CORP: A Cooperative File Replication Protocol for Structured P2P Networks

Haiying Shen
Department of Electrical and Computer Engineering
Clemson University
Clemson, SC 29631, USA
shenh@clemson.edu

Abstract

File replication and file consistency maintenance are two important protocols for achieving overall high performance of peer-to-peer file sharing systems. In spite of the significant dependency of consistency maintenance on file replication, there has been little file replication research devoted to tackling the dependency issue. Most traditional file replication protocols focus on improving file lookup efficiency without considering the subsequent consistency maintenance. This paper presents a COoperative file Replication Protocol (CORP). CORP observes a set of factors including file update rate, file popularity, node available capacity, file load and node locality that affect the efficiency of consistency maintenance. It considers these factors to achieve high efficiency not only in file replication but also in consistency maintenance. CORP generates less replicas and optimally chooses infrequently-updated or popular files to replicate. In addition, it makes file replicas in physically close nodes. Further, CORP can dynamically adjust the number of replicas based on ever-changing file popularity and visit pattern. Simulation results demonstrate the efficiency and effectiveness of CORP in comparison with other file replication protocols. It dramatically reduces the overhead of both file replication and consistency maintenance. Moreover, it yields significant improvement in reducing overloaded nodes.

1. Introduction

Over the past years, the immense popularity of the Internet has produced a significant stimulus to peer-to-peer (P2P) networks. The distributed and self-organizing features of a P2P network allow it to grow considerably without performance degradation. One of the widely used P2P applications is file sharing systems such as BitTorrent [1]. A recent study [2] shows that between 49 and 83 percent of all Internet traffic is P2P related, with peaks of over 95 percent during the night. In a file sharing system, if a node receives a large volume of file requests at one time, it would become a hot spot, leading to delayed response. For example, total of 5.5 million viewers used P2P TV platforms such as PPLive to watch the opening ceremony of Olympics in 2008 [3]. If the popular files are in one node, they could exhaust the capacity of the node.

File replication protocol to replicate a file to other nodes is an effective strategy to avoid such hot spots [4–12]. Though files such as video and mp3 files are static or infrequently-changed, some other files are frequently changing. For instance, the news for the number of medals earned by each country is frequently updated during the Olympic period. Furthermore, other P2P applications need consistency support to deliver frequently-updated contents, such as directory service [13], online auction [14], remote collaboration [15], shared calendar [16], P2P web cache [17] and online games [18]. Thus, file consistency maintenance protocol is indispensable to file replication to keep the consistency between a file and its replicas. The cost of consistency maintenance mainly depends on a number of factors determined in the phase of file replication, including the number of replicas, the update rate of the file, and the distances between the file's owner and its replica nodes. Considering this dependency issue, a file replication protocol should be efficient and farseeing to facilitate scalable, low-cost and timely consistency maintenance. However, there has been little file replication research devoted to tackling the dependency issue of consistency maintenance.

Most previous file replication methods in structured P2P networks focus on improving file lookup efficiency without considering subsequent file consistency maintenance. Specifically, most methods determine replica nodes based on node ID [4–7] or location [8–11]. ID-based methods select replica nodes based on the relationship between node ID and the file's ID, and location-based methods choose replica nodes in a file query path between a file requester and a file provider. Both groups of methods concentrate on avoiding hot spots to improve query efficiency, but neglect the impact of file replication on file consistency maintenance. They fail to simultaneously take into account the non-uniform and time-varying file popularity and update rate, node available capacity and file load, as well as the locality feature of structured P2P networks during file replication, which are vital factors for the efficiency of both file replication and consistency maintenance.

This paper presents a COoperative file replication Protocol (CORP) that not only achieves high efficiency in file replication but also supports efficient file consistency maintenance. Moreover, CORP does not compromise the effectiveness of file replication in reducing hot spots and improving lookup efficiency. Essentially, CORP is characterized by the following features:

- *The consideration of file update frequency and popularity.* It makes less replicas for frequently-updated or infrequently-visited files in order to reduce file consistency maintenance overhead and meanwhile increase replica utilization.
- *The consideration of node available capacity and file load.* It avoids exacerbating the hot spot problem by proactively choosing nodes with sufficient capacity for replicas. More importantly, CORP reduces the number of replicas by optimally selecting files to replicate and replica nodes.
- *The consideration of node locality.* It replicates files in physically close nodes without relying on an extra network structure to achieve high efficiency of both file replication and consistency maintenance.
- *The adaptiveness of file popularity and visit pattern.* It dynamically adjusts the number of replicas to handle ever-changing file popularity and visit pattern for efficient file replication and consistency maintenance.

The rest of this paper is structured as follows. Section 2 presents a concise review of representative file replication approaches for structured P2P networks. Section 3 presents the CORP file replication protocol. Section 4 shows the performance of CORP in static situation as well as dynamic situation in comparison with other file replication protocols. Section 5 concludes this paper.

2. Related Work

Driven by tremendous advances of P2P file sharing systems, numerous file replication protocols have been proposed. One group determines replica nodes based on IDs. PAST [4] is a P2P-based file system for large-scale persistent storage service. In PAST, each file is replicated in a number of nodes whose IDs match most closely to the file's ID. PAST has load balancing algorithm for non-uniform node storage capacities and file sizes, and it uses caching along the lookup path for non-uniform popularity of files to minimize fetch distance and balance query load. Similarly, CFS [5] is a P2P read-only storage system. CFS stores blocks of a file and spreads blocks evenly over the available servers to prevent large files from causing unbalanced use of storage. It uses distributed hash function to replicate each block on nodes immediately after the block's successor on the Chord ring [19] in order to increase file availability. CFS also has a caching protocol that leaves cached copies of file along the query path. LessLog [6] constructs a lookup tree based on node IDs to determine the locations of replica nodes. HotRoD [7] is a structured P2P based architecture with a replication scheme, in which a group of successive peers on a ring are replicated and rotated over the ID space. By tweaking the degree of replication, it can trade off replication cost for access load balancing.

