A Lightweight and Cooperative Multi-Factor Considered File Replication Method in Structured P2P Systems

Haiying Shen*, *Member, IEEE,* Guoxin Liu, *Student Member, IEEE*

Abstract—File replication is widely used in structured P2P systems in order to avoid hot spots in servers and enhance file availability. The number of replicas and replication distance affect the file replication cost. These two elements and the replica update frequency determined in the file replication stage also affect the cost of subsequent consistency maintenance. However, most existing file replication protocols focus on improving file lookup efficiency without considering its cost and its subsequent influence on consistency maintenance. This paper studies the problem about how a server chooses files to replicate and where to replicate files in order to achieve low cost in both file replication and consistency maintenance stages without compromising the effectiveness of file replication. This paper presents a lightweight and Cooperative multi-factOr considered file Replication Protocol (CORP) to achieve this goal. CORP simultaneously takes into account multiple factors including file popularity, update rate, node available capacity, file load, and node locality, aiming to minimize the number of replicas, update frequency, and replication distance. CORP also dynamically adjusts the number of replicas based on ever-changing file popularity and visit pattern. Extensive experimental results from simulation and PlanetLab real-world testbed 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. In addition, it exhibits high adaptiveness to skewed lookups and yields significant improvement in reducing overloaded nodes. Specifically, compared to the other replication protocols, CORP can reduce more than 71% of file replicas, 84% of overloaded nodes, 94% of consistency maintenance cost, and 72% of file replication and consistency maintenance latency.

Keywords: Peer-to-peer systems, File replication, Consistency maintenance, Proximity.

1 Introduction

Over the past years, peer-to-peer (P2P) file sharing systems [1–8] have attracted a great deal of attention in research. A popular (i.e., hot) file in a node can easily exhaust its capacity. File replication protocols [9–18] that replicate a hot file to other nodes can avoid such hot spots. Consistency maintenance that updates replicas when a file is changed is indispensable to file replication with non-static files. This is evidenced by numerous previously proposed file consistency maintenance protocols [17–24] for P2P systems. Consistency maintenance is needed in P2P systems (i.e., OceanStore [25] and Publius [26]) that permit users to modify their files. In addition, future P2P applications also need consistency support to deliver frequently-updated contents such as directory service [27], online auction [28], remote collaboration [29], shared calendar [30, 31], P2P web cache [32], and online games [33].

Replicating files and requiring that the replica nodes of a file be reliably informed of all updates to the file could be costly in a large-scale P2P network. The cost in the file replication is mainly determined by the number of replicas and replication distance (i.e., physical network distance) between file owners and replica nodes. These two elements and the replica update frequency also affect the cost of subsequent consistency maintenance. Therefore, a file replication method should proactively consider the three elements in order to reduce not only its own cost but also consistency maintenance cost.

However, there has been little file replication research devoted to tackling the dependency issue of consistency maintenance. Most previous methods in structured P2P networks determine the replica node based on node ID [9–12] or location [13–18]. 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 (including the requester) and a file provider. Both groups of methods focus on where to replicate files to enhance file availability and avoid hot spots to improve query efficiency, but neglect the cost of file replication and its impact on consistency maintenance cost. This work complements the previous works by studying a different problem in file replication: how a server chooses files to replicate and where to replicate files in order to achieve low cost in both file replication and consistency maintenance stages without compromising the effectiveness of file replication.

To address this problem, we propose a lightweight and Cooperative multi-factOr considered file Replication Protocol (CORP). CORP simultaneously takes into account multiple factors including file popularity, update rate, node available capacity, file load, and node locality, aiming to minimize the number of replicas, update frequency and replication distance. CORP is characterized by the following features.

- *The consideration of node available capacity and file load.* In order to reduce the number of replicas, CORP replicates higher-load files into nodes with sufficient capacity for the load. Replicating higher-load files releases more load off a server, and sufficient-capacity replica nodes avoid the need of multiple replicas, resulting in fewer total replicas of the server.
- *The consideration of file update frequency.* CORP makes more replicas for infrequently-updated and fewer replicas for frequently-updated files in order to reduce the cost of update propagation in consistency maintenance.
- *The consideration of file popularity.* CORP makes more replicas for popular files and fewer replicas for unpopular files to increase replica utilization.
- *The consideration of node locality.* CORP replicates file in physically close nodes to reduce cost in both file replication and consistency maintenance.

- * *Corresponding Author. Email: shenh@clemson.edu; Phone: (864) 656 5931; Fax: (864) 656 5910.*
- *The authors are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634. E-mail: {shenh, guoxinl}@clemson.edu*

Digital Object Indentifier 10.1109/TC.2012.104

- *The adaptiveness of file popularity and visit pattern.* CORP dynamically adjusts the number of replicas to deal with ever-changing file popularity and visit pattern.

In this paper, we consider a situation in which the visit and update rates of files do not vary greatly with very high frequency. Also, a node replicates its files only when it is overloaded and the main goal of replication is to release the load of overloaded nodes. There are many techniques that can be used to increase file availability and file retrieval efficiency. A lightly-loaded node can also replicate its highly-popular files to reduce the lookup hops [34]. Replicating a file or file location hint to certain locations such as frequent requesters [13, 15], nodes near frequent requesters, or along query path [9, 10] can enhance file query efficiency. Splitting a large file into small pieces and replicating parts of the file [10] can increase the service capacity of a large file rapidly. CORP can employ these techniques to further improve its performance. These techniques are orthogonal to our study.

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 and Section 5 present the performance of CORP in static situation as well as dynamic situation in comparison with representative file replication protocols. Section 6 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 [9] is a P2P-based file system for large-scale persistent storage service. In PAST, each file is replicated on a number of nodes whose IDs match most closely to the file's ID. CFS [10] is a P2P read-only storage system that 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 a distributed hash function to replicate each block on nodes immediately after the block's successor on the Chord ring [2] in order to increase file availability. LessLog [11] constructs a lookup tree based on node IDs to determine the locations of replica nodes. HotRoD [12] is a structured P2P based architecture with a replication scheme, in which a group of successive peers on a ring is "hot" when at least one of these peers is hot. "Hot" arcs of peers are replicated and rotated over the ID space.

Another group of replication methods replicate files based on node location by choosing replica nodes along a file lookup path between a client and a server (including the clients). Beehive [13, 34] replicates an object at nodes i hops prior to the server in the lookup path. It decides a file's replication degree based on its popularity in order to achieve constant lookup performance. Stading *et al.* [14] proposed Backslash, 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 [15] specifies at what 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 [16] places a replica of a file on a node with most incoming lookup requests for fast replica location. EAD [35] replicates a file in query forwarding nodes that frequently forwards queries of the file. CUP [17] and DUP [18] cache metadata along the lookup path in a structured P2P system. OceanStore [25] aims to build a global persistent storage utility. It uses agents to collect and analyze client-access information for determining the location of replicated nodes. Its files are replicated on multiple servers for security concern without restricting the placement of replicas. Rubenstein and Sahu [36] also discussed the scalability achieved by the fact that user requests create additional replicas, which improves system performance. Squirrel [37] is a distributed P2P web cache to facilitate mutual sharing of web objects among client nodes. It enables nodes to export their local caches to other nodes in the corporate network, thus synthesizing a large shared virtual web cache.

