CFR: A Peer-to-Peer Collaborative File Repository System

Meng-Ru Lin, Ssu-Hsuan Lu, Tsung-Hsuan Ho, Peter Lin, and Yeh-Ching Chung[1]

Department of Computer Science, National Tsing Hua University
Hsin-Chu, Taiwan300, ROC
{mrlin, shlu, anson}@sslab.cs.nthu.edu.tw, peter@dr-lin.com, ychung@cs.nthu.edu.tw

Abstract. Due to the high availability of the Internet, many large cross-organization collaboration projects, such as SourceForge, grid systems etc., have emerged. One of the fundamental requirements of these collaboration efforts is a storage system to store and exchange data. This storage system must be highly scalable and can efficiently aggregate the storage resources contributed by the participating organizations to deliver good performance for users. In this paper, we propose a storage system, Collaborative File Repository (CFR), for large scale collaboration projects. CFR uses peer-to-peer techniques to achieve scalability, efficiency, and ease of management. In CFR, storage nodes contributed by the participating organizations are partitioned according to geographical regions. Files stored in CFR are automatically replicated to all regions. Furthermore, popular files are duplicated to other storage nodes of the same region. By doing so, data transfers between users and storage nodes are confined within their regions and transfer efficiency is enhanced. Experiments show that our replication can achieve high efficiency with a small number of duplicates.

Keywords: peer-to-peer, storage system, Coupon Collection Problem, CFR.

1 Introduction

The exploding growth of the Internet has enabled organizations across the globe to share resources and collaborate in large scale projects such as SourceForge [21], SEEK[20], and grid systems [1] [5] [11] [25], etc. One of the most fundamental needs of these types of projects is a platform to store and exchange data. A storage system is needed for keeping and distributing the large amounts of source codes, programs, and documentations. To construct such a storage system, machines contributed by volunteering organizations are used to store and mirror the generated data. How to build a scalable and efficient storage system to aggregate the resources contributed by the participating organizations has been an active research issue.

The peer-to-peer computing has received much attention in the past few years. Pioneering applications such as Napster [16] and KaZaA [9] offered platforms for

[1] The corresponding author

users to easily exchange files without a centralized storage. The second generation of peer-to-peer storage systems [2] [10] [15] [18], mostly built on top of structured routing schemes [19][22], further provide mechanisms to guarantee on object location, and adopt more sophisticated replication and caching schemes.

The benefits of peer-to-peer techniques include scalability, fault tolerance, resource sharing, and load balancing among the participating machines. These appealing properties closely match the requirements of storage systems used in large scale collaboration projects mentioned above.

In this paper, we propose a scalable, loosely coupled, and efficient storage system, Cooperative File Repository (CFR), for large scale collaboration projects. The CFR consists of two modules, overlay management and file management modules. The overlay management module maintains connectivity between the participating nodes using a two-layer overlay network. The file management module provides an interface for users to access CFR and manages the files stored in CFR. Replicas are automatically created for all files stored in CFR. Caching is employed to further enhance performance. CFR achieves scalability by incorporating peer-to-peer techniques to aggregate the contributed storage nodes. Efficiency is achieved by exploiting the geographic locality of the storage nodes. Using the region overlay, CFR can replicate files to storage nodes in all geographic areas.

To evaluate the performance of CFR, both simulation analysis and experimental test are conducted. Simulation results verify that our proposed caching scheme can effectively reduce the average download time compared to the one without caching scheme. For the experimental test, we implement CFR on Taiwan UniGrid [25]. Different region configurations are implemented and the top 10 download files from the SourceForge site are used as the test data set. The experimental result shows that the downloading time of the 4-region configuration is almost 3 times faster than that of the 1-region configuration, that is, the region concept of CFR can enhance the performance of file downloading.

The remainder of this paper is organized as follows. In Section 2, we discuss various systems that are related to our system. In Section 3 we briefly describe the system overview of our CFR. In Sections 4 and 5, we introduce the overlay management and file management of CFR, respectively. The simulation results are presented in Section 6. In Section 7, we perform the experimental test on Taiwan UniGrid.