Another group replicates files based on node location by choosing replica nodes along a file lookup path between a client and a server (including the clients). Beehive [8, 20] achieves $O(1)$ lookup latency for common Zipf-like query distributions by replicating an object at nodes i hop prior to the server in the lookup path and choosing different i for different objects. It aims to find the minimum replication extend to achieve constant lookup performance. Backslash [9] is a collaborative web mirroring system with file replication and caching methods for flash crowds. A node periodically injects files in its local file collection, and also pushes cache to one hop closer to requester nodes when overloaded. LAR [10] specifies the overloaded degree of a server that a file should be replicated. Overloaded nodes replicate files at the query initiator and create routing hints on the reverse path. Overlook [11] places a replica of a file on a node with most incoming lookup requests for fast replica location. It needs to keep track of client-access history to decide the replica nodes. Squirrel [21] is a distributed P2P web cache. Its key idea is to facilitate mutual sharing of web objects among client nodes by enabling these nodes to export their local caches to other nodes in the corporate network.

However, few work tries to optimize consistency maintenance in the file replication stage. Our previous work [22] addressed this problem by arranging each node to keep track of queries it receives to make replicas of popular files, and remove under-utilized replicas. CORP deals with the problem from a completely different direction. It observes a set of factors that affect the efficiency of consistency maintenance and takes into these factors in file replication to optimize consistency maintenance. Rigid replica node determination in the ID-based or location-based methods makes it difficult to consider the factors. For instance, without considering the available capacity of replica nodes, the methods may make the hot spot problem even more severe.

Without strict specification of replica nodes, CORP replicates files in physically close nodes based on node available capacity with consideration of file update and visit rates, which leads to efficient file replication and consistency maintenance. CORP is developed based on our preliminary work Plover [23], which only considers node locality and available capacity. Plover considers available capacity only to avoid overloaded nodes rather than reducing the number of replicas. In addition, it needs to build an additional supernode structure. In contrast, CORP mainly aims to reduce the number of replicas by considering capacity, and it does not need to build an extra supernode structure. More importantly, CORP observes and considers other critical factors that affect consistency maintenance to reduce the overhead of consistency maintenance and meanwhile increase replica utilization.

3. Cooperative File Replication

CORP aims to achieve both efficient file replication and consistency maintenance. To realize its goal, it optimally chooses source files to replicate (briefly called *source files*) and replicates them in physically close nodes, and tries to minimize the number of replicas. Specifically, CORP observes and takes into account a set of factors including file popularity, update rate, node available capacity, file load, and node locality in file replication. Figure 1 shows the goal of CORP and the factors its considers during the file replication.

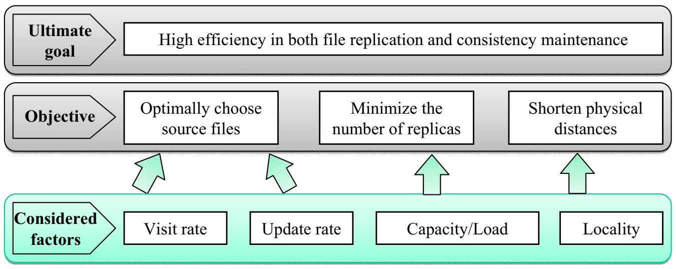

Ultimate goal
High efficiency in both file replication and consistency maintenance
Objective
Optimally choose source files
Minimize the number of replicas
Shorten physical distances
Considered factors
Visit rate
Update rate
Capacity/Load
Locality

Figure 1. The goals of CORP and its considering factors.

To optimally choose source files, COPR considers *file visit rate and file update rate*. An unpopular file does not necessarily have many replicas since the replicas would have low probability to be visited, and vice versa. Thus, a highly-popular file should have more replicas while an unpopular file should have less replicas. Update rate is another important factor to consider in file replication for lightweight consistency maintenance. Different files have different update rates. For instance, files for auction applications may change frequently while video or mp3 files hardly change. Therefore, frequently-updated files should have less replicas in order to reduce the overhead of update message propagation in consistency maintenance.

To minimize the number of replicas, COPR considers *node available capacity and file load*. An overloaded node needs to replicate its highly-popular files to release its file visit load. In most previous replication methods, once a file owner is overloaded due to a file's visits, it replicates the file to a certain node based on ID or location. As a result, many replicas lead to high overhead for consistency maintenance. By considering node available capacity and file load, CORP brings three benefits. First, CORP helps to replicate a file to less nodes to handle all the file's visits. Second, when a node chooses files to replicate, it selects the files with higher load. This reduces the number of replicas since higher-load files release more load than low-load files. Third, CORP avoids overloading replica nodes by replicating a file to a node with sufficient available capacity.

In addition, CORP considers *locality* by discovering physically close nodes to a file's owner for file replication. Short physical distances between a file owner and its replica nodes help to reduce the overhead of file replication and subsequent consistency maintenance. Moreover, it also helps to increase lookup efficiency since a file owner can efficiently forward a file query to its physically nearby replica nodes.

