Capability-Aware Object Management based on Skip List in Large-Scale Heterogeneous P2P Networks

TAKASHI TOMIMOTO, TAKUJI TACHIBANA, KENJI SUGIMOTO
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192
JAPAN
E-mail: {takuji-t,kenji}@is.naist.jp

Abstract: - In this paper, we propose an object management method in large-scale heterogeneous P2P networks. In the proposed method, objects can be stored in and searched from nodes by considering node's capabilities. The proposed system is based on skip list, and two identifications are utilized; TypeID and HashID. The TypeID is used to specify each node's capabilities, on the other hand, HashID is used for providing load balancing among nodes with similar capabilities. According to the two identifications, message routing for storing and searching objects is performed. We evaluate the performance of the proposed method by simulation, and we investigate the effectiveness of the method. Numerical examples show that the proposed method can manage objects based on node's capabilities and can provide the scalability when the number of nodes is large and the difference in node's capabilities is large.

Key-Words: - Peer-to-Peer networks, Skip list, DHT, Heterogeneous networks, Object management

1 Introduction

Peer-to-peer (P2P) networks have been widely used over the Internet for various applications such as Internet telephony [1], distributed data storages [2], data streaming [3], and online games [4].The numbers of P2P users and objects increase year by year, and in order to manage a large number of users and objects, structured P2P networks based on distributed hash table (DHT) have emerged, including Chord [5] and Pastry [6].

In the DHT-based P2P networks, a key is assigned to each node and each object. An object is stored in a node whose key is closest to the object's key. In addition, an objective is searched from a node whose key is closest to the object's key. By using a hash function for the key assignment, the load balancing can be provided among participating nodes.

Moreover, the difference in node's capabilities becomes large year by year [7,8]. For example, multimedia file sharing, traffic/weather prediction, and stress level monitoring of buildings are just beginning to be utilized by mobile hand-held devices or sensor nodes [9-11]. In the future, it is indispensable to build large-scale heterogeneous P2P networks.

Fig. 1 shows a large-scale heterogeneous P2P network where several types of objects are utilized.

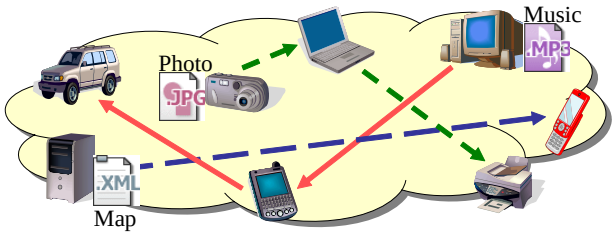


Fig. 1: Heterogeneous P2P network for multiple applications.

In such P2P network, it is expected that each object is stored in a node whose capabilities satisfy the requirement of the object. In addition, it is expected that each object is searched from a node whose capabilities satisfy the requirement of the object. For example, music files should be stored in high-performance computers and temperature data should be searched from sensor nodes. Hence, objects should be managed by considering node's capabilities.

In this paper, we propose an object management method that is used in large-scale heterogeneous P2P networks. In the proposed method, objects can be stored in and searched from nodes by considering node's capabilities. This proposed method is based on skip list [12].

In the capability-aware object management, two identifications are utilized; TypeID and HashID.

TypeID is assigned to each node according to its own capabilities such as forwarding speed, data storage, mobility, and availability. It is also assigned to each object according to capabilities of node which the object should be stored in or searched from. On the other hand, HashID is assigned to each node and each object with a hash function.

When an object is stored in or searched from a node according to pre-specified TypeID and HashID, message routing is performed based on the two identifications. First, according to TypeID, a message is routed to one of the nodes whose TypeIDs are the same as the pre-specified TypeID. Then, the message is routed according to HashID, and finally the message is received by a node whose HashID is closest to the pre-specified HashID. By using the proposed method, it is expected that each object can be managed according to node's capabilities based on TypeID and that the load balancing can be provided based on HashID. In addition, this method has high scalability for heterogeneous environments.

We evaluate the performance of the proposed method by simulation, and we investigate the effectiveness of this method. Moreover, we evaluate its performance over physical networks. Note that our proposed routing algorithms and simulation results have been updated from those in our previous works [13].

The rest of the paper is organized as follows. Section 2 describes P2P system based on skip list, and Section 3 explains the proposed capability-aware object management. Numerical examples are shown in Section 4 and finally, conclusions are presented in Section 5.

2 P2P System based on Skip List

Skip list has been proposed as a randomized balanced tree data structure [12]. In this data structure, each data has a specific key and every data is sorted by key.

Fig. 2 shows an example of skip list with three levels. As shown in this figure, every data is doubly linked in increasing order by key at level 0, and at level i $(i>0)$, each data in level $i-1$ appears in level i with probability p $(0 \le p \le 1)$. The lists at higher level allow the sequence of data to be traversed quickly. Therefore, when data with a particular key is searched, the searching process is performed at higher level.

Recently, data structures similar to the skip list have been utilized in large-scale P2P networks; skip graph [14] and SkipNet [15]. P2P nodes in the two networks correspond to data in the skip list. As

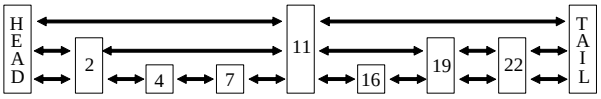


Fig. 2: Skip list with three levels.