However, few works consider the dependency of replica consistency maintenance on file replication and try to optimize consistency maintenance in the file replication stage. Our previous work [38] addressed this problem by removing under-utilized replicas to reduce the number of replicas, but did not take into account the previously mentioned factors in file replication to optimize consistency maintenance. CORP is distinguished from Plover [39] by taking into account file update rate and visit rate to reduce the overhead of consistency maintenance and increase replica utilization.

There are other works on file replication in structured P2P networks that are focused on other aspects instead of replica node location. Zhou *et al.* [40] proposed an on-line pointer replication algorithm which yields low worst case query latency. To reduce the replication cost, Liu *et al.* [41] presented a detailed description of a replication strategy based on file-partition. Lin *et al.* [42] formulated file replication problem as an optimization problem, and proposed several heuristic algorithms. Ni *et al.* [43] proposed a model to design file replication schemes for P2P file sharing systems. The model introduces expected costs for serving user file requests, which are computed from node up/down statistics. Based on the model, the authors develop several methods to determine the sets of nodes to store copies of the files in order to optimize certain performance metrics.

3 Lightweight and Cooperative File Replication

3.1 An Overview of CORP

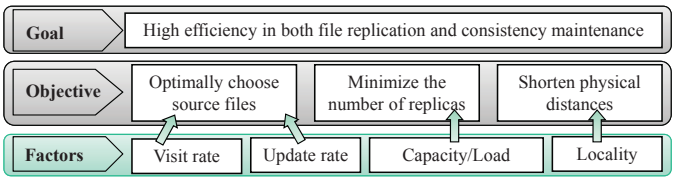


Fig. 1. The goals of CORP and its considering factors.

Figure 1 shows the goal of CORP and the factors it considers during the file replication to achieve the goal. To optimally choose source files to replicate (briefly called *source files*), CORP considers *file popularity and file update rate*. An unpopular file does not necessarily have many replicas since the replicas would have low probability to be visited, and a highly-popular file should have more replicas. Different files have different updates. For instance, files for auction applications may change frequently while video or mp3 files hardly change. Frequently-updated files should have fewer replicas in order to reduce update message propagation overhead.

To minimize the number of replicas, CORP considers *node available capacity and file load*. More replicas lead to higher overhead for consistency maintenance. By considering node available capacity and file load, CORP brings three benefits. First, unlike the previous methods to replicate a file to many nodes, each of which is responsible for partial visit load, CORP tries to replicate a file to a single node capable of handling the file's visits. Second, when a node chooses files to replicate, it selects the files with higher load, hence releasing more load and reducing replicas. Third, CORP avoids overloading replica nodes by considering node available capacities.

CORP also considers *locality* by discovering physically close nodes to a file's owner for file replication. Short physical distance between a file owner and its replica nodes helps to reduce the overhead for file replication and subsequent consistency maintenance by update propagation. Moreover, it also helps increase query processing efficiency since a file owner can forward a file query to its replica node efficiently. We present the three steps in CORP and their challenges in below.

(1) When an overloaded node chooses source files to replicate in order to release its load due to file visit, 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 3.2)
(2) When overloaded nodes and lightly-loaded nodes periodically 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 without relying on an additional structure? (Section 3.4)
(3) When repository nodes arrange file replication between overloaded nodes and lightly-loaded nodes, how to take into account the aforementioned multiple factors to minimize the number of replicas? (Section 3.5)

3.2 Generating Information for File Replication

As the works in [44–48], we assume that there is only one bottleneck resource for optimization although a node's capacity includes its storage space, bandwidth, and CPU. The capacity metric in practice should be a function of these factors [3, 15]. Most P2P systems like KaZaA and Gnutella allow the nodes to set the maximum upload and download bandwidth. We also assume that each node has a capacity it is willing to contribute to the system, which is in a percentage of the node's actual capacity. For simplicity, we represent a node's capacity (denoted by C) by the number of bits it can transfer in responding file queries during a time unit, say one second. Similarly, a node's load (denoted by L) is measured by the number of bits it needs to transfer in response to file queries.

A file's visit rate can be measured by the number of visits during a time unit. To truly reflect a file's popularity by its visit rate, CORP considers the file's current and past visit rates, introduces a weighted average visit rate of a file as below:

$$V_{new} = \beta V_{old} + (1-\beta)V_{current} \ (0 \le \beta \le 1),$$

where the weight β is a discount factor serving as the fading mechanism.

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: $L_A = \sum_{i=1}^{m} L_{\mathbb{F}_A[i]}$, where $L_{\mathbb{F}_A[i]}$ denotes the load of file $\mathbb{F}_A[i]$ in node A. The load of a file on a node is determined by the file's size and visit rate (i.e., popularity) on the node. That is, $L_{\mathbb{F}_A[i]} = S_{\mathbb{F}_A[i]} \times V_{\mathbb{F}_A[i]}$, where $V_{\mathbb{F}_A[i]}$ and $S_{\mathbb{F}_A[i]}$ respectively denote the visit rate and size of file $\mathbb{F}_A[i]$ in node A. We define the update rate of file i 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 the visit rate. We assume the update message size is the same for each file. We leave a more sophisticated strategy considering different update message sizes as our future work.

Each node A periodically measures $V_{\mathbb{F}_A[i]}$ 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 i. When node A is overloaded, in order to release excess load $\Delta L_A = L_A - C_A$, it needs to choose a number of source files to replicate. Later on, when node A receives a file request, if it is overloaded, it forwards the request to one of the file's replica nodes in a round-robin manner, so the load can be distributed among the replica nodes. We represent the source files of node A by $\mathcal{F}_A = \{\mathcal{F}_A[1], \mathcal{F}_A[2], \ldots, \mathcal{F}_A[\tilde{m}]\}$ ($\mathcal{F} \in \mathbb{F}, 1 \le \tilde{m} \le m$) and represent their corresponding loads by $\{L_{\mathcal{F}_A[1]}, L_{\mathcal{F}_A[2]}, ..., L_{\mathcal{F}_A[\tilde{m}]}\}$. Thus,

$$\sum_{i=1}^{\tilde{m}} L_{\mathcal{F}_A[i]} \ge \Delta L_A. \quad (1)$$

Suppose the number of replicas of file $\mathcal{F}_A[i]$ is n_i. We use $R_{\mathcal{F}_A[i],j} (0 < j \le n_i)$ to denote the j^{th} replica node of file $\mathcal{F}_A[i]$, and use $D(A, R_{\mathcal{F}_A[i],j})$ to denote the distance between server A and replica node $R_{\mathcal{F}_A[i],j}$.

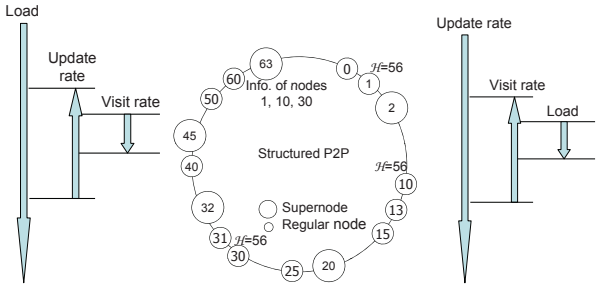
Recall that the objectives of CORP include:
(1) minimize the number of replicas, i.e., $\sum_{i=1}^{\tilde{m}} n_i$;
(2) minimize update rate of replicas; i.e., $\sum_{i=1}^{\tilde{m}} n_i \cdot U_{\mathcal{F}_A[i]}$;
(3) maximize the utilization of replicas, i.e., $\sum_{i=1}^{\tilde{m}} (V_{\mathcal{F}_A[i]})$;
(4) minimize the distances between the server and replica nodes, i.e., $\sum_{i=1}^{\tilde{m}} \sum_{j=1}^{n_i} D(A, R_{\mathcal{F}_A[i],j})$.