CORP periodically runs three phases. First, overloaded nodes optimally choose source files. Second, overloaded nodes and lightly-loaded nodes report the information of their source files and available capacities to a number of repository supernodes. The information of physically close nodes is marshalled in the same repository supernode. Third, the repository supernodes arrange the source files in overloaded nodes to be replicated to their physically close nodes. Specifically, CORP addresses the following challenges:

(1) When an overloaded node chooses source files to release its load due to file visits, how to consider node capacity and file load to reduce the number of replicas? How to reduce the number of infrequently-visited and frequently-updated source files? (Section 3.1)
(2) When overloaded nodes and lightly-loaded nodes report the information of their source files and available capacities to repository supernodes, how to make sure that the information of physically close nodes is gathered together in a supernode? (Section 3.2)
(3) When repository nodes arrange file replication between overloaded nodes and lightly-loaded nodes, how to take into account the factors of available capacity, file load, file update rate and visit rate to minimize the number of replicas? (Section 3.3)

3.1. Source File Selection for Replication

As the works in [24, 25], we assume that there is only one bottleneck resource for optimization though a node's capacity includes its storage space, bandwidth, CPU, and etc. For simplicity, we represent a node's capacity by the number of bits it can transfer during a time unit, say one second, in response to file queries. We assume that each node has a capacity it is willing to contribute to the system, denoted by C, which is in a percentage of the node's actual capacity. Similarly, a node's load is measured by the number of bits it needs to transfer in response to file queries, denoted by L. A file's visit rate can be measured by the number of visits during one second, denoted by V. To truly reflect a file's popularity by its visit rate, CORP considers the file's current visit rate and past visit rate, and gives more weight to the current visit rate. It introduces a weighted average visit rate of a file as below:

$$V_{new} = \gamma V_{old} + \beta V_{current},$$

where weights γ $(0 \leq \gamma \leq 1)$ and β $(0 \leq \beta \leq 1)$ are discount factors.

Assume node A contains m files, labelled as $\mathbb{F}_A[i](i = 1, 2, \ldots, m)$, then L_A is the sum of the load of these files. That is, $L_A = \sum_{i=1}^{m} L_{\mathbb{F}_A[i]}$, where $L_{\mathbb{F}_A[i]}$ denotes the load of file $\mathbb{F}_A[i]$. The file access delay experienced at a node depends on the overall load on the node measured by the number of bits transferred during a time unit. Therefore, the load of a file on a node is determined by the file's size and visit rate on the node. That is, $L_{\mathbb{F}_A[i]} = S_{\mathbb{F}_A[i]} \times V_{\mathbb{F}_A[i]}$, where $S_{\mathbb{F}_A[i]}$ and $V_{\mathbb{F}_A[i]}$ respectively denote the size and visit rate of file $\mathbb{F}_A[i]$ in node A. We define the update rate of file k in node A, denoted by $U_{\mathbb{F}_A[i]}$, as the number of the file's updates in one second. The update rate can be determined by the same way as visit rate as shown in the above formula. Each node A periodically measures the $V_{\mathbb{F}_A[i]}$ and $U_{\mathbb{F}_A[i]}$, and maintains the $L_{\mathbb{F}_A[i]}$, $S_{\mathbb{F}_A[i]}$, $V_{\mathbb{F}_A[i]}$ and $U_{\mathbb{F}_A[i]}$ of each of its files in a table.

When a node, say node A, doesn't have enough capacity to handle all queries, it needs to replicate its files to release its excess load $\Delta L_A = L_A - C_A$ by choosing a number of source files to be replicated. Based on the objective of CORP to minimize the number of replicas, high-load files should be the choices because their replication can release node A's load quickly and meanwhile reduce the number of replicas.

In addition, the factors of visit rate and update rate should also be considered. For instance, during the 2008 Olympic period, its related news was more popular than the news of 2004 Olympics. During the period, the news of the number of medals earned by each country was updated frequently while the video of the Olympic Opening Ceremony hardly changed. Because the replicas of infrequently-visited files will have low probability to be visited, such files should have less or no replicas in order to reduce idle replicas and consistency maintenance overhead.

On the other hand, frequently-visited files should have more replicas in order to improve lookup efficiency. It also reduces the overhead of file replication due to small sizes of source files. This is because given certain load, higher visit rate means smaller file size. For example, file i's visit rate is 100 and its size is 100 bits and file j's visit rate is 1 and its size is 10000 bits. Although replicating either file can release load of 10000 units, replicating file j with 10000 bits generates higher cost than replicating file i with 100 bits. Moreover, the replica of j will be less frequently used than the replica of file i since j's visit rate is less than i's visit rate. Hence, file i with higher visit rate should be a better choice than j for replication. At the same time, frequently-updated files also should have less replicas in order to reduce consistency maintenance overhead. For instance, Olympic video is a better choice than the news of Olympic medals for file replication during the Olympic period. Avoiding producing replicas for the news saves the cost of the subsequent update propagation to keep the consistency between the source file and its replicas.

Considering all these factors, to choose source files to be replicated, a node orders its files as shown in Figure 2. Specifically, the files in $\mathbb{F}$ are sorted in descending order of their load, the files with the same load are ordered in ascending order of their update rates, and the files with the same update rates are further ordered in descending order of their visit rates.