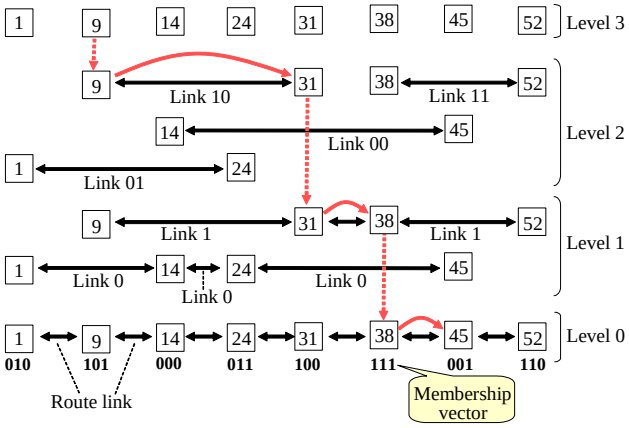


Fig. 3: Routing mechanisms for a skip graph.

shown in Fig. 3, in a skip graph with h $(h \ge 1)$ levels, each P2P node has a specific key and all nodes are doubly linked in increasing order by key at level 0, as is the case with the skip list. At level i $(1 \le i \le h)$, there are one or more doubly linked lists and each node belongs to one of the lists. Each node has an identification called *membership vector* in addition to its own key, and a list where a node belongs at level i is determined by its own membership vector.

On the other hand, in the SkipNet, ring data structure is used instead of list data structure. Fig. 4 shows an example of SkipNet with four levels. As shown in this figure, there are one or more rings at each level, and rings at level i are obtained by splitting a ring at level $i-1$ into two disjoint sets. Each node has two identifications called *name ID* and *numeric ID*, and a node belongs to one of the rings at each level according to its own numeric ID. In every ring, nodes are sorted by name ID.

In both the skip graph and SkipNet, message routing is performed according to the two identifications. A message can be transmitted to a node with pre-specified identifications. By using the identifications effectively, the skip graph and SkipNet can provide several functions such as object storage, path locality, and constrained load balancing for large-scale P2P networks.

3 Proposed Method

In this paper, we propose a capability-aware object management in large-scale heterogeneous P2P networks. The proposed method is based on the skip

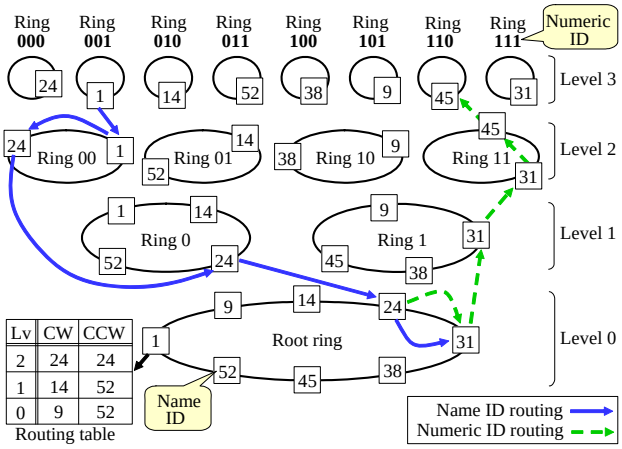


Fig. 4: Data structure and message routing in SkipNet.

list in order to consider node's capabilities, and ring data structure is used as is the case with SkipNet for the efficient message routing. In the following, we explain our proposed method in terms of ID assignment, node structure, message routing, and an example of use.

3.1 ID Assignment

The proposed method utilizes two identifications called *TypeID* and *HashID*. TypeID and HashID correspond to name ID and numeric ID in SkipNet, respectively.

The TypeID is assigned to each node for specifying its capabilities such as forwarding capability, data-storage capability, mobility, and availability. In this paper, for the simplicity, we assume that four-digit TypeID (w, x, y, z) is used as follows.

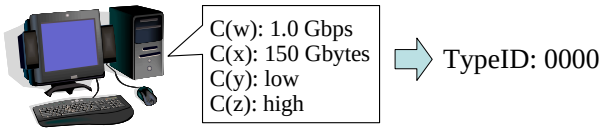
- First digit (w) : Forwarding capability
- Second digit (x) : Data-storage capability
- Third digit (y) : Mobility
- Fourth digit (z) : Availability

Here, the number of digits of TypeID can be changed depending on how many capabilities should be considered, and a capability can be denoted with more than one digit in order to represent the capability in more detail.

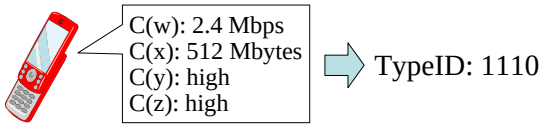
Table 1 shows an example about how each digit number is determined. When the forwarding capability $C(w)$ is 1.0 Gbps, data-storage capability $C(x)$ is 150 Gbytes, mobility $C(y)$ is low, and availability $C(z)$ is high for a high-performance computer, TypeID is set to $wxyz = 0000$ (see Fig. 5(a)). In the case of a cell-phone whose $C(w)$ is 2.4 Mbps, $C(x)$ is 512 Mbytes, $C(y)$ is high, and $C(z)$ is high, TypeID of this node is set to $wxyz = 1110$ (see Fig. 5(b)).

