system individually and each node has a different lifetime as well as various durations for online and offline. To generate this dynamic churn, we use the Poisson distribution to identify the average lifetime of each node. In turn, the on/off duration of a node is assigned by using the exponential distribution whose mean is inherited from the preset value by the Poisson distribution. As the mean of the Poisson distribution decreases, most nodes have short lifetime and frequently join and leave the p2p system. On the other hand, as the mean of the Poisson distribution increases, most nodes are reliable and have long lifetime. In this simulation, setting the mean of Poisson distribution to 4, we figured out that the lifetime distribution is very similar to the measured result of the other research [9].

The comparative systems are follows: DHTP means the DHT based p2p system, especially PASTRY and BAP(1:N_t) means our new p2p system in which each representative node obtains about N_t leaf nodes. That is, the number of representative nodes are $1/N_t$*(total number of nodes). For example, when the total number of nodes is 2048, BAP(1:64) has 32 representative nodes and its LBID bit is 5. At the same case, BAP(1:16) has 128 representative nodes and its LBID bit is 7. Table 1 shows the variation of LBID for each BAP according to the number of nodes. When N_t is big,

Table 1
LBID variation for each BAP.

Number of nodes	512	1024	2048	4096	8192
BAP(1:16)	5	6	7	8	9
BAP(1:32)	4	5	6	7	8
BAP(1:64)	3	4	5	6	7

the number of representative nodes is small and LBID bit is also small. When the number of node increases, LBID also increases to keep the number of leaf nodes for each representative node.

We measure and compare the performance of both systems at the stable phase. To pass over both of the bootstrap phase and the transient phase, we start gathering data after 500 tick times from the beginning of our simulator. The average node arrival rate for 1 tick time depends on the total number of participant nodes. If the total number of nodes is 2048, the average node arrival rate is about 26 and if the number is 8192, the arrival rate is about 110. At any case, 500 tick times are enough to assume that both the bootstrap phase and the transient phase pass.

4.2. Data maintenance traffic

The main problem of the current DHT p2p is the high management cost, especially the data traffic such as objects and replicas to keep the high data availability. Fig. 12 shows the comparison of the data traffic usage. To evaluate this, we assume that each node obtains averagely same number of objects, that is, if the total number of nodes is 100, the total number of objects is 100,000, and if the total number of nodes is 200, the number of objects is 200,000. The number of total nodes is varied from 512 to 8192. The object size is averagely 2 MByte. In this case, our p2p reduces the data management cost extremely. The main reason of this improvement is the behavior of leaf nodes. In DHT based p2p, the frequent joins or leaves of leaf nodes cause the huge compulsory copies. However, in BAP, the dynamic behavior of leaf nodes hardly affects the data availability and this churn cannot generate much data traffic. According to these, our p2p system can reduce much more data management traffic to keep the same level of the high data availability than the DHT based p2p system.

One of facts, we should note, is when the number of representative nodes increases, the management traffic also increases. That is, BAP(1:64) reduces more traffic than BAP(1:32) and BAP(1:16). On the same node characteristics, the BAP(1:32) needs more representative nodes than BAP(1:64), and the average availability of the representative nodes of BAP(1:32) is less than BAP(1:64). In BAP(1:32), the failures of representative nodes occur more than BAP(1:64) and BAP(1:32) exhausts more network bandwidth than BAP(1:64).

4.3. Lookup hops

In the p2p system, the lookup cost is also important parameter for the scalability because there are too many participants. Fig. 13 shows the comparison of the lookup hops. For all systems, the lookup hops are proportion to the *logN*, where *N* is the total number of nodes. However, BAP performs more efficient lookup than normal DHTP. The reason is that our p2p system mainly uses the representative nodes to route the lookup request and the number of these nodes are much less than the total nodes. These representative nodes are more stable and more powerful than other nodes and they are durable nodes for the many routing requests.

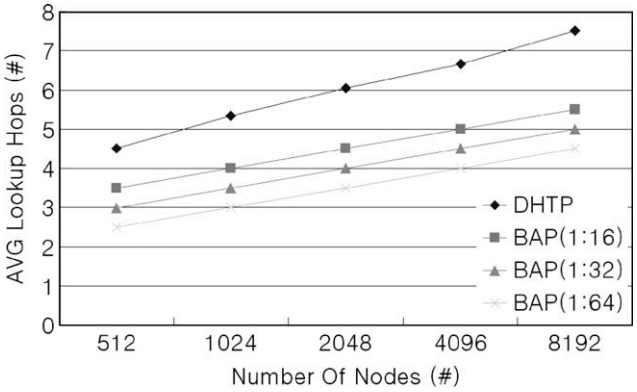


Fig. 13. Comparison of lookup hops.