The selection process has three steps. The first step is to choose the initial source files with the highest loads in order to reduce the number of replicas. The files in $\mathbb{F}$ are selected in a top-down manner until the sum of the loads of the selected source files is no less than the load to release. We represent the selected files of node A by $\mathcal{F}_A = \{\mathcal{F}_A[1], \mathcal{F}_A[2], \ldots, \mathcal{F}_A[\tilde{m}]\}$ ($\mathcal{F} \in \mathbb{F}, 1 \leq \tilde{m} \leq m$), and their corresponding loads are $\{L_{\mathcal{F}_A[1]}, L_{\mathcal{F}_A[2]}, ..., L_{\mathcal{F}_A[\tilde{m}]}\}$. The source nodes are chosen in such a way that $\sum_{i=1}^{\tilde{m}} L_{\mathcal{F}_A[i]} \geq \Delta L_A$. The next step reduces the total update rate of the source files in $\mathcal{F}_A$, thus reducing consistency maintenance overhead. In this step, the rest of files in $\mathbb{F}$ continue to be fetched in the top-down manner. Assume a fetched file is $\mathbb{F}_A[\tilde{m}+1]$. It is compared to the files in $\mathcal{F}_A$ one by one from $\mathcal{F}_A[1]$ to $\mathcal{F}_A[\tilde{m}]$. If $U_{\mathbb{F}_A[\tilde{m}+1]} < U_{\mathcal{F}_A[k]}$ $(1 \leq k \leq \tilde{m})$ and the condition $\sum_{i=1}^{\tilde{m}} L_{\mathcal{F}_A[i]} \geq \Delta L_A$ is still satisfied after replacing $\mathcal{F}_A[k]$ with $\mathbb{F}_A[\tilde{m}+1]$, file $\mathcal{F}_A[k]$ is replaced by file $\mathbb{F}_A[\tilde{m}+1]$. In the third step, if there is no such file in $\mathcal{F}$ that can be replaced, the file $\mathbb{F}_A[\tilde{m}+1]$ is compared to the files in $\mathcal{F}_A$ in a sequential order again. This step is to increase the total visit rate of the source files in $\mathcal{F}_A$, thus increasing the utilization of file replicas and reducing file replication cost. If $U_{\mathbb{F}_A[\tilde{m}+1]} = U_{\mathcal{F}_A[k]}$ and $V_{\mathbb{F}_A[\tilde{m}+1]} > V_{\mathcal{F}_A[k]}$ $(1 \leq k \leq \tilde{m})$, and the condition $\sum_{i=1}^{\tilde{m}} L_{\mathcal{F}_A[i]} \geq \Delta L_A$ is still satisfied after replacing $\mathcal{F}_A[k]$ with $\mathbb{F}_A[\tilde{m}+1]$, file $\mathcal{F}_A[k]$ is replaced by file $\mathbb{F}_A[\tilde{m}+1]$.

Consequently, source files constitute a subset of node A's resident files, satisfying the following condition.

$$\text{minimizes } \tilde{m}; \quad \text{minimizes } \sum_{i=1}^{\tilde{m}} U_{\mathcal{F}_A[i]}; \quad \text{maximizes } \sum_{i=1}^{\tilde{m}} V_{\mathcal{F}_A[i]} \tag{1}$$

$$\text{subject to} \quad (L_A - \sum_{i=1}^{\tilde{m}} L_{\mathcal{F}_A[i]}) \leq C_A. \tag{2}$$

3.2. Locality-aware Information Clustering

Recall that nodes report the information of their source files and available capacities to repository supernodes in phase two of CORP. We use IR to denote the information for file replication. The IR of overloaded node A is represented by a set of 6-tuple:
$IR = < L_{\mathcal{F}_A[i]}, S_{\mathcal{F}_A[i]}, V_{\mathcal{F}_A[i]}, U_{\mathcal{F}_A[i]}, ID_{\mathcal{F}_A[i]}, ip\_addr(A) >$,

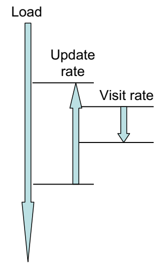


Figure 2. File ordering in source file selection (phase 1).

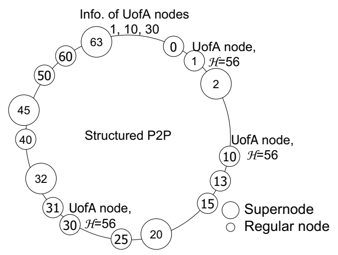

Info. of UofA nodes 1, 10, 30
63
0
1
UofA node, $\mathcal{H}$=56
2
60
50
45
40
Structured P2P
UofA node, $\mathcal{H}$=56
10
13
15
32
31
UofA node, $\mathcal{H}$=56
30
25
20
Supernode
Regular node

Figure 3. Information clustering (phase 2).

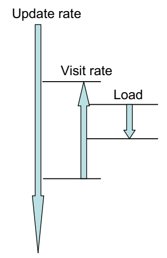


Figure 4. File ordering in replica arrangement (phase 3).

in which $ID_{\mathcal{F}_A[i]}$ denote the ID of file $\mathcal{F}_A[i]$ and $ip\_addr(A)$ denotes the IP address of node A. The IR of lightly-loaded node B is in a 2-tuple representation $IR =< \delta C_B, ip\_addr(B) >$, where $\delta C_B = C_B - L_B$ is node B's available capacity.

CORP realizes locality-aware file replication through locality-aware information clustering which groups the IR of physically close nodes together. It is developed by leveraging hash-based proximity clustering in our previous work [25] which deals with load balancing problem. Similar works of locality-awareness can also be found in [26, 24, 27]. However, there are few works tackling locality-aware file replication which reduces the overhead of both file replication and consistency maintenance.

We choose Chord structured P2P network [19] as an example to explain this clustering method. It can be applied to any existing structured P2P network such as Tapestry, CAN, Pastry, and Kademlia. CORP employs the techniques of landmark clustering and Hilbert curve [27] to calculate the *Hilbert number* of each node, denoted by $\mathcal{H}$. The Hilbert number indicates node physical closeness on the Internet. Two nodes with close Hilbert numbers are physically close to each other, and vice versa. Due to the space limit, for more information of the methods, please refer to [27].

In general, supernodes are nodes with high capacity and fast connections, and regular nodes are nodes with lower capacity and slower connections. For simplicity, we define a node with capacity greater than a predefined threshold as supernode; otherwise a regular node. The *logical distance* of node A and node B is calculated by $D(A, B) = |ID_B - ID_A|$. The clustering method assigns regular nodes to their logically closest supernodes. As a result, logically close nodes constitute a cluster with a supernode, which operates as a server to the regular nodes in the cluster.