Table 1: Assignment of four-digit TypeID.

	0	1
w	$C(w) \geq 1$ Gbps	$C(w) < 1$ Gbps
x	$C(x) \geq 50$ Gbytes	$C(x) < 50$ Gbytes
y	$C(y) = \text{low}$	$C(y) = \text{high}$
z	$C(z) = \text{high}$	$C(z) = \text{low}$



(a) High-performance computer.



(b) Cell-phone.

Fig. 5: TypeID assignment.

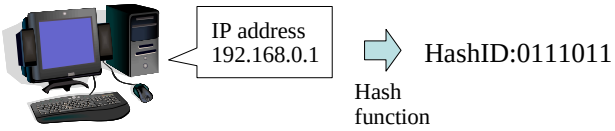


Fig. 6: HashID assignment.

On the other hand, HashID is assigned to each node by applying a collision-resistant hash function to its IP address or others. For example, HashID of the high-performance computer in Fig. 5(a) is set to 0111011 based on its IP address (see Fig. 6). Moreover, a character ":" is used for each node in order to discriminate its own TypeID and HashID as *TypeID:HashID*. In the case of Fig. 5(a) and Fig. 6, this high-performance computer is denoted as 0000:0111011.

3.2 Node Structure

Fig. 7 shows a node structure for the capability-aware object management in a case of four-digit TypeID. In this structure, there are one or more rings at each level, and rings at level i are obtained by splitting a ring at level $i-1$ into multiple disjoint sets. The number of levels is $H+1$ when the number of digits of HashID is H .

Each node belongs to a ring at every level so that i -digits prefix of HashID is shared by other nodes.

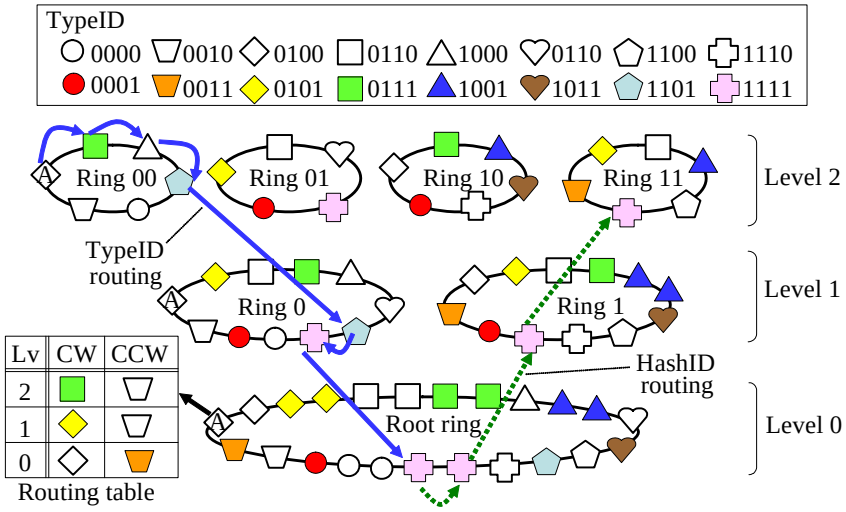


Fig. 7: Node structure and message routing based on TypeID and HashID.

For example, a node whose the first digit of HashID is 1 belongs to Ring 1 at level 1.

Because nodes are sorted by TypeID at each ring, nodes with the same TypeID, i.e., similar capabilities, are located in a ring sequentially. Here, the number of nodes in a top-level ring is one if all nodes have different HashID. As the level becomes low, the number of nodes in a ring becomes large. Therefore, by using higher-level rings, messages are traversed to a node quickly.

Under this node structure, each node has a routing table which includes neighbor nodes at each level (see node A in Fig. 7). This routing table is used for message routing, which is explained in the next subsection.

3.3 Message Routing

In the proposed method, message routing for storing and searching an object is performed based on TypeID and HashID. Moreover, message routing for node join and departure procedures is also performed based on the two identifications. Figs. 8 and 9 show two message routing algorithms. In the following, we explain the routing algorithms in a case where a source node sends a message to a destination node.

