IEICE TRANSACTIONS

on Communications

VOL. E102-B NO. 3
MARCH 2019

A PUBLICATION OF THE COMMUNICATIONS SOCIETY

The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

A Dynamic-Clustering Backup Scheme for High-Availability Distributed File Sharing Systems*

Hoai Son NGUYEN[†a)], **Dinh Nghia NGUYEN**[†,††], ***Nonmembers***, ***and*** **Shinji SUGAWARA**[†††], ***Senior Member***

SUMMARY DHT routing algorithms can provide efficient mechanisms for resource placement and lookup for distributed file sharing systems. However, we must still deal with irregular and frequent join/leave of nodes and the problem of load unbalancing between nodes in DHT-based file sharing systems. This paper presents an efficient file backup scheme based on dynamic DHT key space clustering in order to guarantee data availability and support load balancing. The main idea of our method is to dynamically divide the DHT network into a number of clusters, each of which locally stores and maintains data chunks of data files to guarantee the data availability of user data files even when node churn occurs. Further, high-capacity nodes in clusters are selected as backup nodes to achieve adequate load balancing. Simulation results demonstrate the superior effectiveness of the proposed scheme over other file replication schemes.

key words: *peer-to-peer (P2P), distributed hash table (DHT), file backup, load balancing*

1. Introduction

With the rapid growth of the Internet, nowadays more and more computing resources such as storage capacity and network bandwidth can be shared between Internet users with the help of peer-to-peer (P2P) network applications. Among them, P2P file sharing applications such as BitTorrent, Morpheus, eDonkey and Gnutella are popular since they allow users to easily store and share their data files with each other. In order to provide efficient mechanisms for data placement and lookup, distributed hash table (DHT) algorithms [1]–[3] are broadly used. In a DHT-based file sharing system, when a node needs to back up a data file, it creates a unique DHT key from the content of the file and then sends a file replica to a node responsible for the DHT key. When a node wants to query for a file, it sends the query message to the node responsible for the DHT key of the file based on the DHT routing algorithm. The responsible node then returns the file replica. However, irregular and frequent join/leave of nodes in a P2P network poses a problem with file availability since data files will be lost due to irregular and frequent churn (i.e. join/leave) of nodes. Further, the heterogeneity of nodes in terms of storage capacities may cause load imbalance among nodes. In this paper, we tackle the problem of how to maintain data availability with minimal data storage and maintenance overheads in DHT-based file sharing systems.

In order to guarantee data availability for DHT-based P2P networks, a number of replication methods have been proposed. The conventional approach is to allow a node to replicate its responsible data to a number of nearest neighbour nodes [2], [6], [11]. This approach is simple, however, data migration cost is high and node overloading may prevent proper data replication. RelaxDHTs [8] relaxes the requirement of replicating at nearest neighbors but it still suffers the problem of storage unbalancing between nodes and high cost of data migration. Another approach is to place replicas of data files at a number of distinct nodes and redirect a file query to these nodes [1], [5], [12]. This approach can achieve query load balancing between replica nodes. However, the maintenance of data replicas to ensure data availability requires a large amount of overhead and the problem of storage load unbalancing still exists.

This paper presents an efficient file backup scheme based on DHT key space clustering in order to guarantee data availability and support load balancing for DHT-based file backup systems. The main idea of our method is to divide a DHT-based network into a number of clusters, each of which locally stores and maintains the availability of data files. A data file will be encoded into a number of data chunks and these data chunks will be stored in nodes in the same cluster. When a node leaves the network, lost chunks of data files are replicated to other nodes to guarantee the number of available data chunk. We propose a simple cluster management method to allow the selection of high-capacity nodes for data storing in order to avoid load unbalancing between nodes in a cluster. We utilize a simple locality-preserving method to allow nodes, which are physically close to each other, to join the same cluster. As a result, we can reduce the update time of cluster information and the costs for data maintenance. Simulation results demonstrate the effectiveness of our proposed scheme in comparison with the conventional backup scheme. It can achieve high data lookup success rate with a small overhead of data storage and chunk maintenance. In addition, it yields significant reductions in node overloading.

The rest of this paper is structured as follows. Section 2 presents our cluster-based file backup scheme for DHT-based file backup systems. Section 3 shows the per-

Manuscript received April 10, 2018.
Manuscript revised July 26, 2018.
Manuscript publicized September 10, 2018.

[†]The authors are with the Faculty of Information Technology, VNU-University of Engineering and Technology, Vietnam.
[††]The author is with the Foreign Language and Informatic Center, People Security Academy, Vietnam.
[†††]The author is with the Faculty of Engineering, Chiba Institute of Technology, Narashino-shi, 275-0016 Japan.

*This paper is an extended version of a conference paper that appeared as [20].

a) E-mail: sonnh@vnu.edu.vn

DOI: 10.1587/transcom.2018EBP3108

formance evaluation of our proposed scheme. Section 4 presents a concise review of representative file backup approaches for structured P2P systems and Sect. 5 concludes this paper with remarks on possible future work.

2. Cluster-Based File Backup Scheme

2.1 Overview

We consider a DHT-based file backup system, which utilizes a ring-based DHT algorithm (e.g. Chord [2], Pastry [3], etc.) as a substrate of the P2P network. The algorithm creates one-dimensional circular key space across multiple nodes in a P2P network. Each node in the network is identified by a key, called node identifier, and is responsible for a portion of the DHT key space between its counterclockwise neighbor's identifier and its own identifier. In order to route messages to nodes responsible for DHT keys, each node maintains a set of links to its successor node (i.e., its clockwise neighbor) and a number of other nodes defined by the DHT routing algorithm. A message containing a key as its destination is routed to the node responsible for the key based on the DHT routing algorithm. With the Chord protocol [2], it can route a message to a node responsible for a key in $O(\log N)$ hops while each node only needs to maintain $O(\log N)$ links to other nodes, where N is the number of nodes in the network. The use of a DHT routing algorithm can guarantee the scalable and fault-tolerant characteristics of our file backup system.

Our backup scheme dynamically divides a DHT network into a number of clusters, each of which contains a number of nodes responsible for a continuous portion of DHT key space. When a node has a data file that needs to be backed up, it encodes the data file into n data chunks using erasure code [9] and these data chunks are stored in a number of nodes in the same cluster, which have high available storage capacities. As the characteristic of erasure code, if a number, say k ($k < n$), of any chunks of a data file are collected, the file can be reconstructed. Here, k and n are predefined system parameters. In order to maintain the availability of data files when a node churn occurs in the network, the number of data chunks of a file is kept to be larger than or equal to k, the number of chunks necessary to reconstruct the file.

The next subsections describe our proposed backup scheme in detail.