Physically close nodes will report their information to the same supernode or logically close supernodes. A structured P2P network provides two main functions: `Put(ID,object)` and `Lookup(ID)` to store an object with an ID in its owner node, and to retrieve the object. In Chord, an object is assigned to the first node whose ID is equal to or follows the key in the ID space. If two objects have the same or close IDs, then they are stored in the same node or close nodes in the ID space. Because Hilbert number represents node physical closeness, if nodes report their IR with their Hilbert number as the key by put($\mathcal{H}, IR$), the IR of physically close nodes with the same or close Hilbert numbers will reach the same node or logically close nodes. The destination nodes further forward the IR to their supernodes. As a result, the IR of physically close nodes are pooled in the same supernode. Figure 3 shows an example of the information pooling in the supernode ring. In the example, physically close regular node *n*1, *n*10 and *n*30 periodically send their IR with their Hilbert number 56 as destination. The IR will first arrive at *n*60, and then is forwarded to *n*63.

We call the destination supernode of the reporting nodes as their *cluster server*, and call the reporting nodes as the cluster server's *clients*. Therefore, the physically close nodes connect to their cluster server virtually by put($\mathcal{H}, IR$), and they reach their cluster server by Chord function Lookup($\mathcal{H}$). Rather than building an extra supernode network that clusters physically close nodes, the clustering method only arranges regular nodes to connect to their logically close supernode and depends on the original P2P functions for identifying physically close nodes, which greatly reduces the extra overhead for extra structure maintenance.

In CORP file replication, nodes periodically report their IR to their cluster servers. Specifically, each overloaded node reports the IR of its source files and each lightly-loaded node reports the IR of its available capacity (i.e. δC) to its cluster server periodically. The cluster server then arranges the replica nodes for each source file. The communication and file replication between physically close nodes enhance the efficiency of not only file replication but also file consistency maintenance, because files or update messages are transferred along short physical distances. Some supernodes may be overloaded due to the load of file replication arrangement. The load balancing mechanism in [25] can be adopted to deal with the load imbalance between supernodes.

Before a supernode leaves, it finds its preceding and succeeding supernodes and moves its clients to them. The clients then connect to their new supernodes. Lazy-update is used to handle supernode failures. That is, if a supernode

Table 1. Lists of available capacity and source files.

Information in a cluster server	
Available capacity list	Source file list
$< \delta C_B, ip\_addr(B) >$	$< L_{\mathcal{F}_A[1]}, S_{\mathcal{F}_A[1]}, V_{\mathcal{F}_A[1]}, U_{\mathcal{F}_A[1]}, ID_{\mathcal{F}_A[1]}, ip\_addr(A) >$
$\ldots$	$\ldots$
$< \delta C_G, ip\_addr(G) >$	$< L_{\mathcal{F}_A[\tilde{m}]}, S_{\mathcal{F}_A[\tilde{m}]}, V_{\mathcal{F}_A[\tilde{m}]}, U_{\mathcal{F}_A[\tilde{m}]}, ID_{\mathcal{F}_A[\tilde{m}]}, ip\_addr(A) >$

fails or leaves without warning, its clients will not receive response from the supernode. The clients then connect to the new supernodes by probing their neighbors. For more information of the clustering method, please refer to [25].

3.3. File Replication Arrangement

This section presents how the source files are replicated in the lightly-loaded nodes after the IR of physically close nodes is marshalled in the cluster servers, with the objective of enhancing the efficiency of both file replication and consistency maintenance.

Upon receiving the IR from its clients, each cluster server stores the IR into a pair of lists of source files and available capacity correspondingly as shown in Table 1. It then arranges replica nodes for each source file. In order to reduce the number of replicas, the ideal situation is that each source file has one replica node. Thus, it is important to avoid fragmenting the available capacity of a node so that the capacity can be used for a source file with higher load. Therefore, the available capacity list is sorted in ascending order of δC; $max\delta C$ represents the maximum δC in the list. To locate a replica node for source file $\mathcal{F}_A[i]$, the cluster server checks the available capacity in the sorted list one by one in a top-down manner until $\delta C \geq L_{\mathcal{F}_A[i]}$. Hence, a source file is assigned to the most fit lightly-loaded node which has minimum available capacity left after the source file is replicated to it. This avoids fragmenting the available capacity, and makes full use of the existing available capacity.

As explained, a frequently-updated or infrequently-visited file should have less replicas. The factor of update rate has higher priority than visit rate since it directly affects the overhead of consistency maintenance. When a number of files have the same update rate and visit rate, the file with higher load should be resolved first in order to avoid capacity fragmentation and reduce the number of replicas. For instance, the available capacity list is $\{100, 200, 300, 400\}$ and the load list of source files is $\{300, 250, 200, 150\}$. If the list of source files is sorted in descending order, after assigning each file based on the above algorithm, the 100 available capacity is left. If the load list is ordered by ascending order, the load 300 has to be separated and replicated in three nodes. Thus, sorting the load list in ascending order leads to capacity fragment and more replicas.

Based on the analysis, as shown in Figure 4, CORP orders the IR pieces of source files in descending order of update rate. Files with the same update rate are ordered in ascending order of visit rate, and files with the same visit rate are further ordered in a descending order of their load values. CORP orders the IR pieces of available capacity in ascending order. File replication arrangement is executed between a pair of the two lists. In the algorithm, each IR piece in the source file list is fetched, and its replica node is searched in the available capacity list in a top-down manner until the available capacity is no less than the source file's load. This algorithm guarantees that frequently-updated files have a higher priority to be assigned to lightly-loaded nodes with higher available capacities, then infrequently-visited files, and finally higher load files. The algorithm reduces the number of the source files' replicas and the total update rate of the replicas, subsequently reducing the overhead of consistency maintenance.