At first, the source node starts message routing based on TypeID as shown in Fig. 8 (solid lines in Fig. 7). If the TypeID of the source node shares some common prefixes with that of the destination node, the source node determines a direction of message routing from both its own TypeID and destination node's TypeID (see (A) of Fig. 8). In Fig. 8, clockwise direction is denoted as *true* and counter clockwise direction is denoted as *false*. On the other hand, if the source TypeID and the destination

```
SendMsg(TypeID, HashID, msg) {
  if(LongestPrefix(TypeID, localNode.TypeID) == 0)   … (A)
    msg.dir = RandomDirection();
  else if(TypeID < localNode.TypeID)
    msg.dir = false;  // CounterClockwise
  else
    msg.dir = true; // Clockwise

  msg.TypeID = TypeID;  msg.HashID = HashID;
  RouteByTypeID(msg);
}

RouteByTypeID(msg) {
  h = localNode.MaxRoutingTableHeight;  … (B)
  while(h >= 0) {
    if(LiesBetween(msg.dir, localNode.TypeID,
       localNode.RoutingTable.[h][msg.dir].TypeID ,
       msg.TypeID ) == false )                          … (C)
    {
      h--;
      continue;
    }

    if(LiesBetween(msg.dir, localNode.TypeID,
       localNode.RoutingTable.[h][msg.dir].TypeID ,
       msg.TypeID ) == true )                           … (D)
    {
       NextCandidateNode = localNode.RoutingTable.[h][msg.dir];
       if(CheckIfAlreadyVisited(msg, NextCandidateNode))  … (E)
       {
          h--;
          continue;
       }
       msg.AlreadyVisited(localNode);
       SendtoNode(NextCandidateNode, msg); … (F)
       return;
    }
  }

  if( localNode.TypeID != msg.TypeID ) …  (G)
  {
     NegativeAck(msg);
     return;
  }

  msg.dir = true;
  RouteByHashID(msg);  … (H)
}
```

Fig. 8: Routing algorithm based on TypeID.

```
RouteByHashID(msg) {
  if( msg.HashID == localNode.HashID  ||
                              msg.FinalDestination == true ) {   … (A)
     DeliverMessage(msg);
     return;
  }

  if(msg.StartNode != null &&  localNode == msg.startNode ) {  … (B)
     msg.FinalDestination = true;
      SendtoNode(msg.bestNode);
     return;
  }

  h = CommonPrefixLen(msg.HashID, localNode.HashID); … (C)

  if( h > msg.ringLvl) {
     msg.ringLvl = h;
     msg.startNode = msg.bestNode = localNode;
  }
  if( abs(localNode.HashID - msg.HashID) < abs(msg.bestNode.HashID -
        msg.HashID) ){  … (D)
     msg.bestNode = localNode;
  }

  if( localNode.RoutingTable[h][msg.dir].TypeID == msg.TypeID ){  … (E)
     SendtoNode(msg, localNode.RoutingTable[h][msg.dir]);
  }
  else if( msg.dir == true ){  … (F)
     msg.FinalDestination = false;
     msg.startNode = null;
     msg.dir = false;
     SendtoNode(msg, localNode);
  }
  else if( msg.dir == false ) {   … (G)
     msg.FinalDestination(true);
     SendtoNode(msg, msg.bestNode);
  }
}
```

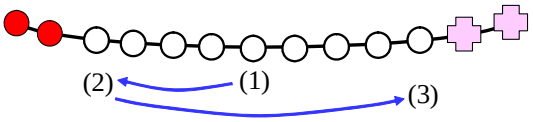
Fig. 9: Routing algorithm based on HashID.

TypeID have no common prefix, a routing direction is selected at random.

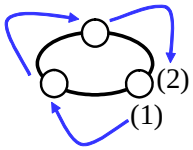
Then, along the selected direction, the source node tries to find a candidate of the next node from the top-level pointer in its own routing table (see (B)), and if such a node cannot be found, the level of pointer is decreased (see (C)). By using higher level pointer preferentially, as is the case with the skip list, the message reaches the destination node quickly.

When a candidate of the next node is found, the current node checks whether the candidate node has already received the message (see (D) and (E)). If the node has received the message previously, the source node tries to find a new candidate of the next node from lower level pointer again. Otherwise, the source node sends the message to the candidate node as the next node (see (F)). This process continues until the message is received by a node whose TypeID is the same as the destination TypeID.

When the message arrives at a node with destination TypeID, the message routing based on TypeID terminates (see (H)). If there is no node with destination TypeID, this message routing fails and negative acknowledgment is sent back to the source node (see (G)). Note that in the actual system, a message will reach a node with the closest TypeID.



(a) List structure.



(b) Ring structure.

Fig. 10: Node structure for message routing based on HashID.

However, this is out of scope in this paper because it depends on the implementation.

Just after the termination of the message routing based on TypeID, message routing based on HashID starts as shown in Fig. 9 (dotted lines in Fig. 7). This message routing is performed only among nodes with the destination TypeID. Note that the message tends to be routed in a list structure at most of rings (see Fig. 10(a)) but the message may be routed in a ring structure at higher-level ring (see Fig. 10(b)).

In the message routing based on HashID, at first, the node checks the number of digits which are shared between its own HashID and the destination HashID (see (C)). When the number of shared digits is h, the message routing starts at level h. The initial routing direction is set to *true* in Fig. 8.

The node checks whether TypeID of the neighbor node is the same as its own TypeID. If the neighbor node has the same TypeID, the node forwards the message to the neighbor node (see (E) of Fig. 9 and (1) of Fig. 10). At this time, when HashID of the neighbor node is closer to the destination HashID than its own HashID, the information about the neighbor node is stored in the message as the best node (see (D)). When the neighbor node has a different TypeID, the current node reverses the routing direction and continues the message routing (see (F) of Fig. 9 and (2) of Fig. 10(a)).

The message routing based on HashID terminates when the node with the destination HashID is found (see (A)), when the message routing for both directions finishes (see (G) of Fig. 9 and (3) of Fig. 10(a)), or when a node receives the message again in the initial direction (see (B) of Fig. 9 and (2) of Fig. 10(b)).