2 Related Work

Many peer-to-peer data storage systems have been proposed in the past, and there are quite a few papers on comparisons of various peer-to-peer file sharing/storage applications published [6] [7]. CFS [2] is a Unix-style read only file system layered on top of the Chord [22] [23] protocol. A DHash layer lies between the file system and Chord to handle block management. OceanStore [10] is a persistent wide-area transactional storage, layered on top of its own probabilistic routing protocol. OceanStore applies erasure coding to files, splitting them into multiple blocks, to achieve robustness. PAST [18] is a large scale persistent storage system layered on the Pastry [19] protocol. PAST can be layered on other routing protocols with some loss of locality and fault resilience properties. All of the storage systems mentioned above create replicas to the files or blocks stored in the system and employ caching. IVY [15] is a log-based file system that supports concurrent write operations. IVY, like CFS, uses Dhash to store the logs. Kelips [4] is a file system layered on its own routing scheme with $O(1)$ lookup time. The fast lookup, however, comes at the cost of larger memory usage and background communication overhead.

CFR shares many similarities with PAST. Like PAST, CFR stores and replicates whole files, and is not bounded to a specific routing scheme. Unlike PAST, we do not rely on the underlying routing protocol to take locality into consideration. Our system partitions the participating nodes into groups, like Kelips, but uses different partition scheme. Kelips uses hashing to determine the group of a node while ours is based on geographic locality.

Many past works have proposed different ideas of using hierarchical multiple ring topologies in overlay networks. HIERAS [26] and [14] are both routing schemes that adopt this topology. In [14], the participating peers are organized into multiple layers of rings with separate identifier spaces to reflect administrative domains and connectivity constraints. Boundary Chord [8] is a replica location mechanism used in grid environments. Boundary Chord adopts a two-layer multiple ring topology to group nodes according to logical domains. In comparison with these systems, CFR adopts a two-layer hierarchy of multiple rings.

3 System Overview

Figure 1 shows the system architecture of CFR and the functions offered by the system components. The CFR system consists of two modules: Overlay Management Module (OMM) and File Management Module (FMM).

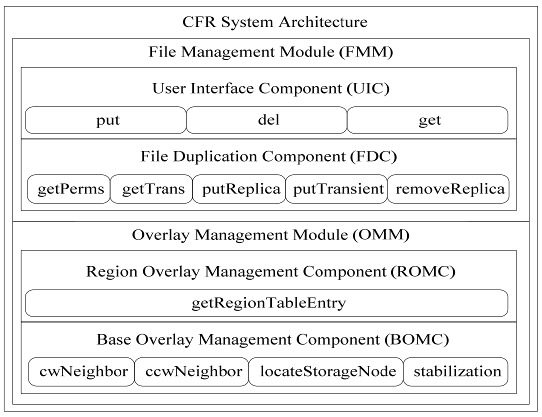

CFR System Architecture
File Management Module (FMM)
User Interface Component (UIC)
put
del
get
File Duplication Component (FDC)
getPerms
getTrans
putReplica
putTransient
removeReplica
Overlay Management Module (OMM)
Region Overlay Management Component (ROMC)
getRegionTableEntry
Base Overlay Management Component (BOMC)
cwNeighbor
ccwNeighbor
locateStorageNode
stabilization

Figure 1. The system architecture of CFR.

OMM is responsible for maintaining connectivity between the participating storage nodes using a two-layer overlay network. The two-layer overlay network consists of two overlays, the base overlay and the region overlay. These two overlays are maintained by the Base Overlay Management Component (BOMC) and Region Overlay Management Component (ROMC), respectively. ROMC maintains the required routing information in a data structure called the region table.

FMM is used for providing functions that are related to files in CFR. FMM consists of two components: the User Interface Component (UIC) and the File Duplication Component (FDC). UIC provides an interface for users to access the files which are stored in CFR. Duplications of files in CFR are automatically created in order to enhance performance and increase availability. The File Duplication Component (FDC) is responsible for creating the duplications.

4 The Overlay Management of CFR

In this section, we will describe the overlay management of CFR. It can be divided into the base overlay and the region overlay.

4.1 The Base Overlay

The purpose of the base overlay is to route messages between any two storage nodes in the system. The base overlay is constructed and maintained by BOMC. In the base overlay, each participating storage node has a node ID that is obtained by hashing the IP address of the node using a consistent hash function, such as SHA-1 [3] or MD5 [17]. Using this method, participating storage nodes are organized as a ring, the *base* ring, according to their IDs.