A node records the replica nodes for each of its source files and the visit rate that each replica node is responsible for. Later on, if the node receives a file query when it is overloaded, it forwards the query to one of its replica nodes according to the visit rates they are responsible for. A replica node responsible for higher visit rate will receive more queries and vice versa.

A cluster server may not be able to resolve all source files for all of its clients. Recall that physically close nodes report their IR to logically close cluster severs. Therefore, the logical distances between a cluster server and its succeeding cluster servers or preceding cluster servers represent the physical distances between their clients. Based on this, a cluster server s can contact its succeeding cluster server and preceding cluster server, represented by *suc(s)* and *pre(s)*, to replicate its unsolved source files to their physically close nodes. It sends messages to its successor and predecessor [19]. The messages will be forwarded successively until arriving at supernodes. After file replication arrangement is performed between the supernodes' available capacity lists and the cluster server's source file list, if the source file list is still nonempty, *suc(suc(s))* and *pre((pre(s))* are attempted. This process is repeated until the cluster server's source file list becomes empty. Communication and file replication between physically close nodes reduce overhead.

A node periodically checks its load status. If it is lightly-loaded, it first removes some of its replicas before reporting its available capacity. The replicas of files with low visit rate, high update rate or low load should be removed firstly in order to reduce the overhead of consistency maintenance and increase replica utilization. An overloaded node includes the replicas in itself into its own files for source file selection, and can also select these replicas to be further replicated. Each node periodically reports its source files or available capacity based on its experienced actual load, and increases or decreases the number of replicas of its files adaptively based on file popularity and visit pattern. Correspondingly, each node notifies replica nodes to delete replicas or requests its cluster server for more replicas.

For example, in skewed lookups where many nodes query for a highly-popular file, CORP enables the file owner quickly locate physically close nodes with sufficient capacities to replicate the file in order to release its load. Later on, the file owner forwards the file queries to the replica nodes, which will further make replicas of the popular file when overloaded. Consequently, there will be more and more replicas for the highly-popular file until no node is overloaded due to the visits of the file. In the previous file replication methods, the file will be replicated to other nodes without the consideration of node available capacity, which may overload the replica nodes and exacerbate the situation of overloaded nodes. CORP selects source files and replicates them based on recent file visit rate and it is dynamically adaptive to lookup and visit pattern. From the view of the entire system, a file with increasing popularity will have more and more replicas and a file with decreasing popularity will have less and less replicas. It helps to guarantee the high utilization of replicas and reduce the overhead of consistency maintenance for unnecessary replicas.

4. Performance Evaluation

We designed and implemented a simulator for evaluating CORP based on Chord. There are two other classes of file replication methods: ID-based and location-based. We chose PAST [4] and LAR [10] in each class and compared the performance of CORP with them. Briefly, PAST replicates a file to a number of nodes whose IDs match most closely to the file's ID. LAR replicates a file to query initiators. To make them comparable, we did not create routing hints in lookup paths, and we updated every file once using broadcasting method in each protocol in consistency maintenance. The number of nodes in the system was set to 4096, and the number of files was set to 20480. The visit rate and update rate of each file was randomly chosen from [1,10]. We assumed bounded Pareto distribution for the capacity of nodes and the sizes of files. This distribution reflects real world where there are machines with capacities that vary by different orders of magnitude. The network topology was generated by GT-ITM [28]: "ts5k-large". It has 5 transit domains, 3 transit nodes per transit domain, 5 stub domains attached to each transit node, and 60 nodes in each stub domain on average. We define *node utilization (NU)* as the fraction of a node's capacity that is used: $NU_A = L_A/C_A$. *System utilization* is the ratio between the system's total load and the system's total capacity: $\sum_{i=1}^{n} L_i/\sum_{i=1}^{n} C_i$.

4.1. Capacity-aware File Replication

Figure 5 shows the total number of file replicas versus system utilization. We can observe that the number of replicas increases with the increase of system utilization. It is because higher system utilization leads to more overloaded nodes, resulting in more file replicas. We can also find that LAR and PAST generate dramatically more replicas than CORP. Recall that LAR and PAST do not consider node available capacity during file replication. They replicate files at the query initiators and logically close nodes respectively based on node load status. File replication without considering node available capacity will result in unnecessary replicas. In addition, the replica nodes may not have sufficient capacity for the replicas, leading to more overloaded nodes. In contrast, CORP proactively takes into account node available capacity during file replication. It not only avoids unnecessary file replication, but also avoids exacerbating the overloaded node problem by choosing nodes with sufficient available capacity as replica nodes. Thus, it outperforms LAR and PAST by reducing overloaded nodes and extra overhead for load balancing and file consistency maintenance.

4.2. Locality-aware File Replication

This test shows the effectiveness of CORP to achieve locality-aware file replication between physically close nodes. Figure 6 shows the cumulative distribution function (CDF) of total load of replicated files with system utilization approaching 1 in "ts5k-large". We can see that in "ts5k-large," CORP is able to replicate 95% of total load of replicated files, while LAR and PAST replicate about 30% within 10 hops. Almost all replications in CORP are within 15 hops, while LAR and PAST protocols replicate only 80% of the total file load within 15 hops. The results show that CORP replicates most files in short distances but LAR and PAST replicate most file in long distances. The more file replicated in the shorter distances, the less overhead incurred in both file replication and consistency maintenance. The results indicate that CORP performs superiorly compared to LAR and PAST with regards to the communication overhead in both file replication and consistency maintenance.