In order to decrease the number of hops for message routing over physical networks, we have to consider the network proximity in our message

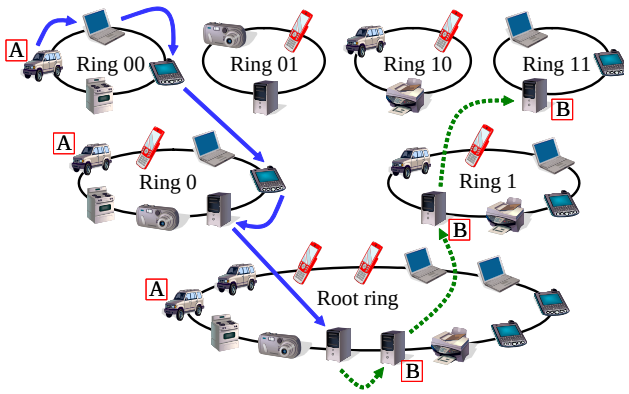


Fig. 11: Example of object management where car node A downloads an object from server node B.

routing algorithm. However, this is out of scope because some solutions have been proposed in [15] and those are available in our method.

3.4 Example of Use

Fig. 11 shows a P2P network where several types of nodes have participated by using a P2P software with our proposed method. In this network, we assume that music files have been stored in high-performance file servers. That is, TypeID of music file is the same as that of high-performance file server.

For example, when a driver (car node A) tries to listen to a favorite song, the driver inputs its title and file type into the software. In this software, TypeID of the music file is determined from its file type, and HashID is determined with a hash function from its title.

Then, message routing starts in the P2P network to find a destination node with the music file based on the determined TypeID and HashID. If the message reaches the destination node (server node B), the driver can download the object from this node.

If a DHT-based P2P network is built for each node type, i.e., for each TypeID in our method, a similar object management may be implemented. However, this implementation requires that each node has at least a pointer to each P2P network. Therefore, the number of pointers becomes large when the number of node types becomes large. On the other hand, in our proposed method, each node has only information about neighbor node's TypeIDs. Therefore, in the future, it is expected that the proposed method is effective in large-scale heterogeneous P2P networks.

4 Numerical Examples

In this section, we evaluate the performance of the proposed method by simulation. We assume that the number of P2P nodes is N and the number of objects is M. We consider how M objects are stored in N nodes by using the proposed method.

In this P2P network, a four-digit TypeID is assigned to each node and each object. For the simplicity, in the following, we denote four-digit TypeID with decimal number format, for example, TypeID 0101 is denoted as TypeID 5. We assume that TypeID i $(0 \le i \le 15)$ is assigned to a node (an object) with probability γ_i (σ_i). On the other hand, HashID is denoted as 128 bits binary string, and it is assigned to a node (an object) with a hash function.

Under this situation, we evaluate by simulation the performance of the proposed capability-aware object management. In order to perform the performance comparison, we also evaluate the performance of a conventional DHT-based method where node's capabilities are not considered.

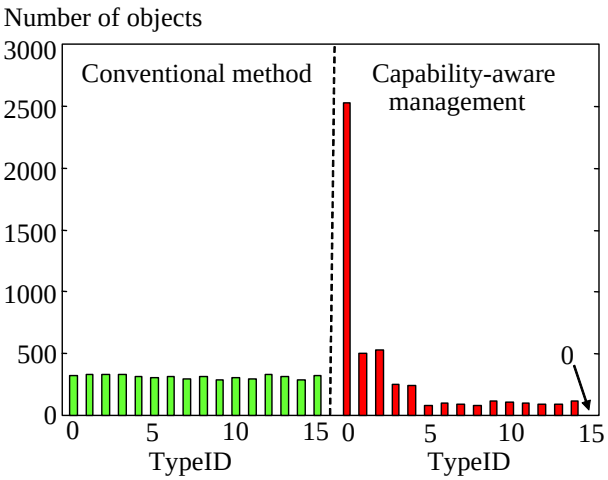
4.1 Impact of TypeID Assignment for Object

First, we investigate the impact of TypeID assignment for each object. Here, the number of nodes is $N = 16384$, and TypeID i $(0 \le i \le 15)$ is assigned to a node with probability $\gamma_i = 1/16$. In addition, we assume that the number of objects is $M = 5000$.