Moreover, the slope of the DHTP is steeper than the slopes of other BAPs. In general DHT based p2p systems the routing table is filled with unreliable nodes. When these nodes leave, many routing entries can contain wrong and failed routing information. This wrong information due to many routing faults and it takes times to route to the right destination. Sometimes, this failure misleads the route and the number of lookup hops increases. According to this fact, the slope of DHTP becomes steeper. However, in BAP, every routing entry is filled with reliable nodes and there are very few routing faults. This makes the slopes of our p2p system gentler than DHT based p2p systems.

Moreover, when the number of LBID bits is big, the average routing hop is also big. That is, BAP(1:64) has less representative nodes than BAP(1:32) and BAP(1:64) also has less routing table than BAP(1:32). According to this small routing table, BAP(1:64) needs small routing hops than BAP(1:32).

4.4. Control traffic

Fig. 14 shows the needed control messages for the various BAP. There are three types of control messages: the LFID table updates, the LBID table updates and the Join Messages. The number of the LFID tables update is affected by the number of the leaf nodes for

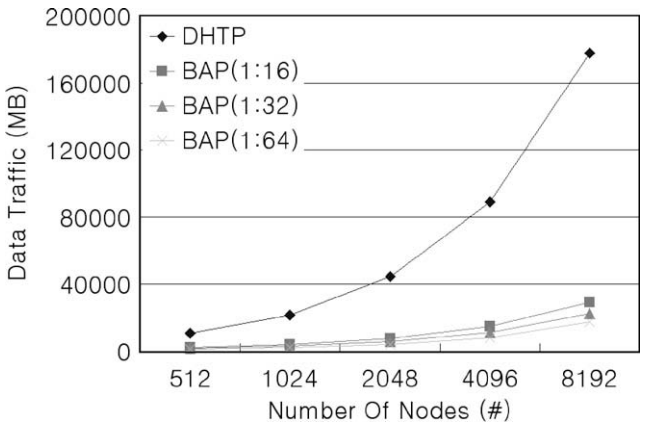


Fig. 12. Comparison of total data traffic usage.

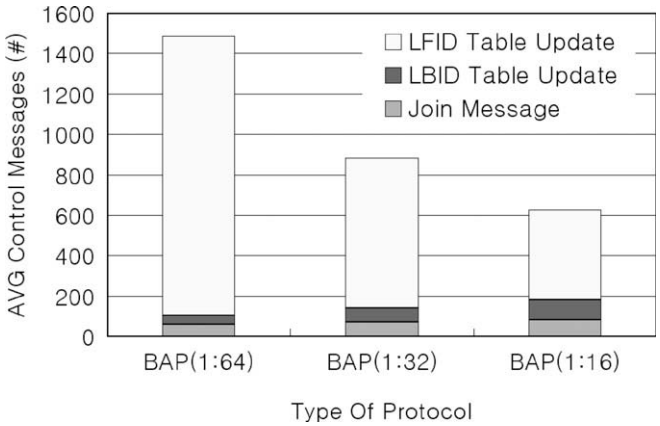


Fig. 14. Average control messages for each BAP, node number = 2048.

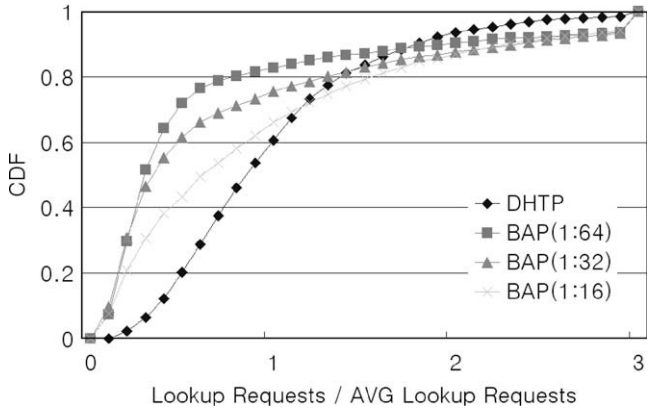


Fig. 15. Lookup distribution for the total nodes.

a representative node. The number of the LBID table updates and the number of the Join messages are affected by the number of the representative nodes. When the number of leaf nodes increases, the number of the LFID table updates increases. As the same way, when the number of representative nodes increases, both of the number of the LBID table updates and the number of the Join messages increase.