2.2 Cluster Information Management

In our scheme, DHT key space is divided into partitions, each of which corresponds to a cluster. For a d-bit DHT key space, the border of k^{th} cluster is defined by the first key and the last key of the cluster in the clockwise direction. Nodes that are responsible for DHT keys falling between the first and the last key of a cluster belong to that cluster. We define a node, which is responsible for the first key of a cluster, as the first node of the cluster and a node, which is responsible

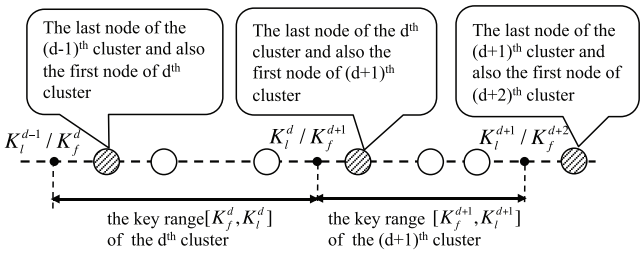

Fig. 1 The key range of clusters.

for the last key of a cluster, as the last node of a cluster. The last node of a cluster is also the first node of the next cluster as shown in Fig. 1. A node determines whether it is the first node or the last node of a cluster by checking if it is responsible for the first or the last key of the cluster.

Nodes in the same cluster exchange messages between each other to update the information of the cluster. An update message contains the following information.

- The key range of the cluster
- The address of the first node of the cluster
- A list of high-capacity nodes: the list of nodes in the cluster, which have high available storage capacities. The available storage capacity of a node is the amount of data that the node is still able to store. It is calculated by the difference between the storage capacity of a node and the amount of data chunks already stored in the node.
- A list of nodes left the network in an update round. The first node of the cluster determines the list of nodes leaving the network by the notification messages sent by the leaving nodes or their successor nodes as described in Sect. 2.4.
- Hop count: The number of nodes that the update message travels over beginning from the first node. The hop count will be reset to 0 by the first node when it sends a new update message.

The first node of a cluster initiates the update process in a round by sending an update message to its successor node. The successor node then updates the received message with its information and sends the message to the next node and so on. When the last node of the cluster receives the message, it sends the message back to the first node and the first node will send the message in the next update round (Fig. 2). The update time of cluster information is the total delay time of sending messages between nodes in the cluster, and is roughly proportional to the number of nodes in the cluster.

If the first node of a cluster is failed in the middle of an update process, its successor node will detect the failure due to the stabilization procedure of the DHT algorithm [2], [3]. Since the successor node is within the same cluster with the failed first node, it keeps information of the cluster including the key range of the cluster. The successor node now is responsible for the first key of a cluster. Thus, it itself becomes the new first node of the cluster and sends an update

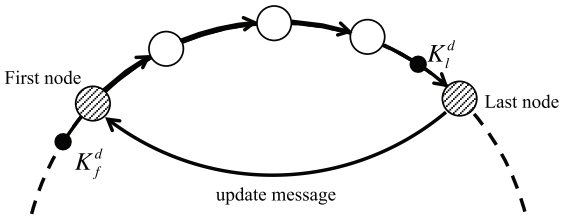

Fig. 2 Transmission of update messages in the d^{th} cluster whose key range is $[K_f^d, K_l^d]$.

message immediately. The last node of the cluster cannot send back the update message using the address of the failed first node. In this case, the last node sends back the update message by the use of the conventional DHT routing procedure. The destination address of the message now becomes the first key of the cluster. Similarly, in the case a last node of a cluster is failed, its successor node will become the last node of the cluster when it receives an update message sent from the predecessor node of the failure node, thanks to the stabilization procedure of the DHT algorithm.

When a node receives an update message, it stores the information of the cluster into its database. It then checks the capacity of nodes in the list of high-capacity nodes. If its capacity is bigger than the capacity of any node in the list, it will insert its information to the list including its address and capacity. If the list is full, it removes the information of the lowest capacity node. When the first node in the cluster receives an update message from the last node, it can obtain the list of high-capacity nodes in the cluster and send this list within an update message in the next round.

The list of high-capacity nodes in a cluster is used for each node in the cluster to select nodes to store its backup data chunks. By using this list, we can avoid high network overhead due to probe messages used to find available nodes for data backup. When a node joins the network, it can get cluster information from its successor node including the list of high-capacity nodes in its cluster.

Update messages will be sent periodically. However, when the number of nodes leaving the network due to failures in a period is higher than a threshold value, the first node of the cluster will send an update message immediately without waiting for the end of the update period.

2.3 Data Backup and Queries

In our backup scheme, data chunks of a file are stored in a number of nodes, which belong to the same cluster as the node responsible for the DHT key of the file. The DHT key of a file is a unique key created from the content of the file and is used to query the backup file. The node responsible for the DHT key of a file (i.e. responsible node) will manage the data backup information of the file. The information includes the list of backup nodes storing the data chunks of the file. When node churn occurs in the network, the responsible node maintains the availability of the file by monitoring the number of available data chunks as mentioned in Sect. 2.4 and performing a further backup if necessary.

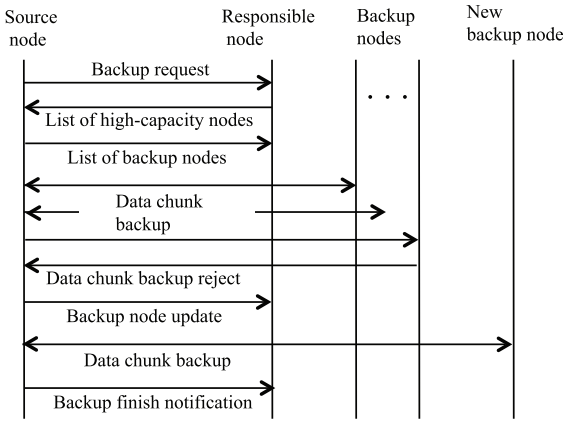

Fig. 3 Data file backup procedure.

A data backup process for a new data file is done as follows (Fig. 3).

- Step 1. When a node has a new data file to backup (i.e. the source node), it first creates the DHT key of the file and sends a backup request message to the node responsible for the DHT key based on DHT-routing algorithm.
- Step 2. The responsible node sends the list of high-capacity nodes in the cluster to the source node. After receiving the list of high-capacity nodes, the source node randomly selects a number of backup nodes from the list and notifies the responsible node about the list of backup nodes.
- Step 3. The source node creates data chunks from the file using erasure code and sends each data chunk and the DHT key of the file to a backup node for data storage. If a backup node cannot accept a data chunk, it informs the source node and the source node will select another backup node to store the data chunk and update the responsible node on the information of the new backup node.
- Step 4. When the backup process finishes, the source node notifies the responsible node of the finish of the process. The responsible node stores into its database the information of the backup file including the DHT key of the file and the list of backup nodes.

If a source node is failed before the backup process of a file finishes, the node responsible for the DHT key of the file cannot receive the confirmation message from the source node. In this case, it will confirm the number of backup data chunks with nodes in the list of backup nodes of the file. If this number is smaller than k (i.e. the minimum number of data chunks needed to reconstruct the file), the responsible node will discard the information of the file and inform nodes in the list of backup nodes to discard data chunks of this file. The file then will be considered as not being backed up. If the number of backup data chunks is bigger than k, the file will be considered as being backed up and the responsible node stores into its database the information of

the backup file.

The use of erasure coding scheme for file redundancy can improve file availability while still maintaining the same level of redundancy compared with replication scheme. If a file is encoded into n data chunks and the number of data chunks needed to reconstruct the file is k, the size of a data chunk will be $|F|/k$, where $|F|$ is the size of the file, and the effective redundancy factor will be n/k [9]. The file can be reconstructed even though any $n - k$ data chunks are lost. However, erasure coding scheme has the disadvantage that it requires more computation cost to create data chunks from a file and reconstruct a file from its data chunks.