According to Formula (1) and objective (1), we need to minimize $\tilde{m}$ and n_i, and hence choose high-load files for replication. According to object (2), we need to minimize $U_{\mathcal{F}_A[i]}$ and n_i. Frequently-updated files should have fewer replicas in order to reduce consistency maintenance overhead. According to objective (3), we need to maximize $V_{\mathcal{F}_A[i]}$ and minimize n_i. Replicating popular files can increase the replica utilization. Based on this, infrequently-visited files should have less or no replicas since they will have low probability to be visited, thus decreasing idle replicas and file consistency maintenance overhead. Also, frequently-visited files should have more replicas in order to improve average lookup efficiency in the system. In addition, it reduces the overhead of file replication due to smaller replicating file size because a higher frequently-visit rate means small size given a certain amount of load. According to objective (4), we need to minimize n_i and replicate files to physically close nodes.

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

$$\text{minimizes } \tilde{m}; \text{ minimizes } \sum_{i=1}^{\tilde{m}} U_{\mathcal{F}_A[i]}; \text{ maximizes } \sum_{i=1}^{\tilde{m}} V_{\mathcal{F}_A[i]} \text{ subject to } (\sum_{i=1}^{\tilde{m}} L_{\mathcal{F}_A[i]}) \le L_A - C_A. \quad (2)$$

We give the update rate a higher priority than the visit rate because the update rate directly affects the consistency maintenance cost. Accordingly, to choose source files to replicate, a node orders its files as shown in Figure 2(a). Specifically, the files 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.



(a) Source file selection (b) Info. clustering (c) Replication arrangement

Fig. 2. Three stages in CORP.

The source file selection process has two 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 load of the selected source files is no less than the load to release.

The next step reduces the total update rate of the source files in the selected files $\mathcal{F}_A$, thus reducing consistency maintenance overhead. It also increases the total visit rate of the source files in $\mathcal{F}_A$, thus increasing the utilization of file replicas and reducing file replication cost. In this step, the remaining 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 swaped with file $\mathbb{F}_A[\tilde{m}+1]$. Otherwise, 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 swaped with file $\mathbb{F}_A[\tilde{m}+1]$. Algorithm 1 shows the pseudocode of source file selection executed by node A.

Algorithm 1: Source file selection executed by node A.

```
SelectSourceFile (F_A, ΔL_A) // F_A is the list of all files {
  Order F_A by load in descending order, then update rate in
  ascending order, and then visit rate in descending order
  //find the initial F_A in F that satisfy Sum_{L_{F_A}} ≥ ΔL_A
  while (Sum_{L_{F_A}} < ΔL_A) do {
    Add F_A[i] to F_A
    Sum_{L_{F_A}} += L_{F_A[i++]} }
  while i < F_A.length do // find replacements in F_A {
    can_replace = false //flag: file F_A[i] can replace a file in F_A
    for (j = 0; j < F_A.length; ++j) do {
      if (Sum_{L_{F_A}} − L_{F_A[j]} + L_{F_A[i]}) ≥ ΔL_A then {
        can_replace = true
        if U_{F_A[i]} < U_{F_A[j]} then {
          //replace F_A[j] because F_A[i] has smaller update rate
          swap(F_A[j] , F_A[i])}
        else if U_{F_A[i]} == U_{F_A[j]} && V_{F_A[i]} > V_{F_A[j]} then {
          // replace F_A[j] because F_A[i] has larger visit rate
          swap(F_A[j], F_A[i])}}}
    i++ //get the next file in F_A[i]
    if !can_replace then
      break //stop checking due to small load of the rest of files}
  return F_A}
```

3.3 Integrated Consideration of Update and Visit Rates

We also propose a method to integrate file update rate and visit rate in source file selection to give both factors the same priority in consideration. We define a metric called replication priority (R) measured by V/U of a file. Node A sorts its files in descending order of their load, and the files with the same load are ordered in descending order of their replication priorities. Node A executes the same first step. In the second step, the rest of the 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 $R_{\mathbb{F}_A[\tilde{m}+1]} > R_{\mathcal{F}_A[k]}$ $(1 \leq k \leq \tilde{m})$ 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 $\mathbb{F}_A[\tilde{m}+1]$. This step aims to maximize the sum of Rs of the selected files in order to increase the replica utilization, reduce the cost of file replication and consistency maintenance.

3.4 Locality-aware Information Clustering

Nodes periodically report the information for file replication (IR) to repository supernodes. The IR of overloaded node A is for source files, which is represented by:

$$IR =< L_{\mathcal{F}_A[i]}, S_{\mathcal{F}_A[i]}, V_{\mathcal{F}_A[i]}, U_{\mathcal{F}_A[i]}, ID_{\mathcal{F}_A[i]}, IP(A) >,$$

in which $ID_{\mathcal{F}_A[i]}$ denotes the ID of file $\mathcal{F}_A[i]$ and $IP(A)$ denotes the IP address of node A. The IR of lightly-loaded node B is for available capacity, which is in a representation: $IR =< \delta C_B, IP(B) >$, 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 collects the IR of physically close nodes together. It is developed by leveraging hash-based proximity clustering in our previous work [44]. Similar works of locality-awareness can also be found in [21, 47, 49]. These works did not deal with file replication. We choose the Chord structured P2P network [2] as an example to explain this clustering method. It can be applied to any other existing structured P2P networks.

The hash-based proximity clustering method [44] represents the physical distances between nodes by indices, called Hilbert numbers. Briefly, each node measures its physical distances to the pre-determined m landmarks, and uses the vector of distances $< d_1, d_2, \ldots, d_m >$ as its coordinate. Then, the node uses the space-filling curves [50] to transform the coordinate to a Hilbert number, which indicates node physical closeness on the Internet. Two nodes with closer Hilbert numbers are physically closer to each other and vice versa.

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 a supernode; otherwise it is 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. A structured P2P network provides two main functions, `Put(ID,object)` and `Lookup(ID)` [51], to store an object with an ID in its owner node and to retrieve the object. In Chord, the 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, the objects are stored in the same node or close nodes in ID space. Therefore, if nodes report their information for file replication with their Hilbert number, $\mathcal{H}$, as the key by `put(`$\mathcal{H}, IR$`)`, the IR of nodes with the same or close $\mathcal{H}$ will reach the same node or logically close nodes because the Hilbert number represents node physical closeness. The nodes further forward the IR

to their supernodes. As a result, the IR of physically close nodes are pooled in the same supernode. Figure 2(b) shows an example of the information pooling in the supernode ring. The links between regular nodes and their supernode are omitted for clearness. Physically close regular nodes *n*1, *n*10 and *n*30 periodically send their IR with their Hilbert number 56 as the 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 nodes as the cluster server's *clients*.

Before a supernode leaves, it finds its preceding and succeeding supernodes in the supernode ring and moves its clients to them. The clients then connect to their new supernodes. The supernode ring use lazy-update to handle supernode failures. That is, if a supernode leaves or fails without warning, its clients will not receive a response from the supernode. The clients then connect to new supernodes. Proposition 3.1 sheds insight into the subtleties of the clustering method.