In this figure, BAP(1:64) needs most control messages and BAP(1:16) uses least messages. This is because the majority of the control messages for BAP are the LFID table updates. BAP(1:64) keeps the average number of leaf nodes in about 64 and in BAP(1:16), the average leaf nodes for a representative node is about 16. That is, BAP(1:64) has four times more leaf nodes than BAP(1:16). On the other side, BAP(1:16) has four times more representative nodes than BAP(1:64). According to this, the average number of the LFID table updates of BAP(1:64) is about four times more than BAP(1:16). Consequently, even if in BAP(1:64) the number of the LBID table updates and the number of the Join messages are less than BAP(1:16), BAP(1:64) needs about 2.5 times more control messages than BAP(1:16).

4.5. Load balance

Fig. 15 shows the lookup distribution for the total nodes. In this figure, we define the lookup load of a node as the number of its lookup requests divided by the average number of lookup requests of whole nodes. As the nature of the previous DHT based p2p system, the load is distributed to the whole of nodes by the shape of the normal distribution and the average load of nodes is nearly 1. This behavior causes the heavy information maintenance overhead because the nodes which join and leave very frequently can be responsible for the relatively big ID region. On the other hand, in our p2p, the type of the load distribution can be classified into the representative nodes and the leaf nodes. About 75% of nodes have fewer loads than other nodes because these nodes act as leaf nodes which join and leave frequently and they takes the responsible for small ID region which is assigned by the LFID. The average load of the leaf nodes is about 0.4 and these nodes are distributed uniformly. Otherwise, the representative nodes take much more loads because they are alive for a long time and represent for the sub-region. The average load of these nodes is about 2.

This feature which classifies the load according to the characteristics of 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. Our p2p system can exploit these powerful components efficiently and easily because the server-like nodes locates for the representative nodes automatically.

5. Conclusions

In this paper, we suggest the dynamic nodeID based heterogeneity aware p2p system to reduce the information maintenance overhead by exploiting the heterogeneity of participant nodes efficiently. Unlike the DHT based p2p systems, the nodeID of a node changes on the fly based on its characteristic in order to support the p2p system efficiently and each nodes takes the different responsibility in accordance with its nodeID. Because the representative node is more reliable and more stable, it acts as the more important role of the routing and the replication. The leaf node which joins and leaves very frequently acts as the simple role to reduce the information maintenance traffic. The representative nodes are mainly identified by the Load-Balanced ID to balance the loads and the leaf nodes are mainly identified by the Load-Free ID to reduce the responsibility and eliminate the compulsory maintenance overhead.

This system 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 representative nodes automatically and can exploit these nodes efficiently and easily. However, our system may over-provision for the representative nodes and this may decreases the performance of our system. The adaptive method for the whole state of nodes to keep the proper number of representative nodes is our ongoing job.

References

[1] FreePastry. Available from: <http://freepastry.org/FreePastry/>.
[2] K. Kim, D. Park, Efficient and scalable client clustering for web proxy cache, IEICE Transactions on Information and Systems E86-D (9) (2003).
[3] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of ACM SIGCOMM 2001, August 2001.
[4] A. Rowstron, P. Druschel, Pastry scalable decentralized object location and routing for large-scale peer-to-peer systems, in: Proceedings of the International Conference on Distributed Systems Platforms (Middleware), November 2001.
[5] B.Y. Zhao, J. Kubiatowicz, A. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing, UCB Technical Report UCB/CSD-01-114, 2001.
[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of ACM SIGCOMM 2001, 2001.
[7] P. Druschel, A. Rowstron, PAST: a large-scale persistent peer-to-peer storage utility, in: Proceedings of HotOS VIII, May 2001.
[8] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, Wide-area cooperative storage with CFS, in: Proceedings of SOSP 2001, October 2001.
[9] S. Saroiu et al., A measurement study of peer-to-peer file sharing systems, in: Proceedings of MMCN 2002, 2002.
[10] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, G.M. Voelker, Total recall: system support for automated availability management, in: Proceedings of NSDI 2004, 2004.
[11] Z. Xu, R. Min, Y. Hu, Reducing maintenance overhead in DHT based peer-to-peer algorithms, in: Proceedings of P2P 2003, 2003
[12] L. Carcs-Erice, E.W. Biersack, P. Felber, K.W. Ross, G. Urvoy-Keller, Hierarchical peer-to-peer systems, in: Proceedings of Euro-Par 2003, 2003.
[13] B. Yang, H. Garcia-Molina, Designing a super-peer network, in: Proceedings of ICDE 2003, 2003.
[14] S. El-Ansary et al., Efficient broadcast in structured P2P networks, in: Proceedings of IPTPS 2003.
[15] Q. Lv et al., Search and replication in unstructured peer-to-peer networks, in: Proceedings of ACM ICS 2002.
[16] B. Godfrey et al., Load balancing in dynamic structured P2P systems, in: Proceedings of Infocom 2004.
[17] L. Xia, Z. Zhuang, Y. Liu, Dynamic layer management in superpeer architectures, in: IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 11, November 2005.