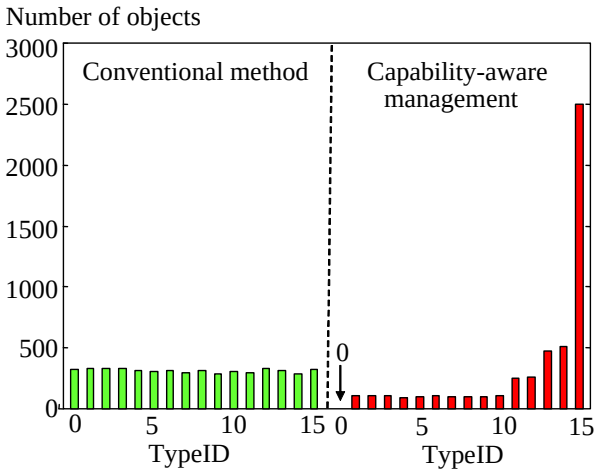
Here, we consider two cases for probability σ_i with which TypeID i $(0 \le i \le 15)$ is assigned to each object. In case 1, $\sigma_0 = 0.5$, $\sigma_1 = \sigma_2 = 0.1$, $\sigma_3 = \sigma_4 = 0.05$, $\sigma_5 = \cdots = \sigma_{14} = 0.02$, and $\sigma_{15} = 0.0$. That is, about half of objects should be stored in nodes with TypeID 0 but no object should be stored in nodes with TypeID 15. On the other hand, in case 2, $\sigma_0 = 0.0$, $\sigma_1 = \cdots = \sigma_{10} = 0.02$, $\sigma_{11} = \sigma_{12} = 0.05$, $\sigma_{13} = \sigma_{14} = 0.1$, and $\sigma_{15} = 0.5$. In this case, about half of objects should be stored in nodes with TypeID 15 but no object should be stored in nodes with TypeID 0.

Fig. 12(a) shows the total number of objects which are stored in nodes with TypeID i in the case 1. From Fig. 12(a), we find that by using the conventional method, objects are stored in all nodes randomly regardless of those capabilities. As a result, in the conventional method, capabilities of each node are never considered, as expected.

On the other hand, we find that each object can be stored in N nodes according to σ_i by using the proposed method. For example, nodes with TypeID 0 stores about half of objects and nodes with

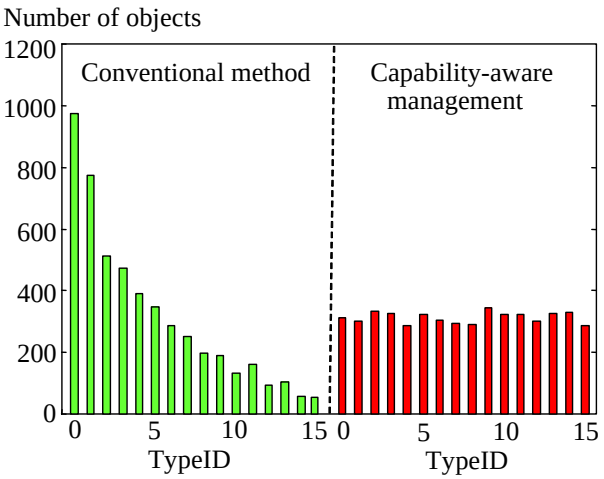


(a) Total number of objects stored in nodes with each TypeID in case 1.

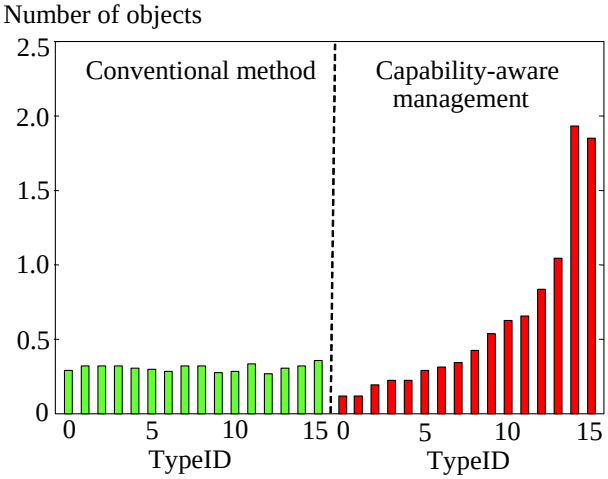


(b) Total number of objects stored in nodes with each TypeID in case 2.

Fig. 12: Impact of TypeID assignment for each object.



(a) Total number of objects stored in nodes with each TypeID.



(b) Average number of objects stored in nodes with each TypeID.

Fig. 13: Impact of TypeID assignment for each node.

TypeID 15 stores no object. Therefore, the proposed method can manage objects by considering node's capabilities. Note that in all the results for the proposed method, an object with TypeID i has been stored in a node with TypeID i necessarily.

Fig. 12(b) also shows the total number of objects in the case 2. From this figure, we find that objects can be stored in nodes with pre-specified capabilities by using the proposed method. However, in the conventional method, objects are stored in all nodes randomly regardless of σ_i. Therefore, the proposed method is effective in heterogeneous environments.

4.2 Impact of TypeID Assignment for Node

Next, we investigate the impact of TypeID assignment for each node. In this subsection, the number of nodes is $N = 16384$ and the number of objects is $M = 5000$.

Fig. 13(a) shows the total number of objects that are stored in nodes with TypeID i, and Fig. 13(b) shows the average number of objects. Here, $\gamma_0 = 0.2$, $\gamma_1 = 0.15$, $\gamma_2 = 0.1$, $\gamma_3 = 0.09$, $\gamma_4 = 0.08$, $\gamma_5 = 0.07$, $\gamma_6 = 0.06$, $\gamma_7 = 0.05$, $\gamma_8 = \gamma_9 = 0.04$, $\gamma_{10} = \gamma_{11} = 0.03$, $\gamma_{12} = \gamma_{13} = 0.02$, and $\gamma_{14} = \gamma_{15} = 0.01$. That is, the number of nodes with TypeID i decreases as i becomes large. On the other hand, TypeID i is assigned to each object with probability $\sigma_i = 1/16$. Hence, more or less the same number of objects should be stored evenly for each TypeID.