4.2 The Region Overlay

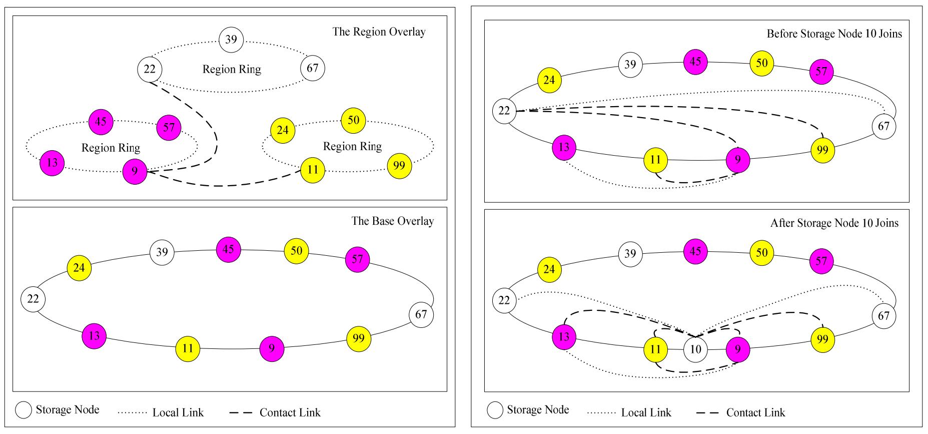
4.2.1 Regions

The basic concept of region is inspired by mirroring scheme on the internet such as SourceForge. User usually can choose a server to download file according to their own geographic locality to achieve efficient downloading. Therefore, the geographic locality can be interpreted as network locality in two end hosts connected to the Internet. In [24], it is shown that topology of the Internet today obeys the Power Law and consists of several dense autonomous system clusters.

We adopt a model to capture the scenario that we mentioned above. We assume that the connection between two participants (storage nodes or users) of CFR is efficient if they are in the same geographic area. In our model, all storage nodes and users, both end hosts in the Internet, are partitioned into disjoint sets called *regions*. We assume that the partition reflects geographic locality.

4.2.2 Construct and Maintain the Region Overlay

Constructing the region overlay can allow the participating storage nodes to contact other storage nodes that are in different regions quickly. This ability aids the file duplication procedures to select target storage nodes to replicate desired files. Details of the file duplication procedures are described in Section 5.



(a) (b)

Figure 2. (a). An example of region overlay with 3 regions. (b). An example of the join process.

We now describe the construction and maintenance procedures of ROMC. First we will introduce some terms and variables that will be used. *R* denotes the total number of regions in the system. Nodes that belong to the same region are called *locals* of each other. Nodes that belong to different regions are called *contacts* of each other. A *link* is an ID-to-address mapping, used to convert node ID to actual network address. Links that point to locals are called *local links*. Links that point to contacts are called *contact links*. Links that are required to form the region overlay which are stored in the *region table* of the participating nodes.

To form the region overlay, each node stores and maintains *R* links in their region tables. The local links in the region table of each node connect nodes from the same region into a ring, called the *region ring*. The region overlay is essentially made up of *R* interconnected region rings. Figure 2(a) shows a system with 3 regions. Storage node 9 stores and maintains 3 links in its region table. A local link points to the clockwise neighbor in its region ring, node 13. Two contact links point to the closest contacts from the remaining two regions in the base ring, nodes 11 and 22, respectively.

A node constructs its region table when it first joins the system, and maintains its region table throughout its lifetime in the system.

Figure 2(b) shows an example of the join process. In Figure 2(b), storage node 10 joins the system. As shown on top of Figure 2(b) all storage nodes between storage node 9 and storage node 67 have a link to storage node 22 before storage node 10 joins. The region table of storage node 9 contains links to storage nodes 11, 13, and 22. After storage node 10 joins, all nodes between storage node 9 and storage 67 are affected. As shown on the bottom of Figure 2(b), the region table of storage node 10 contains links to storage nodes 11, 13, and 22. These links are obtained from storage node 9. All the links that point to storage node 22 are modified to point to storage node 9.