Since the number of nodes in a cluster is large, with high probability the source node will find a backup node that can store data chunks. The random selection of backup nodes from the list of high-capacity nodes helps to avoid high backup load for the highest-capacity nodes in the list.

When a node wants to query a backup file, it sends a query message to the node responsible for the DHT key of the file. The responsible node looks up its database for the DHT key and sends back the list of backup nodes, which store data chunks of the file. The query node then randomly chooses backup nodes from the backup node list and sends query messages to these nodes until it acquires k data chunks necessary to reconstruct the file.

Since the information of backup nodes is essential for query resolution, the responsible node also stores this information in its successor node list. If the node leaves due to failure, the successor node will be responsible for the information.

2.4 File Recovery

When a node leaves the network, it moves information of backup files that it is responsible for to its successor node and sends a notice message about its leaving status to the first node of the cluster. If a node leaves the network without notification due to network failure or node failure, the successor node detects node failure based on the stabilization procedure of the DHT algorithm. The successor node then sends a notice message about the leaving node to the first node of the cluster. However, in both cases, data chunks of backup files stored in the leaving node may be lost. As mentioned above, we need to maintain at least k data chunks of any file in the network to guarantee file availability.

To deal with that problem, the first node of the cluster sends an update message including the list of leaving nodes in a cluster periodically or immediately depending on the number of leaving nodes in an update period. When a node receives an update message, it checks the list of backup nodes for data files that it is responsible for. If a leaving node is in the list, the number of data chunks of each file backed up at the leaving node is recounted. If the number of data chunks of a data file is smaller than a threshold value say m but larger than or equal to k (i.e. the number of data chunks necessary to reconstruct the file), the responsible node performs data backup to recover lost chunks. Here, m must be bigger than or equal to k.

Algorithm 1 Data chunk backup

```
chunkInfo = lookupChunkInfo(dhtKey)
if chunkInfo.size < m
  if chunkInfo.size >= k
    i = random(1,chunkInfo.size)
    backupNode = chunkInfo.getBackupNode(i)
    sendChunkBackupRequest(backupNode, dhtKey, chunkInfo)
  else
    for i=1: chunkInfo.size
      backupNode = chunkInfo.getBackupNode(i)
      sendChunkDeleteRequest(backupNode, dhtKey,
                                    chunkInfo.getChunkID(i))
    end for
  end if
end if
```

Fig. 4 Pseudocode of data backup for a data file at responsible nodes.

In order to recover a lost data chunk of a file, the node responsible for the DHT key of the file will randomly select a node in the list of high-capacity nodes as a backup node and send a chunk backup request message to that node (Fig. 4). The message contains the information of existing data chunks such as the list of backup nodes that are keeping these data chunks. The newly selected backup node will retrieve data chunks from other backup nodes, reconstruct the file and create the lost backup chunk by itself. If there are multiple data chunks necessary to restore, the node will send other restored data chunks to other newly selected backup nodes, which are randomly selected from the list of high-capacity nodes, and updates the list of backup nodes to the node responsible for the DHT key of the file.

If the number of data chunks of a data file is smaller than k, the data file cannot be reconstructed. In this case, the correspondent node sends a chunk delete request message to nodes that keep the data chunks of the file and request these nodes to delete the data chunks.

The procedure of file backup can ensure the availability of files, however, the reconstruction of a file by collecting data chunks from backup nodes will take high cost of data transfer. In order to reduce the backup cost, querying nodes that look up files can create data chunks which are lost due to node leaving and send the data chunks to new backup nodes. This procedure can be performed as follows (Fig. 5).

When a node wants to query a file, it sends a query message to the node responsible for the DHT key of the file. The responsible node will check the number of available data chunks. If there are lost data chunks, it will request the querying node to send lost chunks to newly selected backup nodes. The request is sent, along with the high-capacity node list, in the response message for the querying node. After reconstructing the queried file from existing data chunks, the querying node re-creates the lost data chunks as requested. It then sends the data chunks to backup nodes which are selected among nodes in the high-capacity node list sent from the responsible node. In this case, a lost chunk is recovered only at the cost of sending the chunk from the querying node to the newly backup node.

Due to the propagation delay of information update

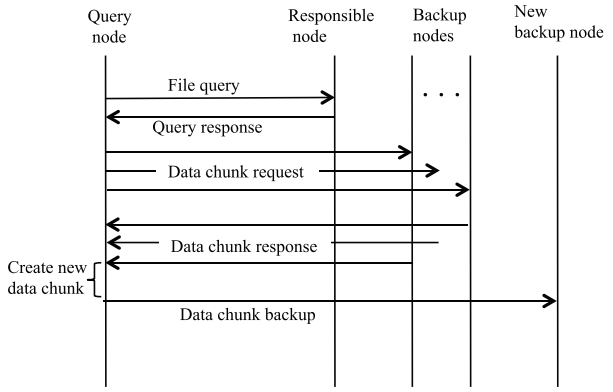

Fig. 5 File query and backup procedure.

messages and the high churn rate of nodes, the number of lost data chunks of a file may be high. If the number of existing data chunks of a file is lower than the number of data chunks necessary to reconstruct a file, the file is not able to be recovered. In this case, the node responsible for the DHT key of the file will notify the node storing existing data chunks to remove these data chunks.

2.5 Dynamic Cluster Construction

In P2P networks, nodes freely join or leave the network. Thus, if the key range of clusters is fixed, the number of nodes in a cluster will change heavily depending on the number of nodes existing in the network. If the number of nodes in a cluster is small, it may not be able to find any available node to back up data due to the lack of storage capacity. If the number of nodes in a cluster is large, the time required for a cluster information update message passing through all nodes in a cluster is large and therefore the time of updating cluster information is also large. It means that the list of high-capacity nodes in the cluster may be stale and the backup may be failed because the source node cannot find any available node in the list for backup. Further, if the list of lost nodes is sent to nodes in a cluster too late, file backup may be failed due to the lack of available data chunks. Therefore, the number of nodes in a cluster should be kept to be a reasonable number.

We propose a dynamic cluster construction method, which allows a cluster to split into two smaller clusters when the number of nodes in a cluster is over a threshold value. In this method, when the first node of a cluster receives an update message from the last node, it checks the number of nodes in the cluster. If the number of nodes in the cluster is over a threshold value, it will initiate the splitting procedure by sending a notification message to the first node of the second cluster by the use of DHT routing. The destination of the message will be the first key of the second cluster. After receiving the acknowledgement message from the first node of the second cluster, it sends an update message to inform all nodes in the first new cluster about the splitting. The first node of the second new cluster also sends an update message to other nodes in the same cluster.

The key range of new clusters is decided as follows. If the key range of an original cluster is $[K_f^d, K_l^d)$, the key range of two new clusters after splitting will be half of the original cluster as Eq. (1).

$$\begin{aligned} K_f^{d_1} &= K_f^d \\ K_l^{d_1} &= K_f^{d_2} = K_f^d + 1/2(K_l^d - K_f^d) \\ K_l^{d_2} &= K_l^d \end{aligned} \tag{1}$$

where $[K_f^{d_1}, K_l^{d_1}]$ are the key range of the first new cluster and $[K_f^{d_2}, K_l^{d_2}]$ are the key range of the second new cluster.