Proposition 3.1: For any set of N_s supernodes and N_r regular nodes, with high probability:
1. Each supernode is responsible for at most $(1+\epsilon)N_r/N_s$ regular nodes, where ϵ equals to $O(\log N_s)$.
2. When an $(N+1)^{st}$ supernode joins in or leaves the network, responsibility for N_r/N_s regular nodes changes hands (and only to or from the joining or leaving supernode).

Proof: We can regard supernodes and regular nodes in the supernode ring as nodes and keys in Chord ring. Theorem 1 in [2] proved that for N nodes and K keys, each node is responsible for at most $(1+\epsilon)K/N$ keys where $\epsilon = \log N$, and a node join and departure leads to responsibility of $O(K/N)$ keys changing hands. □

In CORP, 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 (Section 3.5). Some supernodes may be overloaded due to the load of file replication arrangement. The load balancing mechanisms in our previous work [44] can be adopted to deal with the load imbalance between supernodes.

3.5 File Replication Arrangement

Upon receiving the IR from its clients, each cluster server stores the IR into a pair of source file list and available capacity list. It then arranges replica nodes for each source file. It is important to avoid the fragmentation of the available capacity of a node so that the intact capacity can be for high-load source files. Also, as explained, for efficient file consistency maintenance, a frequently-updated or infrequently-visited file should have fewer replicas. As shown in Figure 2(c), 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. For the integrated consideration of update rate and visit rate, CORP orders IR pieces of source files in ascending order of their replication priorities, and files with the same replication priority are further ordered in a descending order of their load values. CORP organizes the IR pieces of available capacity in a balanced binary search tree based on δC. $max\delta C$ represents the maximum δC in the list. Then, each IR piece in the source file list is fetched, and its replica node is searched in the tree to locate the item that has the minimum available capacity no less than the load of the source file (i.e., $\min \delta C \geq L_{\mathcal{F}_A[i]}$). Thus, a source file is assigned to the most fit lightly-loaded node which has minimum free capacity left after the source file is replicated to it.

This arrangement facilitates to reduce the number of file replicas while releasing the excess load of overloaded nodes. Algorithm 2 shows the pseudocode of the file replication arrangement. This algorithm guarantees that frequently-updated files have the highest priority to be reassigned to a lightly-loaded node with higher available capacity, infrequently-visited files are given priority over higher-load files for the assignment, and higher load files finally. It reduces the number of these files' replicas and their total update, and subsequently reduces consistency maintenance overhead.

Algorithm 2: A cluster server performs file replication arrangement between a pair source file list (SFL) and available capacity list (ACL).

ReplicationArrange (SFL, ACL) *// $\mathbb{F}_A$ is the list of all files* {
for each IR item IR_i in the source file list **do**
//assume it is $< L_{\mathcal{F}_A[k]}, S_{\mathcal{F}_A[k]}, V_{\mathcal{F}_A[k]}, U_{\mathcal{F}_A[k]}, ID_{\mathcal{F}_A[k]}, IP(A) >$
  Search $L_{\mathcal{F}_A[k]}$ in the balanced binary tree of ACL to find the item that has $\min \delta C_B$ with $L_{\mathcal{F}_A[k]} \leq \delta C_B$ *//assume it is* $< \delta C_B, IP(B) >$
  if successfully find the item **then** {
  //node B can afford file $\mathcal{F}_A[k]$'s load totally
    File $\mathcal{F}_A[k]$ is replicated from node A to node B
    Remove the IR item of $\mathcal{F}_A[k]$ from source file list
    if $\delta C_B - L_{\mathcal{F}_A[k]} > 0$ **then**
      Insert $<(\delta C_B - L_{\mathcal{F}_A[k]}), IP(B)>$ back to the tree }
  else {
  //assume the node with $\max \delta C_B$ in ACL is node B and it can afford file $\mathcal{F}_A[k]$'s load partially
    File $\mathcal{F}_A[k]$ is replicated from node A to B
    $b = \lfloor \delta C_B / S_{\mathcal{F}_A[k]} \rfloor$
    Change $V_{\mathcal{F}_A[k]}$ in IR_i to $V_{\mathcal{F}_A[k]} - b$
    Change $L_{\mathcal{F}_A[k]}$ in IR_i to $S_{\mathcal{F}_A[k]} \times V_{\mathcal{F}_A[k]}$
    Insert item IR_i back to the source file list
    if $\delta C_B - (S_{\mathcal{F}_A[k]} \times b) > 0$ **then**
      Insert $<(\delta C_B - (S_{\mathcal{F}_A[k]} \times b)), IP(B)>$
      back to the tree }
}

If the lightly-loaded node with $max\delta C$ does not have enough capacity for a file's total load, it can be responsible for partial load based on visit rate, and leaves the rest of load to other lightly-loaded nodes. A node records the replica nodes for each of its source files and the visit rate the replica node is responsible for. Later on, if it receives a file query when it is overloaded, it forwards the query to the replica nodes based on their responsible visit rates. A replica node responsible for higher visit rate will receive proportionally more queries.

Let n denote the total number of nodes in a supernode's cluster, c_1 denote the percent of overloaded nodes in the cluster, and m denote the average number of reported source files of each overloaded node. A supernode needs to sort all the reported files by load, which has $O(c_1 nm \log(c_1 nm))$ time complexity, and builds the available capacity items of lightly overloaded nodes into a balanced binary search tree, which has $O((1-c_1)n \log((1-c_1)n))$ time complexity. The value of m is small and can be regarded as a constant. Then, the total complexity of first two steps is $O(n \log n)$. Let c_2 denote the average number of replicas of a source file. The file replication arrangement has $O(c_2 nm \log((1-c_1)n))$ time complexity. Regarding c_2 as a constant, the total time complexity of replication arrangement in a super node is $O(n \log n)$.

A cluster server may not be able to resolve all source files

within its cluster. Recall that physically close nodes report their information to logically close cluster servers. Therefore, the distances between neighboring cluster servers represent the physical distances between their virtual clients. For locality-aware probing, a cluster server s probes its logically nearby cluster servers in sequence by probing its successors or predecessors. It contacts its succeeding cluster server and preceding cluster servers represented by *suc(s)* and *pre(s)*. After file replication arrangement operation between their 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 improve the efficiency.

3.6 Adaptive File Replication

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

In skewed lookups, many nodes query for a highly-popular file. CORP enables the file owner to quickly locate physically close nodes with sufficient capacity to replicate the file, and forward the file queries to the replica nodes later on. In traditional file replication methods, the file will be replicated to other nodes without the consideration of available node capacity, which may overload the replica nodes, exacerbating the situation of overloaded nodes. CORP selects source files and replicates files based on their recent file visit rate, thus it is adaptive to time-varying file visit rates.

3.7 Replication for Locality-aware Replica Retrieval

Replicating files on the nodes physically close to the file owner generates low cost for file replication and consistency maintenance since the files and updates are transmitted among physically close nodes. A file normally has a larger size than the update message. If we can replicate a file in or physically near the file requester that frequently requests for this file, it can greatly reduce the transmission cost for file retrieval. Thus, we propose a replication algorithm for locality-aware replica retrieval to further enhance CORP.