4.3. Lightweight File Consistency Maintenance

The cost of a message transmission is directly related to message size and physical distance of the message travelled; we use the product of these two factors of all file update messages to represent the consistency maintenance cost. It is assumed that the size of an update message is 1 unit. Figure 7 plots the file consistency maintenance cost of

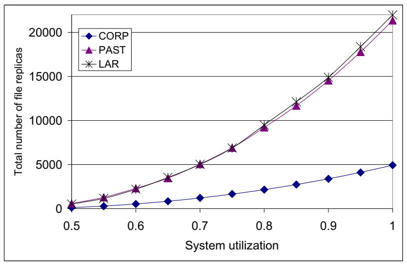
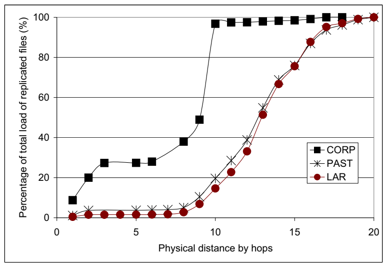
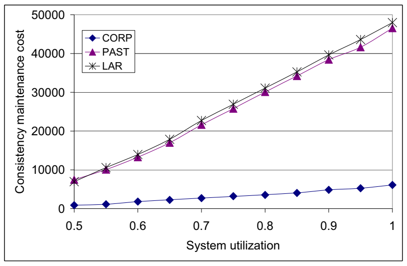


Figure 5. Total replicas.



(a) ts5k-large

Figure 6. CDF of total load of replicated files.



(a) ts5k-large

Figure 7. File consistency maintenance cost.

CORP, PAST and LAR in “ts5k-large”. From the figure, we can see that the cost increases as system load increases, and LAR and PAST incur considerably higher cost than CORP. There are two reasons for the observation. First, LAR and PAST replicate each file without the consideration of node available capacity, file popularity and visit rate, resulting in many unnecessary file replicas or replicas with high update rates. Taking into account of these factors, CORP generates less replicas and less replicas with high update rates. Second, because LAR and PAST neglect locality in file replication, they render significantly higher cost for file update since messages travel long physical distances. In contrast, CORP proactively considers locality in file replication, such that the update messages only travel between physically close nodes. Its features of fewer replicas, replicas with lower update rates, and shorter update message travel distance contribute to low-overhead and timely file consistency maintenance.

4.4. Effect of File Popularity and Update Rate

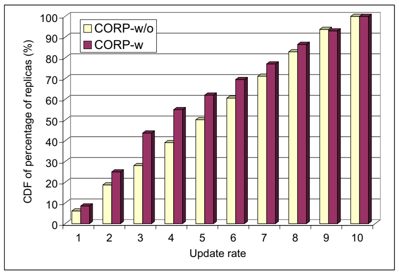
Recall that CORP reduces the replicas of frequently-updated or infrequently-visited files to decrease the overhead of file consistency maintenance. This experiment is to illustrate the effectiveness of CORP that considers file popularity and update rate. We use ”CORP-w” and ”CORP-w/o” to represent CORP protocol with and without file update and visit rates consideration respectively. Figure 8(a) illustrates the CDF of percentage of replicas versus update rate. We can see that CORP-w makes less replicas for frequently-updated files while makes more replicas than CORP-w/o for infrequently-updated files. Taking into account of node available capacity, CORP-w assigns high priorities to infrequently-updated files to be replicated into nodes with high available capacity to reduce the number of replicas, leading to reduction of consistency maintenance overhead. Figure 8(b) demonstrates the CDF of percentage of replicas versus visit rate. We can see that CORP-w renders less replicas for unpopular files compared with CORP-w/o. It is not necessary to produce a large number of replicas for an unpopular file, since it is visited infrequently and seldom generates load on a node. On the other hand, popular files should have more replicas to distribute its load among more nodes and increase replica utilization. From the figure, we can conclude that compared to CORP-w/o, CORP-w generates less replicas for unpopular files, and more replicas for popular files, thus reducing consistency maintenance cost, avoiding idle file replicas, and enhancing replica utilization.

4.5. Performance in a Dynamic Environment

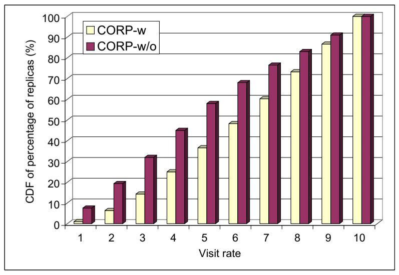
This experiment evaluates the efficiency of CORP in a dynamic environment with churn. In this experiment, we run each trial of the simulation for 20T simulated seconds, where T is a parameterized period set to 60 seconds. We averaged the simulation outcomes as the final results. The file join/departure rate was modelled as a Poisson process at rate of 0.4 as in [19]. That is, there is one file join and one file departure every 2.5 seconds. We ranged node interarrival/interdeparture rate from 0.1 to 0.5 with step size of 0.1. Figure 9 illustrates the average of NUs versus node interarrival/interdeparture rate. By comparing the results of LAR/PAST and CORP, we can make two observations. First, the 99th percentile NUs of CORP are kept no more than 1. This implies that CORP can achieve the file replication goal of reducing overloaded nodes in dynamism. Second, the 99th percentile NUs of LAR/PAST are higher than 1, and they increase as the node interarrival/interdeparture rate increases. Because LAR/PAST overlook the node available capacity factor, they are not able to control node utilization as in static situation. In conclusion, unlike LAR and PAST, CORP ensures a load balance condition by file replication even in a dynamic environment.