5 The File Management of CFR

In this section, we give detailed descriptions of file management procedures in CFR. Files that are stored in CFR can be classified into two types: *permanent file*, and *transient file*. Permanent file will stay in the system until a remove operation is performed on it. Each transient file has a *lifetime* to determine how long it can stay in the system, and will be removed from the system when the system time exceeds its lifetime. A permanent file is associated with a data structure called *permanent table*, which contains all the necessary file management information about a permanent file. Likewise, a transient file is associated with a *transient table* which contains the necessary information about a transient file. The storage space of each storage node is divided into to two areas: *local* and *cache areas*. Permanent files are stored in the local areas of storage nodes, and transient files are stored in the cache areas.

Table 1. An example of a permanent table

filename	App.tgz		
fileID	8		
permNodes	8	11	22
caches	24	50	
hort	359		
long	50		
path	/opt/cfr/local		

Table 2. An example of a transient table

filename	App.tgz
fileID	8
lifetime	50000
path	/opt/cfr/cache

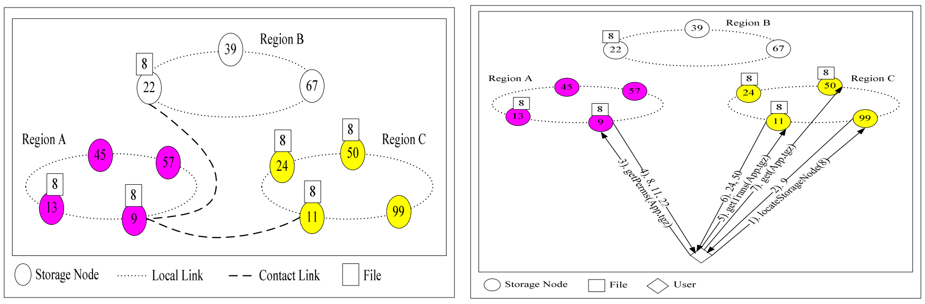
Table 1 shows a permanent table. The filename and the fileID field record the name of the file and the hash value of the filename. Each file will be replicated, and the permNodes field records the storage nodes in different regions that store the replicas when the caches field records the storage nodes in the same region that store the replicas. The long and short fields record the long term and short term access rates of that file, respectively. The path field stores the physical location of the file. Table 2 shows a transient table. The filename, fileID, and path fields are the same as the fields in the permanent table. The lifetime field stores the lifetime of that transient file.

5.1 Insert Files and Create Duplicates in CFR

The put function provides by UIC allows users to insert files into CFR. In CFR, a file will first be put to the node, n_i, whose id is closest to fileID. After the first stage of insertion is completed, node n_i will replicate files to nodes in other regions according to its contact link information.

Transient files are created for reducing the load of the storage nodes hosting popular files as proposed in [12]. In order to cope with this phenomenon, we record the long term download rate, in the scale of days, of each file in the long field of its permanent tables. The transient file will be created when one of the download rates of that file exceeds its threshold.

Figure 3(a) shows an example of file insertion and duplication process. The file App.tgz, which is the same file in Table 1, is inserted into a system with 3 regions. Since the fileID of App.tz is 8, its home is storage node 9. Storage node 9 uses its region table to create two replicas on storage nodes 11 and 22 according to the described creation procedure.



(a) (b)

Figure 3. (a). An example of file insertion and duplication. (b). An example of file retrieval.

5.2 Retrieve and Remove Files in CFR

The get function provided by UIC allows users to retrieve files from CFR. To retrieve a file f_j from CFR, a user u_i first finds the home of f_j, n_h. After n_h is found, u_i invokes the getPerms function on n_h to find the list of storage nodes that stores f_j as a permanent file, and selects the storage node that belongs to the same region as herself. Let this storage node be n_r. u_i invokes the getTrans function on n_r to obtain the list of caches of f_j. u_i then chooses a storage node from all the caches and n_r with equal probability. This will evenly distribute the requests among all the storage nodes that stores f_j or transient file of f_j in the same region.

The remove operation is similar to the get operation. A user first invokes the del function on the home of a file, n_h. n_h then finds all the storage nodes that store replicas and transient file of that file from the permanent tables, and issues requests to remove the files from their storage space. Figure 3(b) shows an example of the file retrieval process.