For some reasons such as the failure of a first node of new clusters just after the splitting process starts, the splitting process of a cluster may be failed. In this case, the successor node of the failed node will become the new first node of the cluster as mentioned in Sect. 2.2. As a result of the failure of the splitting process, the key range stored at nodes in the new first cluster and the key range stored at nodes in the new second cluster may be different. In this case, the first node of the second new cluster will detect the difference since it is also the last node of the first new cluster (Fig. 1) and receives update messages of the first cluster. If nodes in the first new cluster are not updated with the new range, the first node of the second new cluster notifies the first node of the first new cluster to complete the splitting process. If nodes in the second new cluster are not updated with the new range, it sends an update message to other nodes in the same cluster.

The number of nodes in a cluster may decrease due to node churn. In order to keep the number of nodes in a cluster to be large enough, if the total number of nodes of two neighbour clusters is smaller than a threshold value, the two clusters will be merged into one cluster. In this case, when the number of nodes in a cluster is smaller than a threshold value, the first node will check the node number in the neighbour cluster to which it also belongs. If the merging condition is satisfied, the first node will send a notification message to the first node of the neighbour cluster. If the first node of the neighbour cluster agrees to merge, it will send a reply message and the two first nodes will send update message to all nodes in the new merged cluster.

Since a cluster may be split to a number of clusters, data chunks of a file may not be stored in nodes in the same cluster with the node responsible for the key of the file. In this case, when a node storing a data chunk of a file leaves the network, the node responsible for the key of the file may not know the leaving of the node. In order to solve this problem, the first node of a cluster will send the list of leaving nodes in its cluster to the first nodes of i-neighbour clusters. Here, cluster A and cluster B are i-neighbour clusters if there are maximum i clusters between them. The update of leaving nodes is easy to implement because the first node of a cluster is also the last node of the preceding neighbour cluster. The list of leaving nodes in i-neighbour clusters then is sent within the update message to all nodes in the cluster. The node storing a data chunk of a file may belong to a

cluster which is not a *i*-neighbour cluster of the cluster that the node responsible for the key of the file belongs to. In this case, the data chunk will be moved to a node within the same cluster with the node responsible for the key of the file.

2.6 Locality Preserving and Load Balancing

The maintenance of data chunks of a data file in a cluster introduces network overhead due to the movement of data chunks between nodes. Further, the propagation delay between nodes in the same cluster has an effect on the delay of update information, and therefore effects the success ratio of file maintenance. In order to reduce network overhead and delay, nodes in the same cluster should be physically closely located. Another problem is that when the total capacity of files distributed over the cluster exceeds that of nodes belonging to the cluster, a number of nodes in the cluster will be overloaded.

In order to deal with above problems, we allow a newly joining node to select a position in the DHT ring from a set of randomly selected positions. When a new node joins the network, instead of having the node randomly generate only one node-identifier, we let the node generate a set of t node-identifiers, each of which corresponds to a candidate position in the DHT ring. Here, t is a system parameter. The set of t node-identifiers can be generated randomly by making use of a hash function on the combination of the node's unique information such as IP address and several random values. The new node then sends a query message to the successor node of each identifier in the identifier set by DHT routing algorithm in order to get information of the successor node and predecessor node of the identifier and information of the cluster that contains the identifier. After measuring the propagation delays between the joining node and the successor node or the predecessor node of each identifier, the following cost function C^i is then used for the joining node to select the best position among candidate positions.

$$C^{(i)} = (d^{(i)}_{self,succ} + d^{(i)}_{self,pred}) + \alpha C^{(i)}_{bestNodeList} \quad (2)$$

where $d^{(i)}_{self,succ}$ is the delay between the joining node and the successor node of the i^{th} identifier, and $d^{(i)}_{self,pred}$ is the propagation delay between the joining node and the predecessor node of the i^{th} identifier, $C^{(i)}_{bestNodeList}$ is the total capacity of nodes in the high-capacity node list in the cluster that contains the i^{th} identifier and α is a parameter.

The position which has the lowest value of the cost function due to Eq. (2) will be selected as the joining position of the new node. This cost function will guarantee that the new node will be put into the position that the delay to its successor node and its predecessor node is small. Further, if there are two positions that the delay is almost the same, it will join the cluster which has smaller capacity. Although adding delay and overhead to the joining process of a node, this joining scheme helps to keep a good degree of load balancing between clusters and reduce the propagation delay between nodes.

3. Evaluation

3.1 Evaluation Method

We evaluate our backup algorithm by building a simulation program, which simulate the following backup methods

- The successor list replication method [2]: A data file will be replicated at successor nodes of the node responsible for the key of the file. When a new node joins the network and become a new successor node of a node, file replicas will be moved from an old successor node to the new successor node. In the case a successor node leaves the network, the node responsible for the key of the file will maintain the availability of replicas of the file by copying a file replica and store at the new successor node.
- RelaxDHT method [8]: An extension of the successor list replication method. A data file will be replicated at a number of nodes in a list of neighbour nodes. When a new node joins the network, file replicas are moved to the new node only if the node storing the replica is not within the extended list of neighbour nodes. In our simulations, the size of the list of neighbour nodes and the size of the list of extended list of neighbour nodes are set to be 8 and 16 respectively.
- Fixed clustering method: Our proposed method except that the number of clusters and the key range of each cluster is fixed.
- Dynamic clustering method: Our proposed method, in which the number of clusters and the key range of each cluster is dynamically changed due to the number of nodes in the network as described in Sect. 2.5.

The simulator is extended from Jonathan Ledlie's simulator [15]. It operates in discrete time steps. Each time step consists of the following phases: node arrival and departure, information updates, data replication, file consistency maintenance and file query. We use simulation instead of deploying our method in a real network such as Planet Lab because it enables us to track down and observe all system's operations easily. In addition, by using simulation we can check the performance of the system in a wide range of evaluation scenarios.

At each step, nodes arrive and depart with the lifetime of the nodes based on Pareto birth/death distributions. We generated several Pareto birth/death distributions with average lifetime of a node set to be 15 minutes, 30 minutes, 1 hours, 2 hours and 3 hours. The simulation time is set to be 3 hours but we recorded statistics only for the second half of a simulation to avoid instabilities of simulation results. The capacity of a node is generated randomly from 5 to 235 data units. The average capacity of a node is 120 data units. Each file has the same size that is equal to 3 data units.

In order to simulate the delay between nodes, we use the GT-ITM Generator [16] to create a 18750-node transit-stub graph based on transit-stub model (Fig. 6). There are 50

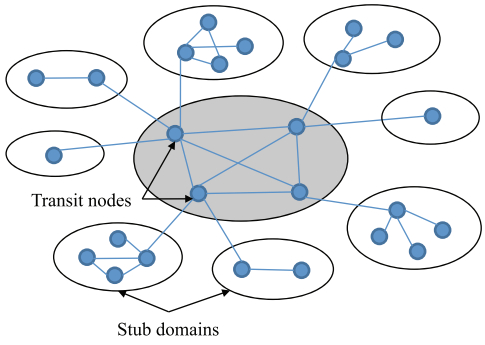

Fig. 6 Example of a transit stub model.