5. Conclusions

In P2P file sharing systems, in spite of the significant impact of file replication on the efficiency of consistency maintenance, the two issues have been typically addressed separately. Most previous file replication methods focus on hot spot elimination and querying efficiency but neglect



(a) Update rate



(b) Visit rate

Figure 8. CDF of percentage of replicas.

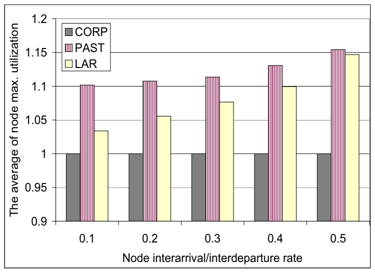


Figure 9. Performance in a dynamic environment.

the effect of file replication on the subsequent consistency maintenance. This paper presents a COoperative file replication Protocol (CORP) that not only achieves high efficiency in file replication but also proactively supports scalable, low-cost, and timely consistency maintenance. It observes a set of factors including file update rate, file popularity, node available capacity, file load and node locality that influence the efficiency of consistency maintenance. It takes into account these factors in file replication to reduce the number of replicas and the replicas of frequently-updated or infrequently-visited files, and to replicate files in physically close nodes. Moreover, it adaptively adjusts the number of replicas based on ever-changing file popularity and visit pattern. Thus, CORP significantly improves the efficiency of both file replication and consistency maintenance. Simulation results demonstrate the efficiency and effectiveness of CORP in comparison with other file replication protocols.

Acknowledgment

This research was supported in part by U.S. NSF grants CNS-0834592, CNS- 0832109 and CNS 0917056.

References

[1] Bittorrent. http://en.wikipedia.org/wiki/Bittorrent.

[2] P2P Traffic Is Booming, BitTorrent The Dominant Protocol. http://torrentfreak.com/p2p-traffic-still-booming-071128/.

[3] Final Tally: Olympics Web and P2P Numbers. http://newteevee.com/2008/08/28/final-tally-olympics-web-and-p2p-numbers/.

[4] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility. In *Proc. of SOSP*, 2001.

[5] F. Dabek, M. F. Kaashoek, D. Karger, and et al. Wide-area cooperative storage with CFS. In *Proc. of SOSP*, 2001.

[6] K. Huang and et al. LessLog: A Logless File Replication Algorithm for Peer-to-Peer Distributed Systems. In *Proc. of IPDPS*, 2004.

[7] T. Pitoura, N. Ntarmos, and P. Triantafillou. Replication, Load Balancing and Efficient Range Query Processing in DHTs. In *Proc. of EDBT*, 2006.

[8] V. Ramasubramanian and E. Sirer. Beehive: The Design and Implementation of a Next Generation Name Service for the Internet. In *Proc. of ACM SIGCOMM*, 2004.

[9] T. Stading and et al. Peer-to-peer Caching Schemes to Address Flash Crowds. In *Proc. of IPTPS*, 2002.

[10] V. Gopalakrishnan, B. Silaghi, and et al. Adaptive Replication in Peer-to-Peer Systems. In *Proc. of ICDCS*, 2004.

[11] M. Theimer and M. Jones. Overlook: Scalable Name Service on an Overlay Network. In *Proc. of ICDCS*, 2002.

[12] M. Roussopoulos and M. Baker. CUP: Controlled Update Propagation in Peer to Peer Networks. In *Proc. of the USENIX 2003 Annual Technical Conf.*, 2003.

[13] T. Hu, S. Ardon, and A. Sereviratne. Semantic-laden peer-to-peer service directory. In *Proc. of the 4th IEEE International Conference on P2P Computing (P2P04)*, 2004.

[14] Marcus Fontoura. Law-governed peer-to-peer auctions. In *Proc. of the eleventh international world wide web conference (WWW2002)*, pages 109–117, 2002.

[15] J. Zhu, J. Gong, W. Liu, T. Song, and J. Zhang. A collaborative virtual geographic environment based on P2P and Grid technologies. *Information Sciences: An International Journal archive*, (21), 2007.

[16] P2P Calendar Synchronizer. http://www.brothersoft.com/p2p-calendar-synchronizer-47655.html.

[17] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proceedings of the 21th ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[18] B. Knutsson, H. Liu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Proceeding of IEEE INFOCOM*, 2004.

[19] I. Stoica, R. Morris, D. Liben-Nowell, and et al. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet

Applications. *IEEE/ACM Transactions on Networking*, 1(1):17–32, 2003.

[20] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *Proc. of NSDI*, pages 99–112, 2004.

[21] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A Decentralized Peer-to-Peer Web Cache. In *Proc. of ACM Symposium on Principles of Distributed Computing*, 2002.

[22] H. Shen. IRM: Integrated File Replication and Consistency Maintenance in P2P Systems. In *Proc. of IEEE ICCCN*, 2008.

[23] H. Shen and Y. Zhu. Plover: Proactive Low-overhead File Replication in Structured P2P Systems. *Journal of Parallel and Distributed Computing (JPDC)*, 2009.

[24] Y. Zhu and Y. Hu. Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 16(4), 2005.

[25] H. Shen and C.-Z. Xu. Hash-based Proximity Clustering for Efficient Load Balancing in Heterogeneous DHT Networks. *Journal of Parallel and Distributed Computing (JPDC)*, 2008. An early version appeared in IPDPS, 2006.

[26] Z. Li, G. Xie, and Z. Li. Locality-Aware Consistency Maintenance for Heterogeneous P2P Systems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2008. Early version is appeared in Proc. of IPDPS'07.

[27] Z. Xu and et al. Turning Heterogeneity into an Advantage in Overlay Routing. In *Proc. of INFOCOM*, 2003.

[28] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proc. of INFOCOM*, 1996.