5.3 Dealing with Storage Node Dynamics

To ensure users can always locate their desired files, dynamic storage nodes must be considered. The addition of new storage nodes and the departure of existing storage nodes will cause files to migrate to different homes. If no corresponding actions are taken, future requests will be routed to their new homes and dropped because the new homes are unaware of their existence. However, migration of all files from one storage node to another will be very costly especially when the total size of files is large. We use redirection to deal with these problems. A storage node can store only the permanent table of a file and records a link to the storage node that store the actual file. This link is called a *reference*. References are created by storing links instead of local paths in the path field of permanent tables.

A joining or leaving storage node will affect its clockwise neighbor in the base ring. It will also affect the nodes between itself and the closest counter-clockwise storage node in its region. In the case of join and voluntary departure, the affected nodes will be notified. The affected nodes will first create references to deal with the change of topology, and schedule physical file migration to be done in the future.

6 Simulation Results

To evaluate CFR, we implemented a simulator and performed several experiments to further understand its behavior. All simulations were run on an IBM eServer, equipped with two Intel(R) Xeon(TM) 2.40GHz CPUs and 1GB of memory. The OS running on the eServer is Debian. The kernel version is 2.6.

6.1 Expected Number of Hops to Collect All Links

The objective of this experiment is to compare the average number of hops[13] to obtain a complete set of R links to the derived expected number of hops. We would also like to verify that the minimal average value appears when the population of storage nodes in all regions is equal. We only show the results with two and three regions. When the number of regions is larger than three, it is difficult to present the results using graphs. However, all results show similar characteristics.

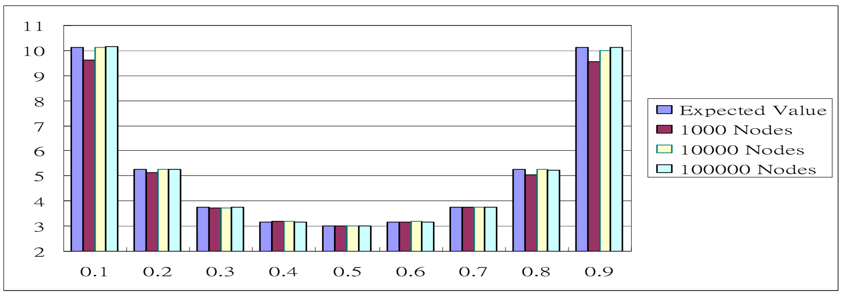


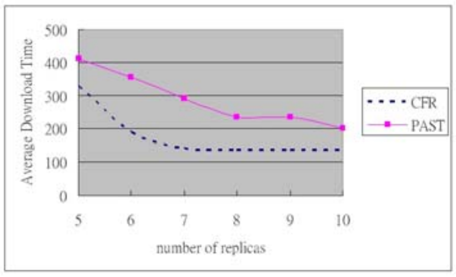
Figure 4. Average number of hops and the expected number of hops to obtain all links in a system consisting of two regions and different number of storage nodes.

Figure 4 shows the average number of hops in a system with two regions, with different node proportions and storage node populations. The x-axis is the proportion of the first region. We can see that all results are close to the expected value. Note that the larger storage node population, the closer the average is to the derived value. This is because the distribution of nodes over the identifier space is more uniform as the number of nodes increase. Also note that the lowest expected value and average values occur at the point where the proportions of the nodes are equal (0.5), which concurs with our derived result.

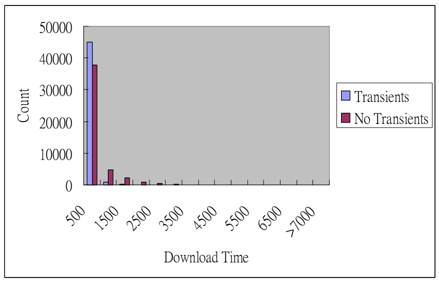
6.2 Evaluation of File Management of CFR

The objective of next experiments is to evaluate the proposed replication strategy and to compare the proposed strategy to PAST. The reason PAST is chosen is because it shares most similarity with CFR. We use the download statistics from the “top 100 downloaded projects in 7 days” web page available from the SourceForge website.

Using this data, we simulated our replication strategy and compare it with the replication strategy of PAST. The system consists of five hundred nodes. The average download time of around 45000 downloads with varying number of replicas created for each file inserted in both CFR and PAST, are shown in Figure 5(a). As shown in the figure, download time decreases as the number of replicas created for both systems. We can see that CFR achieves lower average download time than PAST using the same number of replicas.