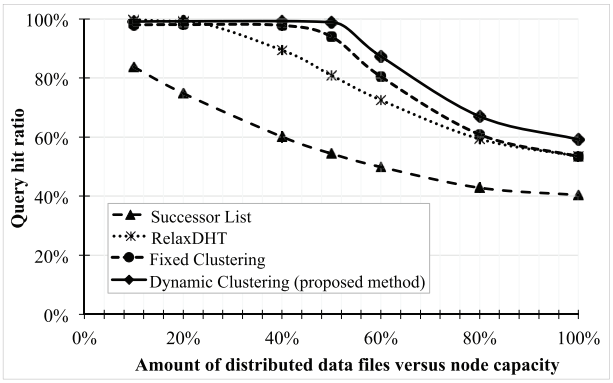

Fig. 7 Successful query rates with different amounts of distributed data files versus node capacity.

transit nodes, which have direct connection to each other. Each transit node connects to 15 stub domains on average and each stub domain contains 25 nodes on average. We use a randomized function to generate the delay between nodes in the same stub domain. The delays between transit nodes are ranged from 100ms to 200ms. The delays between transit nodes to stub domains are ranged between 20ms and 50ms. Delays from nodes to stub domains are ranged from 1ms and 10ms.

We evaluate the performance of the proposed methods by a number of parameters as following subsections.

3.2 Simulation Results

3.2.1 Evaluation of Query Hit Ratio

We perform simulations to evaluate the query hit ratio, defined as the ratio between the number of queries that successfully get a file and the total number of queries. DHT keys of files which are already distributed into the network are randomly selected as queried DHT keys.

In the simulations of four methods, the backup ratio is set to be 2. It means that the total size of backup data chunks is twice as big as the size of data file. In the simulation of our backup methods, the number of data chunks created from a file and stored in the system (i.e. the parameter n) is 6 and the smallest number of data chunks that is necessary to reconstruct a file (i.e. the parameter k]) is 3. The number of data chunks of a file that triggers the recovery procedure (i.e. the parameter m) is set to 4. The size of a data chunk is 1 data unit. In the simulation of successor list backup method and relaxDHT method, two file replicas of a file will be stored at two successor nodes of the node responsible for the key of the file. If the number of replicas of a file is smaller than or equal to one, the node will perform the maintenance procedure to maintain the file replicas in the system.

In the first simulation, we evaluate the query hit ratio with the change of system load, which is measured as the total size of backup data files in the network. We change the number of data files distributed into the network such that the total size of backup data files in the network is changed from 10% to 100% of total storage capacity of the network. The average lifetime of a node is 15 minutes. The total number of joining nodes and leaving nodes is 8000. When the network is stable, about half of them (i.e. 4000 nodes) join the network. Each node takes 10 samples of positions during joining phase. The size of high-capacity node list sent within each update message is set to 20. In the proposed dynamic clustering method, the threshold number for a cluster to split into smaller clusters is 200. Moreover, if the node number of two neighbour clusters is more than 150, they will be merged into one cluster. In the fixed clustering method, the number of clusters is set to be 10. The high-capacity node list in a cluster is updated in a time period equal to the total delay of cluster information update messages. For each round of a simulation, we perform queries on 1% of files already distributed into the network and count the number of files successfully recovered by query nodes.

The evaluation results of query hit ratio are plotted in Fig. 7. We show successful query rate on the y-axis and the percentage of data units per node capacity on the x-axis. The results show that the proposed method can achieve better query hit ratio than the conventional methods. When total data amount of backup data files is 50% of total storage capacity of the network, our dynamic cluster method can achieve a query hit ratio of 98.88% while the query hit ratio of relaxDHT method and the successor list replication method only achieve a query hit ratio of 80.88% and 54.40% respectively. When the system load is small, relaxDHT can achieve as high query hit ratio as the proposed methods. However, when the system load increases, its query hit ratio is not as high as the proposed method. It is because, in the proposed methods, data chunks of files are stored at a number of high-capacity nodes in the same cluster, which often have a large number of nodes. On the other hand, relaxDHT method and the successor list replication method only stores replicas of a file in a limited set of nodes. Therefore, the probability that data chunks of a file cannot be stored due to the lack of available nodes is high compared with the proposed methods.

When total data amount of backup data files increases over 50% of total storage capacity of the network, the query hit ratio of the proposed methods decreases quickly. It is

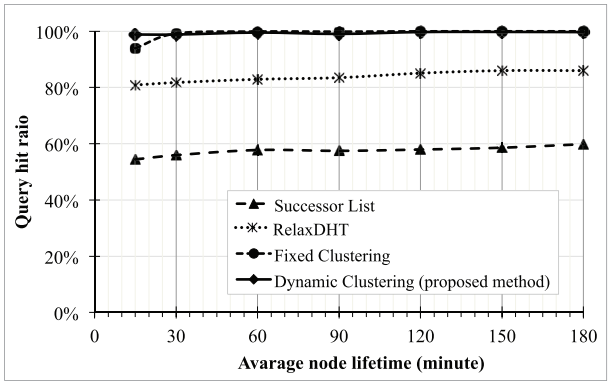

Fig. 8 Successful query rates with different average lifetimes of nodes.

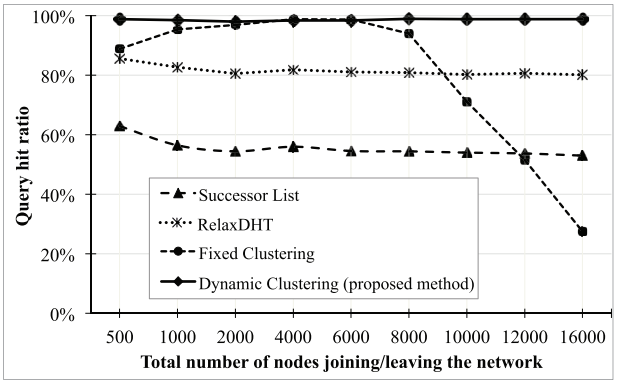

Fig. 9 Query hit ratio with different total number of nodes joining/leaving the network.

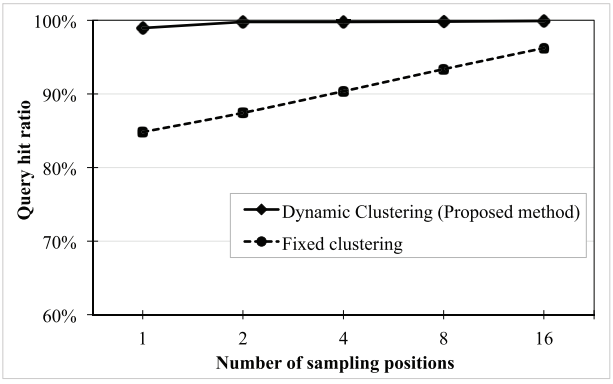

Fig. 10 Query hit ratio with the number of sampling positions.

because in this simulation, we set the backup ratio to be 2. Hence, when total data amount of backup data files increases over 50% of total storage capacity of the network, the amount of data storage required to store all data chunks increases over the total storage capacity of the network and the system can not maintain enough number (i.e. 6 data chunks per file in this simulation) of data chunks for an increasing number of files. As a result, the number of files lost due to the leaving of backup nodes rises quickly.

In the following simulation, we study the effect of node churn rate on the performance of our methods and the successor list replication method. The parameters used in the simulation are kept the same as the previous simulation except the average lifetime of a node is changed between 15 minutes and 180 minutes and the total size of backup data files in the network is set to 50% of total storage capacity of the network. The simulation result is shown in Fig. 8.