When a node requests for a file, it sends out a request along with its Hilbert number to indicate its physical location. CORP pre-defines a threshold, T, for a node's visit rate on a file. T can be the average node visit rate on a file in the system. If the visit rate of a node on a file is larger than T, this file will be replicated in this node or a node physically close to this node. As a result, for a file with a large visit rate from individual nodes, the file's access load will be distributed to individual frequent requesters based on their visit rates, and the file owner is responsible for the accumulated load due to file visits from other nodes in the system.

After a node selects its files to replicate, it compares the visit rate of each individual node on each selected file with T. For example, a selected file i has requesters B, C and D that have visit rates on file i larger than T. Then, node A generates an additional three IR marked by $*$ (IR^*) for file i to report to the cluster servers of nodes B, C and D, respectively. Finally, node A reports IR_A for the remaining visit rate from other nodes denoted by

$$V_{(\mathcal{F}_A[i],r)} = V_{\mathcal{F}_A[i]} - V_{(\mathcal{F}_A[i],B)} - V_{(\mathcal{F}_A[i],C)} - V_{(\mathcal{F}_A[i],D)}$$

by executing put $(\mathcal{H}_A, IR_A)$.

Algorithm 3: Replication for locality-aware replica retrieval.

```
//node A reports IR of selected files to replicate
for each selected file F_A[i] do
  for each node K that has visited file F_A[i] do
    if V_(F_A[i],K) ≥ T then {
    //node K frequently requests for file F_A[i]
      Generate IR*_K
      Send put(H_K, IR_K) }
Generate IR_A
Send put(H_A, IR_A)
//A cluster server executes file replication arrangement
if receive an IR*_K then
  if has an IR_K in the available capacity list then
    if δC_K ≥ S_F_A[i] in IR*_K then {
      Assign file in IR*_K to node K
      δC_K −= S_F_A[i]
    Remove IR*_K }
Executes file replication arrangement
//File request redirection
if receive a request for a file from node B when overloaded then
  if there exists a replica node C having the same H as B then
    Notify node B to retrieve the file from node C
  else
    if there exists a replica node(s) for the file then
      Notify the requester to retrieve the file from a replica node
    else
      Queue the request
```

When B's cluster server arranges file replication, if it has IR from node B in the available capacity list and $\delta C_B > S_{\mathcal{F}_A[i]}$, the cluster server assigns the load in IR^*_B to node B. Then, B can retrieve its frequently visited file from itself. This file replication consumes B's capacity that equals to the file size, since there is no file transmission in retrieving the file. If the cluster server does not have IR from B in the available capacity list, it arranges the file replication as described previously. As a result, this file is replicated to a node physically close to B, say node E. When B asks the file server for this file, the server will notify it to retrieve the file from node E. Hence, the file transmission is between physically close nodes. Algorithm 3 shows the pseudocode for the replication for locality-aware replica retrieval.

4 Simulation 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 [9] and LAR [15] in each class and compared the performance of CORP with them in terms of capacity-awareness, locality-awareness, and file consistency maintenance cost. Briefly, PAST replicates a file to a number of nodes whose IDs match most closely to the file's ID. When a node is overloaded by file queries, it chooses its higher-load files to replicate to its neighbors until it is lightly loaded. The number of replicas per file was set to 5 as used in [9]. LAR replicates a file to query initiators. When a node is overloaded by file queries, it orders the query initiators according to the load

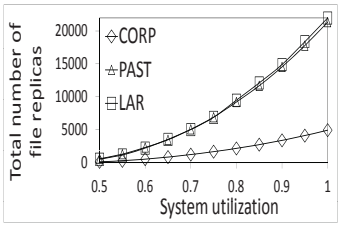


Fig. 3. Number of replicas.

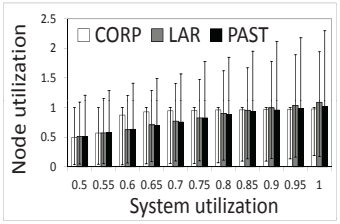
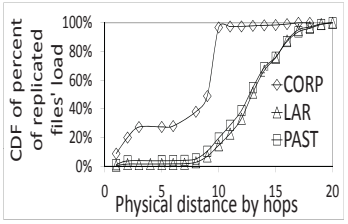
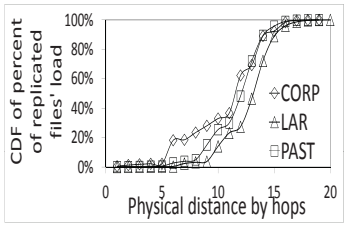


Fig. 4. Node utilization.

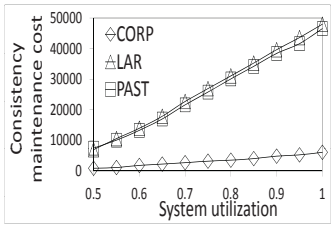


(a) ts5k-large

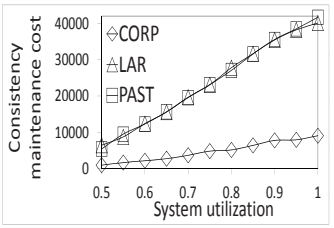


(b) ts5k-small

Fig. 5. CDF of total load of replicated files.



(a) ts5k-large



(b) ts5k-small

Fig. 6. File consistency maintenance cost.

generated by them, and replicates higher-load files to them until being lightly loaded. To make them comparable, we did not create routing hints in lookup paths, and an update message is directly sent from the file owner to the replica nodes 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 is randomly chosen from [1,10]. We define *node utilization* of node A, NU_A, as the fraction of its capacity that is used: $NU_A = L_A/C_A$. *System utilization* (SU) is the ratio between the system's total load and the system's total target capacity, which equals to $\sum_i L_i / \sum_i C_i$. We assumed bounded Pareto distribution for the capacity of nodes and the sizes of files. This distribution reflects the real world where there are machines with capacities that vary by different orders of magnitude. For node capacity, the upper and lower bounds were set to 25×10^4 and 25×10^3 respectively, with a shape of 2. For file size, we set the upper and lower bounds to $\bar{L} \times 10$ and $\bar{L}$ respectively with a shape of 2, and $\bar{L}$ is the average load per node. All files were visited based on their visit rate with randomly chosen requesters, and the experiment stopped when all overloaded nodes release their excess loads by replicating files.

The network topologies were generated by GT-ITM [52]: "ts5k-large" and "ts5k-small". "ts5k-large" represents a situation in which the network consists of nodes from several big stub domains. 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. "ts5k-small" represents a situation in which the network consists of nodes scattered in the entire Internet and there are only a few nodes from the same edge network. It has 120 transit domains, 5 transit nodes per transit domain, 4 stub domains attached to each transit node, and 2 nodes in each stub domain on average. To account for the fact that interdomain routes have higher latency, each interdomain and intradomain hop count as 3 and 1 hop of units of latency respectively.

4.1 Capacity-aware File Replication

Figure 3 shows the total number of file replicas versus SU. We observe that the number of replicas increases with the increase of SU. This is because higher SU leads to more overloaded nodes, resulting in more file replicas. We 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, which results in unnecessary file replication. Although PAST uses load balancing afterwards, it will generate extra overhead. In addition, the replica nodes may not have sufficient capacity for the replicas, leading to more overloaded nodes. Thus, the neglect of node available capacity in file replication leads to more replicas, more overloaded nodes, and extra overhead for load balancing and file consistency maintenance. In contrast, CORP proactively takes into account node available capacity during file replication. It not only avoids unnecessary file replication but also avoids exacerbating overloaded node by choosing nodes with sufficient available capacity as replica nodes. Thus, it outperforms LAR and PAST by controlling the overloaded nodes and extra overhead for load balancing and file consistency maintenance.