(a)



(b)

Figure 5. (a) The average download time of CFR versus PAST with different number of replicas. (b). Comparisons the transfer time between with transient files and without transient files

Next we evaluate the effect of creating transients on performance. In this experiment, the setup is identical to the previous experiment. The result of the experiment shows that the average download time is reduced to about one half when transients are created. Figure 5(b) shows the comparisons the transfer time between with transient files and without transient files.

As shown in Figure 5(b), the use of transient files effectively reduces transfer time. With transient files, it has greatly reduced download time.

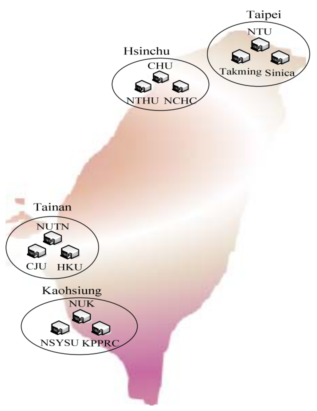
7 Experimental Results

To evaluate the real performance of CFR, we have implemented the CFR system on Taiwan UniGrid [25]. The Taiwan UniGrid is a Grid platform for researchers in

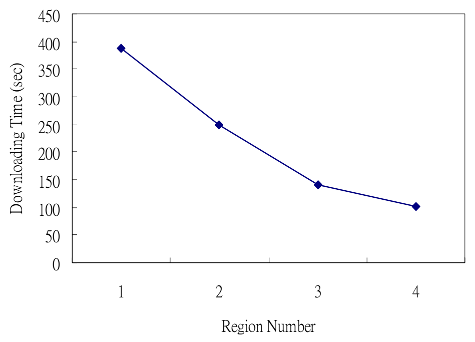
Taiwan to do Grid related research. Currently, the platform contains about 30 sites. We execute the CFR program on 12 sites in 4 cities of Taiwan as shown in Figure 6(a). Each site has 3 storage nodes. We select the top 10 download files, as shown in Table 3, from the sourceforge.net [21] as our test data. To measure the performance of CFR, we have 4 region configurations, 1, 2, 3, and 4, for these 12 sites. For the 1-region configuration, all sites form a region. For the 2-region configuration, sites in {Taipei, Hsinchu} and {Tainan, Kaoshiung} form a region, respectively. For the 3-region configuration, sites in {Taipei}, {Hsinchu}, and {Tainan, Kaoshiung} form a region, respectively. For the 4-region configuration, sites in each city form a region. For each region configuration, a download program is executed in each site to randomly decide whether a client needs to download a particular program or not.

Table 3. Top 10 downloads from sourcesforge.net

Filename	Size (bytes)
7-Zip_Portable_4.42_R2.paf.exe	1193218
7z443.exe	862846
aresregular195_installer.exe	1253674
audacity-win-1.2.6.exe	2228534
Azureus_2.5.0.0_Win32.setup.exe	8799656
DCPlusPlus-0.698.exe	3836577
eMule0.47c-Installer.exe	3534076
eMulePlus-1.2a.Binary.zip	3047952
gimp-2.3.12-i586-setup.zip	14267302
Shareaza_2.2.3.0.exe	4366779



(a)



(b)

Figure 6. (a) Testbed map of CFR in TANET of Taiwan. (b) The average downloading time against region number.

Figure 6(b) shows the average downloading time against the region number. From Figure 6(b), we observe that the overall downloading time goes down while the number of regions increases. Since the region partitioning exploits the geographical relationships of sites, the experimental result also shows that the downloading time of the 4-region configuration is almost 3 times faster than that of the 1-region configuration.

8 Conclusions and Future Work

In this paper, we have proposed a scalable, loosely coupled, and efficient storage system, Cooperative File Repository (CFR), for large scale collaboration projects. The main concept of CFR is to use peer-to-peer techniques to achieve scalability, use a two-layer hierarchy managing participating organizations to eliminate centralized administration authority, and use the geographic locality of the storage nodes and caching mechanism to achieve the efficiency. The simulation and experimental results confirm that CFR can achieve those goals mentioned above.