We found that the proposed dynamic clustering method can achieve high query hit ratio when nodes leave and join the network frequently (i.e. the average lifetime of a node is small) thanks to our proposed file recovery mechanism. The query hit ratio reaches to 98.88% when the average node lifetime is 15 minutes and increases to above 99.72% when the average node lifetime is longer than 120 minutes (i.e. the number of data chunks lost due to the leaving nodes in a time unit is small). The query hit ratios of the relaxDHT method and the successor list replication method are much lower and increase slightly up to 86.00% and 59.84% respectively when the average node life time increases. The query hit ratio with the fixed clustering method is a little lower than the one with the dynamic clustering method when the average node lifetime is 15 minutes but about the same when the average node lifetime increases.

We also investigate the relationship between the query hit ratio and the number of nodes joining into the network. The average lifetime of nodes is set to 15 minutes and the number of clusters in the case of the fixed clustering method is 10. As shown in Fig. 9, the proposed dynamic cluster method can achieve stable performance with high query hit ratio (above 97.99%). The fixed clustering method only achieves high query hit ratio when total number of nodes joining/leaving the network is 2000-6000 nodes but reduces query hit ratio when the number of nodes is smaller or larger than the range of 2000-6000. When the number of nodes in a network is small, some clusters only have a small number of nodes and there is not any backup node available for a number of files. Conversely, when the number of nodes in a network is large, some clusters have a large number of nodes. The delay of update messages in this case is large and therefore some files may be lost due to node churn before being maintained.

In order to show the effectiveness of our proposed node joining scheme for locality preserving and load balancing, we study the effect of the number of sampling positions on the query hit ratio (Fig. 10). When the number of sampling positions increases, the delay between nodes belonging to the same cluster decreases and the load balancing between clusters is also improved. It results in the improvement of query hit ratio from 98.93% to 99.9% in dynamic clustering method and from 84.8% to 96.22% in fixed clustering method when the number of sampling position increases from 1 to 16. In fixed clustering method, the effectiveness of position sampling technique in load balancing contributes a lot to the improvement of query hit ratio since the number of nodes may be very different between clusters.

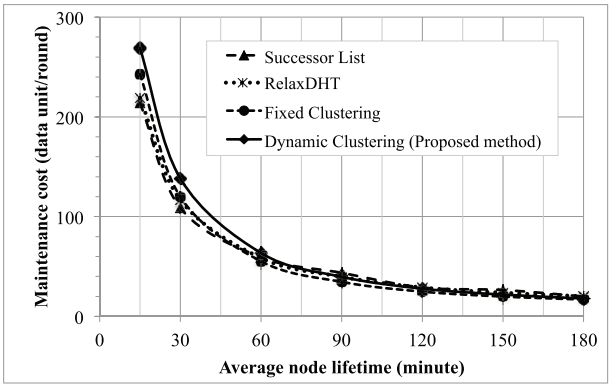

Fig. 11 File maintenance cost with different average lifetimes of nodes.

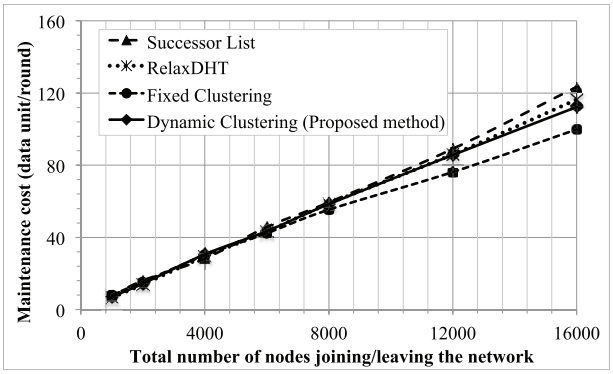

Fig. 12 File maintenance cost with different total number of nodes joining/leaving the network.

3.2.2 Evaluation of Maintenance Cost

File maintenance cost constitutes a major portion of our system overhead. We evaluate the file maintenance cost as the total amount of data chunks moved between nodes to ensure the availability of the data files in the case of node leaving. As shown in Sect. 2.4, the maintenance cost of our methods includes the cost of collecting data chunks of a file and re-distributing lost data chunks. If a lost data chunk is recreated by a querying node, only the cost of re-distributing lost data chunks is calculated. The maintenance cost of the successor list replication scheme is the cost to replicate data files to successor nodes due to node churns.

Since the cost is directly related to node churn, we evaluate the maintenance cost with the change of the average lifetime of nodes. The parameter used in this simulation is the same as the first simulation except that the amount of data files distributed by a node is set to be 20% of node capacity.

The result is demonstrated in Fig. 11. We see that the maintenance cost of three methods decreases significantly when the average lifetime of a node increases (i.e. the number of leaving nodes decreases). When the number of leaving nodes decreases, the number of lost data chunks decrease and therefore the data moving cost for data recovery decreases. When the average lifetime of a node is 15 minutes, the proposed dynamic clustering method requires about 25.61% and 22.77% higher cost for file consistency maintenance than the successor list replication method and relaxDHT method. However, when the average lifetime of a node is longer than 90 minutes, the file maintenance cost of our method is lower than the one of the successor list replication method but the successful query rate of our method is higher than the one of the successor list replication method. It is because in the successor list replication method, when new nodes join the network, data moving is also required to guarantee that the replicas of data files are stored near the responsible nodes.

We also evaluate the maintenance cost with the change of total number of nodes joining/leaving the network. The amount of data files distributed by a node is set to be 20% of node capacity and the average lifetime of a node is set to be 60 minutes. The file maintenance cost of our methods depends on the number of nodes leaving the network. Therefore, as shown in Fig. 12, the file maintenance cost of our method increases linearly with the number of nodes in the network. The simulation results also show that the file maintenance cost of our method is lower than the one of the successor list replication method and relaxDHT method as the number of nodes in the network grows.

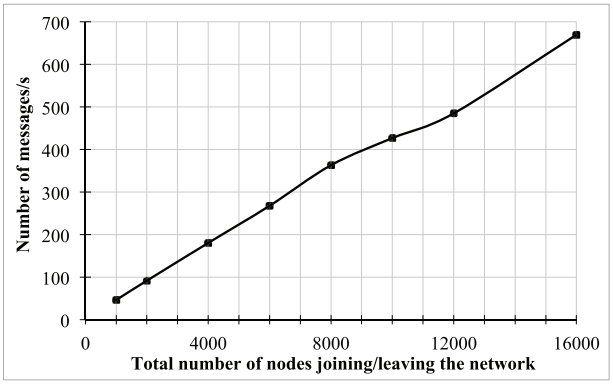

Fig. 13 Total number of update and notification messages per second with different total number of nodes joining/leaving the network.