We measured the maximum NUs of all nodes after all overloaded nodes replicate their files in the first round. Figure 4 plots the median, 1st and 99th percentiles of node NUs versus SU in different methods. It shows that the 99th NUs of LAR and PAST increase with SU due to the same reason as in Figure 3. The median NUs of CORP are sometimes higher than LAR and PAST, but it constrains the 99th NUs within 1, while those of LAR and PAST are higher than 1. The results imply that LAR and PAST incur much more overloaded nodes, while CORP can keep all nodes lightly-loaded by balanced load distribution. The figure also demonstrates that the 99th NU of PAST is higher than that of LAR. This is because PAST replicates files in a file owner's logically close node. Relying on the nodes within a small range around the heavy file owner node will make these replica nodes overloaded. LAR only replicates files in the requesters that are scattered in the network.

4.2 Locality-aware File Replication

This test shows the effectiveness of CORP to achieve locality-aware file replication between physically close nodes. Figures 5(a) and (b) show the cumulative distribution function (CDF) of the total load of replicated files when the SU approaches 1 in "ts5k-large" and "ts5k-small", respectively. We can see that in "ts5k-large," CORP is able to replicate 95% of the 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 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 files in long distances. From Figure 5(b), we can make the same observations although the performance difference between the protocols is not so significant. The results show the superior performance of CORP compared to LAR and PAST with regards to locality-aware file replication.

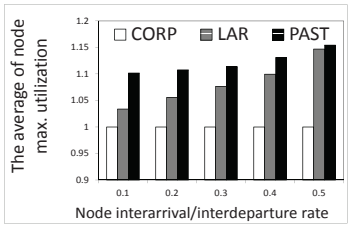
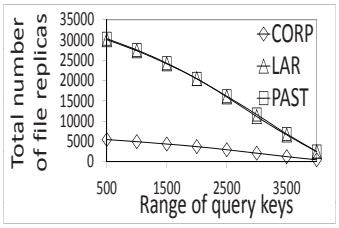
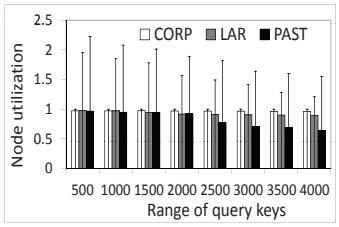


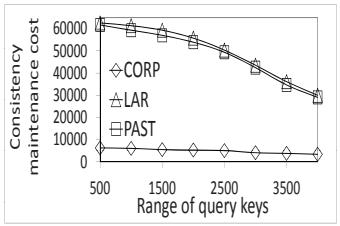
Fig. 7. Performance in churn.



(a) Number of replicas



(b) Node utilization



(c) Consistency maintenance cost

Fig. 8. Performance of file replication protocols in skewed lookups.

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. Figures 6(a) and (b) plot the file consistency maintenance cost of CORP, PAST, and LAR in "ts5k-large" and "ts5k-small", respectively. We see that the cost increases as system load increases, and LAR and PAST incur considerably higher cost than CORP. There are three 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 for files with high update rates. Taking these factors into account, CORP generates fewer replicas for files with high update rates. Second, because LAR and PAST neglect locality in file replication, they render significantly high cost for file updates 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. Third, CORP considers the update rate factor in file replication. It tries to replicate files with lower update rate in order to reduce the subsequent consistency maintenance cost. Its fewer replicas, shorter update message travel distance, and low update rate of replicas contribute to low-overhead file consistency maintenance.

4.4 Performance in a Dynamic Environment

This section 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 measured the maximum node utilization after every t, and averaged the simulation outcomes as the final results. The file join/departure rate was modelled as a Poisson process at rate 0.4 as in [2]; one file joins and one file departs every 2.5 seconds. We ranged node interarrival/interdeparture rate from 0.1 to 0.5 with step size of 0.1. When a replica node leaves, it needs to transfer all of its files and replicas to its nearby nodes. Figure 7 illustrates the average of maximum NUs versus node interarrival/interdeparture rate. We can make a number of observations. First, the 99th percentile maximum NUs of CORP are no higher than 1. This implies that CORP achieves the file replication goal of reducing overloaded nodes in dynamism. Second, the 99th percentile maximum NUs of LAR and PAST are higher than 1, and they increase as the node interarrival/interdeparture rate increases. Because LAR and PAST overlook the node available capacity factor, they are not able to control NU as in the static situation. In conclusion, unlike LAR and PAST, CORP ensures a load balance condition by file replication even in a dynamic environment.

4.5 Effect of Skewed Lookups

Skewed lookups due to non-uniform and time-varying file popularity and node interest variation may lead to load imbalance. In this section, we consider the effect of skewed lookups on file replication. We consider an "impulse" of 20480 queries for files whose IDs are distributed over a contiguous interval of the ID space from 0 to l, where l ranged from 500 to 4000 with 500 step size. We set the SU as 0.8 and tested the metrics of CDF of total load of replicated files, the total number of replicas, the NU, and the file consistency maintenance cost.

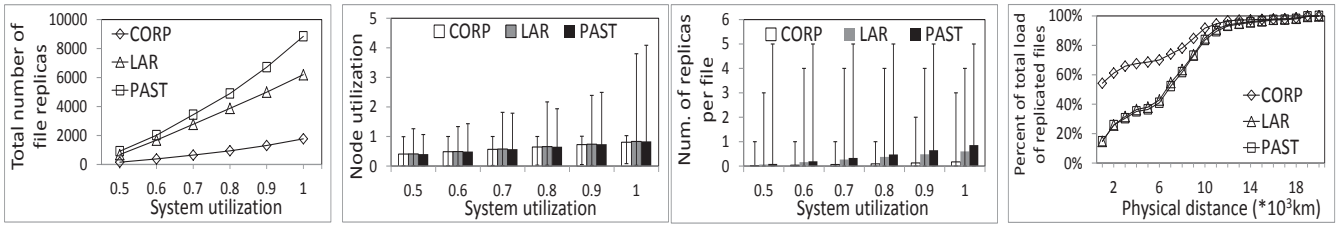
Figure 8(a) plots the total number of replicas versus the range of query keys. We find that the result drops with the increase of range. By comparing to the results without skewed lookups in Figure 3, we can see that skewed lookups generate much more replicas when the query range is smaller than 3500. It is expected because if query load concentrates on shorter ID space, more nodes will be overloaded and more replicas are needed to release their load. We also see that LAR and PAST generate much more replicas than CORP. With node available capacity consideration, CORP is able to deal with skewed lookups due to hot files, but LAR and PAST are not sufficiently flexible to handle skewed lookups. They replicate files to other nodes without considering their available capacity. Thus, the replica nodes will also be overloaded in skewed lookups, and subsequently they need to replicate files to other nodes, thus generating many more file replicas than CORP.

Figure 8(b) shows the median, 1st and 99th percentiles of NU rates in different query ranges. We can make two observations. First, all the 99th rates of CORP are no more than 1, which implies that even in skewed lookups, CORP can control a node's load under its capacity. Second, all the 99th rates of PAST and LAR are greater than 1, and PAST has higher 99th rates than LAR. The results are consistent with those in Figure 4 for the same reasons, and confirm the importance of considering node available capacity during file replication, especially in skewed lookups.