From the simulation results, we observe that the CFR can produce the best performance when all regions have the same number of storage nodes. In real situation, the number of storage nodes of regions may not be equal. How to dynamically combine small regions to one larger region or split one larger region to small regions such that each region has approximate the same number of storages node to keep CFR remain efficient is an important issue for the future study.

Acknowledgement

The work of this paper is partially supported by National Science Council, National Center for High-Performance Computing of the Republic of China under contract NSC 95-2752-E-007-004-PAE, NSC 94-2218-E-007-057, and NCHC-KING_010200.

References

1. China Grid, http://www.chinagrid.net
2. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area Cooperative Storage with CFS,” in the *Proceedings of 18th ACM Symposium on Operating Systems Principles*, Oct. 2001, pp. 202-215.
3. FIPS 180-1, Secure Hash Standard, U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, Apr. 1995.
4. I. Gupta, K. Birman, P. Linga, A. Derms, and R. van Renessie, “Kelips: Building and Efficient and Stable P2P DHT through Increased Memory and Background Overhead,” in the *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS ’03).*
5. grid.org, http://www.grid.org/home.htm
6. R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, “A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems,” in the *Proceedings of International Conference on Information Technology: Coding and Computing*, Apr. 4-6, 2005, vol: 2, pp. 205-213.

7. H. C. Hsiao and C. T. King, “Modeling and Evaluating Peer-to-Peer Storage Architecture,” in the *Proceedings of International Parallel and Distributed Processing Symposium*, Apr. 14-19, 2002, pp. 24-29.
8. H. Jin, C. H., and H. Chen,” Boundary Chord: A Novel Peer-to-Peer Algorithm for Replica Location Mechanism in Grid Environment,” in the *Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN 2005),* Dec. 2005, Las Vegas.
9. Kazaa. http://www.kazaa.com
10. J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, “Oceanstore: An Architecture for Global-Scale Persistent Storage,” in the *Proceedings of 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.
11. LCG, http://lcg.web.cern.ch/LCG/
12. N. Leibowitz, M. Ripeanu, and A. Wierzbicki, “Deconstructing the Kazaa Network,” in the *Proceedings of 3rd IEEE Workshop on Internet Applications*, Jun. 2003.
13. M. R. Lin, “CFR: A Peer-to-Peer Collaborative File Repository System,” National Tsing Hua University, Dept. of Computer Science, Master Thesis, Taiwan, 2006.
14. A. Mislove, and P. Druschel, “Providing Administrative Control and Autonomy in Structured Peer-to-Peer Overlays,” in the *Proceedings of International Workshop on Peer-to-peer Systems*, Feb. 2004.
15. A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, “Ivy: A Read/Write Peer-to-Peer File System,” in the *Proceedings of International 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.
16. Napster, http://www.napster.com
17. R. Rivest, "Message Digest Algorithm MD5", RFC 1321, Apr. 1992.
18. A. Rowstron and P. Druschel, “Storage Management and Caching In PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility,” in the *Proceedings of 18th Symposium On Operating Systems Principles (SOSP ’01)*, Oct. 2001.
19. A. Rowstron and P. Druschel, “Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems,” in the *Proceedings of 18th IFIP/ACM International Conference on Distributed Systems Platforms*, Nov. 2001, pp.329-350.
20. SEEK, http://seek.ecoinformatics.com
21. SourceForge.net, http://sourceforge.net
22. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, “Chord: A Scalable Peertopeer Lookup Service for Internet Applications,” in the *Proceedings of conference on Applications, technologies, architectures, and protocols for computer communications SIGCOMM '01*, 2001, Volume 31, Issue 4, pp. 149-160.
23. I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, “Chord: a scalable peer-to-peer lookup protocol for Internet applications,” in the IEEE/ACM Transactions on Networking, Feb. 2003, Volume 11, Issue 1, pp. 17-32.
24. G. Sagie and A. Wool, “A clustering approach for exploring the Internet structure,” in the *Proceedings of conference on 23rd IEEE Convention of Electrical & Electronics Engineers*, Sep. 2004.
25. Taiwan UniGrid, http://www.unigrid.org.tw/
26. Z. Xu, R. Min, and Y. Hu, “HIERAS: A DHT Based Hierarchical P2P Routing Algorithm,” in the *Proceedings of International Conference on Parallel Processing*, Oct. 2003.