Compared with conventional methods, the proposed method introduces cluster maintenance cost due to the transmission of update messages and notification messages for node leaving and dynamic clustering. We evaluate this maintenance cost by counting the average number of messages the network sends per second. The results are shown in Fig. 13. If total number of nodes joining/leaving the network is 8000 nodes, 363 messages are necessary to send in each second. The cost increases linearly with the number of nodes in the network since each node receives and sends an update message in a period of time and the number of leaving nodes is also proportional to the number of nodes in the network. The number of messages sent by each node in each second is almost the same. In each round about a half of total number of nodes joining/leaving the network, each node sends about 0.1 message per second on average. If the

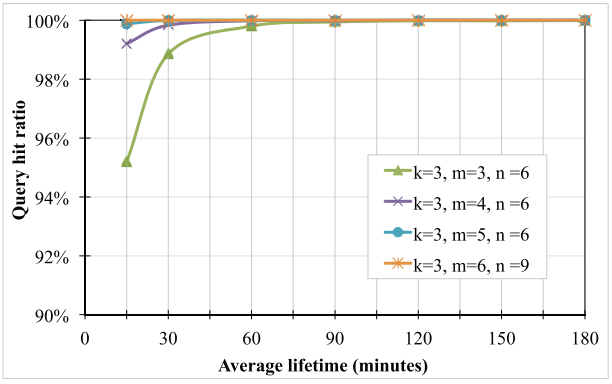

Fig. 14 Effect of file backup parameters on successful query rates with different average lifetimes of nodes.

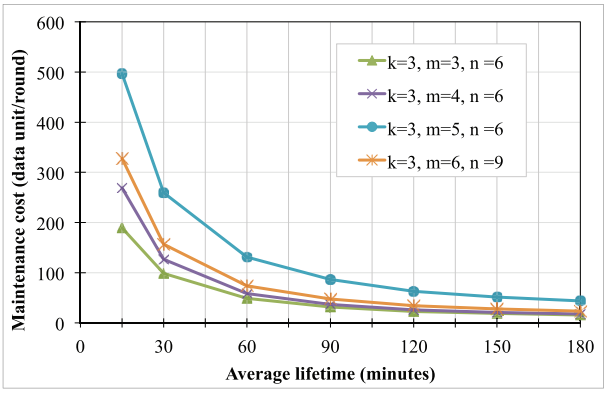

Fig. 15 Effect of file backup parameters on data moving cost with different average lifetimes of nodes.

size of each message is 500 byte, the bandwidth required to send these messages is about 400bps, which is small enough in practice.

3.2.3 Effects of File Backup Parameters

In order to evaluate the effects of file backup parameters, which are the number n of data chunks created from a file, the number k of data chunks required for file reconstruction and the number m of existing data chunks that triggers a data backup to recover lost chunks. In this simulation, we evaluate the following parameter sets ($n = 6, k = 3, m = 3$), (n = 6, k = 3, m = 4), (n = 6, k = 3, m = 5) and (n = 9, k = 3, m = 6) with k unchanged at 3. The total number of joining nodes and leaving nodes is 8000.

In the first simulation, we investigate the effect of file backup parameters on successful query rate and file maintenance cost with different average lifetimes of nodes. Total data amount of backup data files is set to be 20% of total storage capacity of the network. When the average life time is small (i.e. nude churn rate is high), the higher parameter m (i.e.) is, the higher query hit ratio is (Fig. 14). It is because a system with high value of m will trigger a data backup process early and therefore lost data chunks will be recovered early. However, the file maintenance cost will be high since early recovery of lost data chunks requires high cost of data transfer between nodes (Fig. 15). The simulation results show that the file maintenance cost of the system with ($n = 6, k = 3, m = 5$) is 84.69% higher than the one with ($n = 6, k = 3, m = 4$) when average life time of nodes is 15 minutes.

When total data amount of backup data files is 20% of total storage capacity of the network, the parameter set ($n = 9, k = 3, m = 6$) gives the highest query hit ratio for the system. However, the system with $n = 9$ will require 1.5 times more storage space for file backup than the system with $n = 6$. Therefore, when the number of backup data file increases, nodes in the system with $n = 9$ will be overloaded earlier than nodes in the system with $n = 6$ and the query hit ratio of the system with $n = 9$ will decrease faster than the system with $n = 6$ (Fig. 16). By changing these parameters for some high-priority files, we can guarantee the availability of these files in P2P networks.

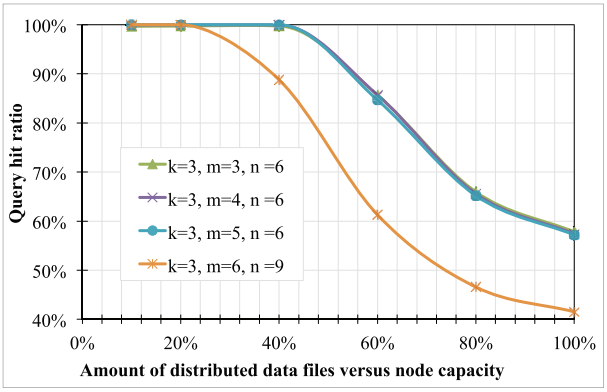

Fig. 16 Effect of file backup parameters on successful query rates with different amounts of distributed data files versus node capacity.

4. Related Works

There are three basic replica placement approaches to improve data availability for DHT-based structured P2P networks. In the neighbor replication approach, the replicas of a data file are stored in the nearest neighbor nodes, i.e. the successor list [1] or the leaf-set [2] of a node responsible for the file. As shown in the simulation results, when node churn rate or the system load is high, this approach cannot maintain the availability of data files effectively which results in a low query hit ratio. Further, the cost of data migrations due to node churn can be high in terms of bandwidth consumption. RelaxDHTs [8] relaxes the requirement of replicating at nearest neighbors by randomly selecting replica nodes from a set of neighbor nodes. Therefore, this approach can reduce the cost of data migrations. However, when the system load is high, this approach still suffers the problem of storage unbalancing between nodes, which results in a low query hit ratio.

Path replication is another file replication method, which chooses replica nodes on the search path from query nodes to a queried node when the queried node suffers high

load [4]. Since the goal of this approach is to reduce the query load for queried nodes, it does not guarantee data availability for backup files when node churn occurs.

The last replication approach is multi publication key replication [11]–[13]. In this approach, a key is associated with a set of r selected points in DHT key space, which correspond to r distinct replica nodes for a file. Symmetric replication [12] is a representative method based on this approach. This approach can achieve a good degree of query load balancing by sending requests to a random replica. However, in order to maintain the persistence of the location of replicas, the cost of replica maintenance for data availability such as the cost of data migration due to node churn or the cost for monitoring the replication degree is high.

A method, which has a similar approach with our algorithm, is Plover [10]. In Plover, nodes are organized in a number of clusters. Each cluster contains a super node and a number of regular nodes, which are physically close. Super nodes form a structured P2P network and maintain metadata of files stored in regular nodes. When a file becomes popular, Plover makes file replication among physically close nodes based on node available capacities. Therefore, Plover can support low-cost and timely consistency maintenance. However, Plover does not provide a method to maintain data availability. Further, super nodes can become bottleneck and easily become a single point of failure in the cluster.

SWARM [14] also organize nodes in clusters based on node interest and proximity and determines the placement of a file replica based on the accumulated query rates of nodes to reduce query overhead. However, SWARM does not guarantee the availability of data files due to the join/leave of nodes.

Bhagwan et al. [17] has investigated the use of erasure codes to improve the level of file availability in Gnutella, an unstructured peer-to-peer network. In [18], Friedman et al. have proposed replicated erasure codes (REC) for storage in P2P networks, which does not increase the storage overhead significantly but reduce the traffic for repair operation. Our system could benefit from the use of the proposed REC code to reduce the cost of file maintenance. However, different from our method, these works do not tackle the problem of node overloading, which may prevent proper data replication.