From Fig. 13(a), we find that by using the conventional method, the total number of objects for each TypeID becomes small as TypeID i increases.

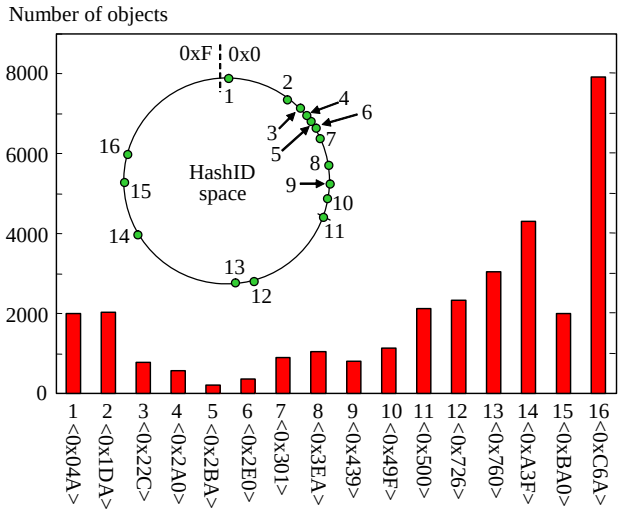


Fig. 14: Total number of objects stored in each node with TypeID 0.

This is because the number of nodes becomes small as i becomes large. As shown in Fig. 13(b), the average number of objects which are stored in each node is almost the same regardless of γ_i.

However, by using the proposed method, each object can be stored at nodes evenly according to σ_i regardless of the number of nodes with each TypeID (see Fig. 13(a)). From Fig. 13(b), a node with larger (smaller) TypeID stores a large (small) number of objects. Therefore, in the proposed method, a large number of objects can be stored in a small number of high-performance computers and a small number of objects can be stored in a large number of low-performance computers.

4.3 Effect for Load Balancing

In this subsection, we investigate how the proposed method can provide the load balancing among nodes with the same TypeID. We assume that the number of nodes is $N = 256$ and the number of objects is $M = 500{,}000$. We assume that the number of nodes with TypeID i is 16 for every i, and σ_i is equal to 1/16 for every i.

Fig. 14 shows the number of objects which are stored in each node with TypeID 0. In the horizontal axis of this figure, 16 nodes are represented as both integer number and the first three-digits of HashID. It is also shown how 16 nodes are spread in HashID space.

From this figure, we find that the numbers of objects for 16 nodes are much different. Therefore, by using the proposed method, objects cannot be stored randomly in nodes with the same TypeID. However, from the node distribution in HashID space, we find that a large (small) number of objects

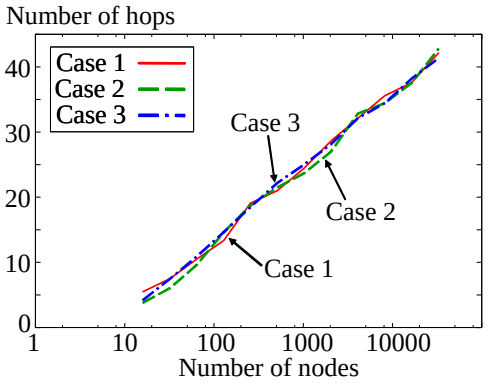


Fig. 15: Average number of hops vs. number of nodes.

are stored in sparsely-distributed (densely-distributed) nodes. This is because objects are stored in nodes according to HashID. Therefore, the proposed method can provide the load balancing if nodes are uniformly-distributed in HashID space, for example, when the number of nodes is large.

4.4 Impact of Number of P2P Nodes

In this subsection, we investigate the impact of the number of nodes on the performance of our proposed method. In the following, the number of nodes is N and the number of objects is given by $M = \lfloor N/2 \rfloor$.

In terms of TypeID assignment for each node, we set $\gamma_i = 1/16$ for every i. On the other hand, we consider three cases in terms of TypeID assignment for each object. In case 1, $\sigma_i = 1/16$ for every i. On the other hand, in case 2, $\sigma_i = 1/8$ when i is from 0 to 8 and $\sigma_i = 0$ when i is from 9 to 15. Moreover, in case 3, $\sigma_0 = 0.5$, $\sigma_1 = \sigma_2 = 0.1$, $\sigma_3 = \sigma_4 = 0.05$, $\sigma_5 = \cdots = \sigma_{14} = 0.02$, and $\sigma_{15} = 0.0$.

Fig. 15 shows the average number of hops for the proposed method in the three cases. From this figure, we find that the average number of hops for every case is almost the same. This denotes that the average number of hops for the proposed method is not affected by the TypeID assignment for each object. Therefore, the proposed method is effective even if various types of nodes have participated in a P2P network.

In addition, we find that the average number of hops increases as the number of nodes becomes large as expected. Nevertheless, the average number of hops is $O(\log N)$ for all the cases. This result shows that our proposed method can provide scalability in terms of the number of nodes.