Figure 8(c) plots the file consistency maintenance cost with skewed lookups. It demonstrates that LAR and PAST incur dramatically higher cost than without skewed lookups in Figure 6, whereas the cost of CORP remains almost stable with different query ranges. The reason for higher cost in skewed lookups is because skewed lookups result in more overloaded nodes, leading to more replicas.

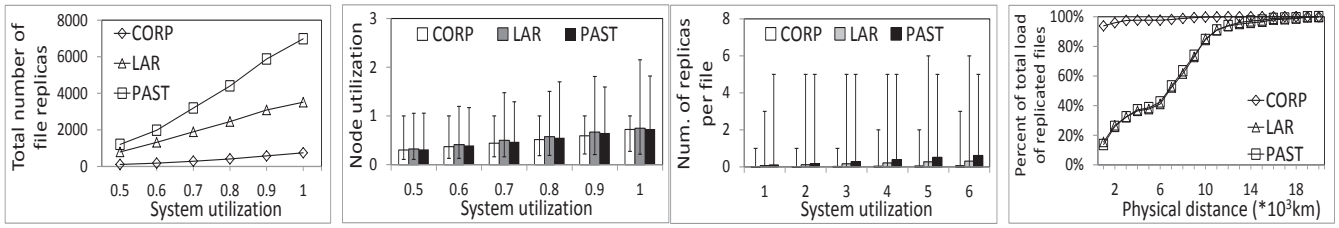
5 PlanetLab Performance Evaluation

We conducted experiments on the real-world PlanetLab [53] testbed to measure the performance of CORP. We randomly selected 350 nodes all over the world, and then generated 350×5 virtual nodes with each real node representing 5 virtual nodes. The nodes can be clustered into 169 groups using the clustering method. We resort to the statistics derived from



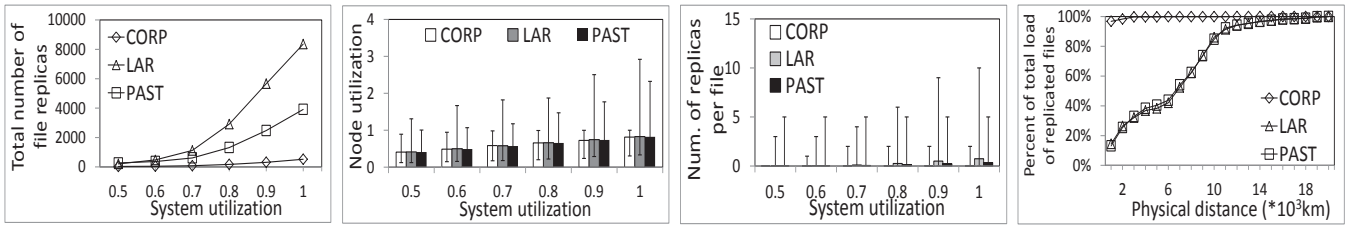
(a) Number of replicas (b) Node utilization (c) Replicas per file (d) CDF of load of replicated files

Fig. 9. Performance on the PlanetLab testbed.



(a) Number of replicas (b) Node utilization (c) Replicas per file (d) CDF of load of replicated files

Fig. 10. Performance in the Pareto distribution scenario with smaller bound ranges.



(a) Number of replicas (b) Node utilization (c) Replicas per file (d) CDF of load of replicated files

Fig. 11. Performance in the random distribution scenario.

4 million MSN video users' viewing behavior in the trace collected by Microsoft [54] for file size, visit rate, and file capacity in the experiment. We set the number of files to 11000, which is $1/10$ of the number of files in the trace. Following the distributions of fie size, visit rate and node capacity in the trace, we designed the experiment setting. The file size obeys the bounded Pareto distribution. We set the upper bound, lower bound and shape to $49/(5 \times 10^2)$MB, $1/(5 \times 10^2)$MB and 1, respectively. The distributions of the file visit rate and the capacity of nodes in the experiment obey the distributions in the trace, which approximately follow the Pareto and mirror-image Pareto distributions respectively. We also used the Pareto distribution for the file update rate. We set the lower bound, upper bound and shape to 1, 10, and 1, respectively. Since PAST [9] does not choose higher-load files to replicate, we let a node randomly choose files for replication in PAST in order to see the performance difference. We used the spherical law of cosines to calculate the distance between two PlanetLab nodes i and j as shown below:

$$d = r \arccos(\sin\phi_i \sin\phi_j + \cos\phi_i \cos\phi_j \cos\Delta\lambda) \quad (3)$$

where r is the radius of the earth, ϕ and λ denote the geographical latitude and longitude, and $\Delta\lambda = |\lambda_i - \lambda_j|$.

5.1 Capacity-aware File Replication

Figure 9(a) plots the total number of replicas versus SU. CORP generates much fewer replicas than LAR and PAST, and the total number of replicas increases as SU increases. The result is in line with that in Figure 3 due to the same reason that CORP chooses higher-load source files and tries to replicate one file to a node that can handle more visits of the file. We also see that LAR generates slightly fewer replicas than PAST. This is because PAST randomly selects files to replicate, while LAR gives higher-load file higher priority to replicate.

Figure 9(b) shows the 1st percentile, median and 99th percentile NU versus SU. The result is consistent with that in Figure 4. CORP can always maintain the NU of no more than 1, while LAR and PAST produce many overloaded nodes. This is because CORP does not assign more load beyond what a node can afford. However, PAST does not consider node available capacity. LAR aims to balance the load between nodes but also does not consider node available capacity.

Figure 9(c) shows the 1st percentile, average and 99th percentile of the number of replicas per file versus SU. Our result shows that the median is always 0, so we demonstrate the average here. We see that the average number increases as SU grows because higher SU leads to more replicas. We also see that the average number and the 99th percentile follow CORP<LAR<PAST. This is because CORP tries to reduce the number of replicas of each file in replication arrangement, while LAR and PAST do not pay attention to the number of replicas. Because PAST always creates 5 replicas for a file, its 99th percentile stays at 5. Also, the 99th percentile of LAR is always smaller than that of PAST because a node can always release its excess load before one of its file is replicated 5 times. We also see that the 99th percentile of CORP stays at 1 when the SU is lower than 0.8. This shows the effectiveness of CORP in constraining the number of replicas of a file.

5.2 Locality-aware File Replication

Figure 9(d) shows the CDF of load of replicated files versus the actual physical distance. We see that within 2×10^3km, 96% of file load is replicated in CORP, while 28% of file

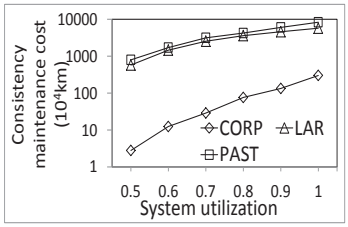
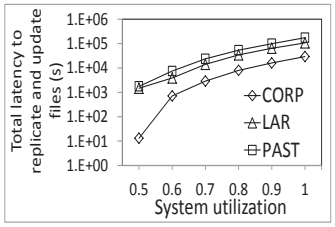
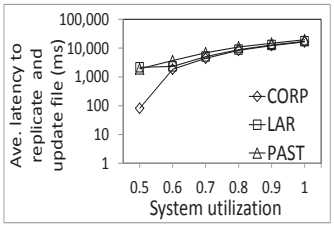


Fig. 12. Consistency maintenance cost.