In [19], the authors proposed a replication method for file availability, which allows a node responsible for a file to select replica nodes from a consistent set based on node availability. The consistent set contains nodes, which are closest to the responsible node. Node availability is calculated based on the Mean Time To Failure and the Mean Time To Recover. The proposed method differs from our method in two folds. Firstly, our method organizes a P2P network as a number of clusters and chooses nodes, which have high available storage capacity in each cluster as new replicas. Thus, our method can achieve a good degree of load balancing. Secondly, our backup method utilizes erasure code to create data chunks of each file and store/maintain data chunks of a file in a number of nodes. As a result, our backup method using erasure code can improve file availability in comparison with whole file replication method [9], [17].

5. Conclusion and Future Works

This paper proposed a cluster-based backup scheme for DHT-based file backup systems. Different from conventional data backup approaches for P2P networks, we organize the network as clusters and store data chunks of a file within nodes in a cluster. We proposed a maintenance method that guarantees file availability by utilizing erasure coding and maintaining the number of data chunks of a file above a threshold value. We also proposed a method for cluster information update, which allows nodes in a cluster to choose high-capacity nodes to store data chunks for file backup. Therefore, with high probability, our proposed system can find backup nodes that are capable to store data chunks for file backup and it results in high query hit ratio of proposed method even when the system load and the rate of node churn are high. The simulation results demonstrate the efficiency of our solution in terms of data lookup success rate and the chunk maintenance cost compared with conventional solutions.

In our future research, we will work on the problem of query overhead by replication of data chunks inside clusters. Another problem to be tackled is the deletion of unnecessary files when a node becomes overloaded. We also implement the proposed scheme in a testbed system and develop applications based on the scheme.

Acknowledgments

This work was partially supported by JSPS KAKENHI Grant Number JP17K00134.

References

[1] S. Ratnasamy, P. Francis, M. Handley, and R. Karp, "A scalable content-addressable network," Proc. ACM SIGCOMM'01, pp.161–172, San Diego, CA, Aug. 2001.
[2] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrisnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," Proc. ACM SIGCOMM'01, San Diego, CA, pp.149–160, Aug. 2001.
[3] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," Proc. IFIP/ACM International Conf. on Distributed Systems Platforms, pp.329–350, Nov. 2001.
[4] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz, "Tapestry: A global-scale overlay for rapid service deployment," IEEE J. Sel. Areas Commun., pp.41–53, Sept. 2006.
[5] M. Landers, H. Zhang, and K.-L. Tan, "Peerstore: Better performance by relaxing in peer-to-peer backup," Proc. 4th International Conference on Peer-to-Peer Computing, pp.72–79, Washington, DC, USA, 2004.
[6] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale persistent peer-to-peer storage utility," Proc. The 18th ACM Symposium on Operating Systems Principles (SOSP), pp.188–201, Alberta, Canada, Oct. 2001.

[7] J. Pang, P.B. Gibbons, M. Kaminsky, S. Seshan, and H. Yu, "Defragmenting DHT-based distributed file systems," Proc. 27th International Conference on Distributed Computing Systems (ICDCS'07), p.14, Toronto, ON, 2007.
[8] S. Legtchenko, S. Monnet, P. Sens, and G. Muller, "RelaxDHT: A churn-resilient replication strategy for peer-to-peer distributed hash-tables," ACM Trans. Auton. Adapt. Syst., vol.7, no.2, pp.28:1–28:18, July 2012.
[9] M. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," J. ACM, vol.36, no.2, pp.335–348, April 1989.
[10] H. Shen and Y. Zhu, "Plover: A proactive low-overhead file replication scheme for structured P2P systems," Proc. IEEE ICC, pp.5619–5623, May 2008.
[11] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher, "Adaptive replication in peer-to-peer systems," Proc. 24th International Conference on Distributed Computing Systems, pp.360–369, March 2004.
[12] A. Ghodsi, L.O. Alima, and S. Haridi, "Symmetric replication for structured peer-to-peer systems," Proc. 3rd International Workshop on Databases, Information Systems, and Peer-to-Peer Computing, pp.74–85, Trondheim, Norway, Aug. 2005.
[13] Z. Trifa and M. Khemakhem, "A novel replication technique to attenuate churn effects," Peer-to-Peer Netw. Appl., vol.9, no.2, pp 344–355, 2016.
[14] H. Shen, G. Liu, and H. Chandler, "Swarm intelligence based file replication and consistency maintenance in structured P2P file sharing systems," IEEE Trans. Comput., vol.64, no.10, pp.2953–2967, Jan. 2015.
[15] J. Ledlie and M. Seltzer, "Distributed, secure load balancing with skew, heterogeneity, and churn," Proc. INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Comunications Societies, vol.2, pp.1419–1430, March 2005.
[16] K. Calvert, M. Doar, and E.W. Zegura, "Modeling Internet topology," IEEE Commun. Mag., vol.35, no.6, pp.160–163, June 1997.
[17] R. Bhagwan, S. Savage, and G.M. Voelker, "Replication strategies for highly available peer-to-peer storage systems," Proc. FuDiCo 2002, pp.40–49, June 2002.
[18] R. Friedman, Y. Kantor, and A. Kantor, "Replicated erasure codes for storage and repair-traffic efficiency," Proc. 14th IEEE International Conference on Peer-to-Peer Computing, pp.1–10, Sept. 2014.
[19] K. Kim and D. Park, "Reducing replication overhead for data durability in DHT based P2P system," IEICE Trans. Inf. & Syst., vol.E90-D, no.9, pp.1452–1455, Sept. 2007.
[20] N.D. Nghia, T.X. Hoang, and N.H. Son, "A cluster-based file replication scheme for DHT-based file backup systems," Proc. International Conference on Advanced Technologies for Communications (ATC), pp.204–209, Oct. 2016.

Hoai Son Nguyen received the B.Eng., B.Eng. and Dr. Eng. degrees from the University of Tokyo, Japan, in 2001, 2003, and 2006 respectively. He is currently a lecturer of the Faculty of Information Technology at VNU-University of Engineering and Technology. His research interests are in the area of mobile wireless networks, P2P overlay networks and Cyber physical systems.

Dinh Nghia Nguyen received the B.Eng. degree from People Security Academy, Vietnam and the M.E. degree from VNU-University of Engineering and Technology, in 2001 and 2006 respectively. He is currently working for the Foreign language and Informatic Center, People Security Academy, Vietnam. He is also a PhD student in the Faculty of Information Technology at VNU-University of Engineering and Technology.

Shinji Sugawara received B.Eng., M.Eng., and Dr.Eng. degrees from Tokyo Institute of Technology, in 1994, 1996, and 1999, respectively. In 1999, he joined the University of Electro-Communications, Tokyo, as an assistant professor, and then in 2005, joined Nagoya Institute of Technology, Japan as an associate professor. From January 2006 to January 2007, he concurrently was a visiting researcher in the University of California, Irvine. In April in 2013, he joined Chiba Institute of Technology, Narashino, Japan as a professor. He is interested in computer communication network, contents retrieval, and distributed systems. He is a member of IEEE and ACM.