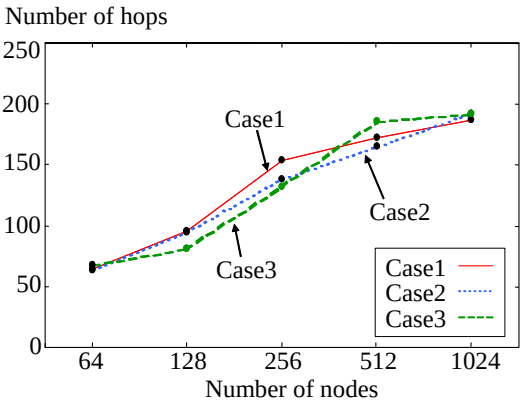


Fig. 16: Average number of hops vs. number of P2P nodes over physical network.

4.5 Performance over Physical Networks

Finally, we evaluate the performance of our proposed method over physical networks. We generate a virtual physical network topology based on Barabási-Albert (BA) model by using Boston university Representative Internet Topology gEnerator (BRITE) [16]. The generated network consists of 5000 nodes in a square.

In this network, the number of P2P nodes is N, and the nodes are randomly selected among 5000 nodes. By using the proposed method, message routing is performed among N P2P nodes based on TypeID and HashID. In parallel, over the physical network, messages are forwarded from a P2P node to another P2P node via non-P2P nodes according to Dijkstra's algorithm.

Fig. 16 shows the average number of hops over the physical network against the number of P2P nodes. In this figure, the number of objects is given by $M = \lfloor N/2 \rfloor$ for the number of nodes N. TypeID i $(0 \le i \le 15)$ is assigned to each node with probability $\gamma_i = 1/16$. On the other hand, in terms of probability σ_i, we consider the same three cases as the subsection 4.4.

From this figure, we find that the average number of hops is $O(\log N)$ for all the cases, as is the case with the previous subsection. Therefore, the capability-aware object management has scalability in terms of the number of nodes over physical networks.

5 Conclusions

In this paper, we proposed the capability-aware object management based on skip list, which is used in large-scale heterogeneous P2P networks. In this method, two identifications are utilized to consider node's capabilities and provide the load balancing. When objects are stored in and searched from a node, message routing is performed according to the two identifications. We evaluated the performance of the proposed method by simulation. From simulation results, we found that each object can be stored in nodes by considering node's capabilities in heterogeneous environments. In addition, we also found that when the number of nodes is large, the load balancing can be provided easily. The average number of hops for the proposed method is $O(\log N)$, and hence the proposed method has scalability in terms of the number of nodes. From these results, our proposed method is one of the most promising object management methods in large-scale heterogeneous P2P networks.

References:

[1] http://www.skype.com/intl/en/welcomeback/.

[2] P. Druschel and A. Rowstron, PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility, *Proc. HotOS VIII, Schoss Elmau, Germany*, 2001.

[3] http://www.peercast.org/.

[4] S. Yamamoto, Y. Murata, K. Yasumoto, and M. Ito, A Distributed Event Delivery Method with Load Balancing for MMORPG, *Proc. 4th ACM SIGCOMM Workshop on Network and System Support for Games*, 2005.

[5] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *Proc. ACM SIGCOMM*, August 2001.

[6] A. Rowstron and P. Druschel, Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems, *Proc. International Conference on Distributed Systems Platforms*, November 2001.

[7] Z. Xu, M. Mahalingam, and M. Karlsson, Turning Heterogeneity into an Advantage in Overlay Routing, *Proc. IEEE INFOCOM'03*, April 2003.

[8] J. Hu, M. Li, H. Yu, and W. Zheng, Tourist: A Self-Adaptive Structured Overlay in Heterogeneous P2P Networks, *Technical Report THTR-CS-HPC-2006-001*, Tsinghua Universitfy, 2006.

[9] C. Chou, D. Wei, C. Kuo, and K. Naik, Anonymous Peer-to-Peer Communication Protocol over Mobile Ad-Hoc Networks, *Proc. IEEE Globecom 2006*, November/December 2006.

[10] Y. Zhang, W. Liu, and W. Lou, Anonymous Communications in Mobile Ad Hoc Networks, *Proc. IEEE INFOCOM'05*, March 2005.

[11] M. Demirbas and H. Ferhatosmanoglu, Peer-to-Peer Spatial Queries in Sensor Networks, *Proc. the 3rd International Conference on Peer-to-Peer Computing*, September 2003.

[12] W. Pugh, Skip Lists: A Probabilistic Alternative to Balanced Trees, *Proc. Workshop on Algorithms and Data Structures*, January 2003.

[13] T. Tomimoto, T. Tachibana, and K. Sugimoto, Capability-Aware ID Assignment and Message Routing based on Skip List in Large-Scale Heterogeneous P2P Networks, *Proc. IEEE Globecom 2007*, November 2007.

[14] J. Aspnes and G. Shah, Skip Graphs, *Proc. the 14th ACM-SIAM Symp. on Discrete Algorithm (SODA)*}, January 2003.

[15] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman, SkipNet: A Scalable Overlay Network with Practical Locality Properties, *Proc. the Fourth USENIX Symposium on Internet Technologies and Systems (USITS'03)*, March 2003.

[16] http://www.cs.bu.edu/brite/.