(a) Latency to replicate and update files



(b) The average latency

Fig. 13. Latency to replicate and update files.

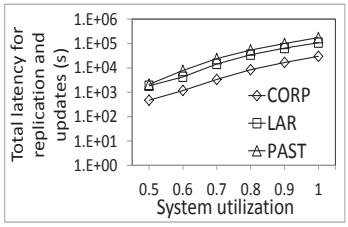
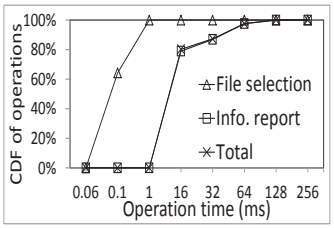
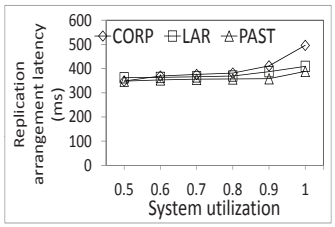


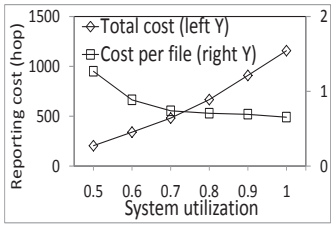
Fig. 14. Total latency.



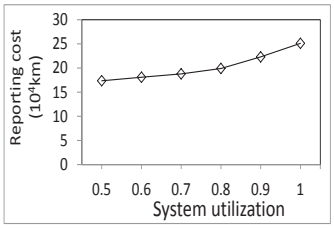
(a) Info. collection latency in CORP



(b) Replica rearrangement cost



(c) Reporting cost in hops



(d) Reporting cost in distance

Fig. 15. Reporting and replication arrangement latency and cost in CORP.

load is replicated in LAR and 25% of file load is replicated in PAST. This result is consistent with that in Figure 5. CORP tries to replicate files to physically close nodes in order to reduce the cost of file replication and consistency maintenance. Therefore, most of the load of replica files is within short distances. In contrast, LAR and PAST do not consider the locality in file replication. Thus, they replicate 96% of the file load within 15×10^3km. The file transmission and update propagation between physically-far nodes generate a high cost.

5.3 Performance in Balanced Scenarios

In this section, we conducted experiments in two less unbalanced scenarios. In one scenario, we reduced the upper bounds of file size, visit rate and update rate in the original Pareto distribution settings by one half. In the other scenario, we used random distribution with the same upper bounds and lower bounds in the original Pareto distribution settings. We reduced the lower and upper bounds of file size to 1/15 of the original in order not to overload the system. Figure 10 and Figure 11 show the performance of different methods in the two scenarios, respectively. For each performance metric, the experimental results in Figure 10 and Figure 11 are consistent with that in Figure 9 due to the same reasons. Because of smaller file size in the less unbalanced scenarios, the system is lightly loaded. Thus, an overloaded node can easily find a physically-close lightly loaded node to release its load. As a result, 93% of files in the Pareto distribution scenario and 96% of files in the random distribution scenario are replicated within 1000km, compared to 54% in Figure 9. All the experimental results confirm the efficiency of CORP in both file replication and consistency maintenance in systems with unbalanced and balanced distributions.

5.4 Lightweight File Consistency Maintenance

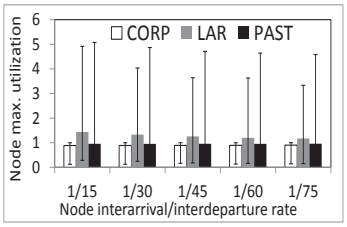
Figure 12 shows the consistency maintenance cost versus SU. We see that as SU increases, the consistency maintenance also increases. Higher SU leads to more overloaded nodes, and triggers more file replication operations and more replicas. Again, as in Figures 6(a) and (b), we see that CORP produces significantly lower consistency maintenance cost than PAST and LAR. This is caused by three reasons. First, CORP tries to replicate files in physically close nodes, generating short distances for update propagation. Second, CORP tries to reduce the number of replicas, producing less update messages. Third, CORP tries to replicate files with lower update rates. Because neither LAR nor PAST considers to minimize the physical distances, replica update rate and the number of replicas, they generate higher consistency maintenance cost than CORP.

5.5 Latency of Creating and Updating Replicas

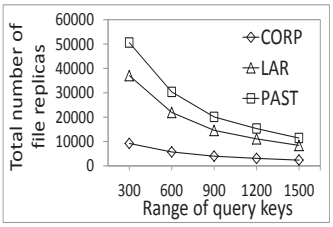
In this test, we measured the actual latency for creating and updating replicas. In order to test the performance in a heavy traffic, we enlarged the file size by 500 times. Figure 13(a) shows the total latency for both file replication and consistency maintenance phases. We see that when SU increases, the total latency increases in all three methods due to increasing number of replicas. CORP has lower latency than LAR and PAST because it creates fewer replicas and tries to replicate files to geographically close nodes. LAR has slightly lower latency than PAST because LAR has a little fewer replicas than PAST. The result indicates the higher efficiency of CORP in replicating and updating files.

Figure 13(b) shows the average latency per file for replicating and updating files versus SU. When SU increases, the average latency increases in all three methods. This is because more file replications generate more load and message transmissions in the system, which produces more traffic congestions and increases node response latency due to longer queuing, processing and propagation delay. When SU equals 0.5, the average latency of CORP is about 1% of the latency of PAST and LAR because CORP's locality-awareness decreases its propagation delay. As SU increases, more congestions occur, which leads to longer delay. Therefore, CORP produces similar average latency as those of PAST and LAR.

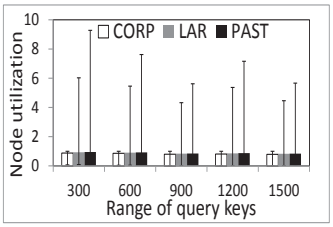
Figure 14 shows the latency for file replication (including information collection and replication arrangement in CORP) and updates. It shows that with extra source file selection and replication arrangement phases, CORP still generates much less latency than others. Comparing Figure 14 and Figure 13(a), we see that CORP has slightly longer latency due to the extra phases. These experimental results confirm



(a) Number of replicas



(b) Node utilization



(c) Consistency maintenance cost

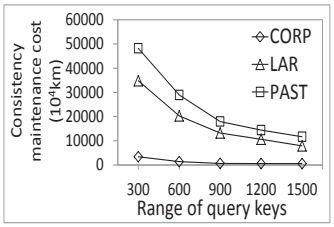


Fig. 16. Node utilization in churn.

Fig. 17. Performance of file replication protocols in skewed lookups.

the efficiency of CORP even with extra information collection and replication arrangement phases.

5.6 Latency of Information Collection and Replication Arrangement

In CORP's information collection phase, each node periodically selects source files (i.e., executes Algorithm 1) and sends information to its supernode. This experiment lasted one hour and each node conducted the operation every 5 minutes. The size of each parameter in IR was set to 1 unit. We measured the latency of each step and the whole phase for each node as shown in Figure 15(a). We see all file selection operations are within 8 ms (99.94% of operations are within 1ms). 87% of information report operations are within 32ms. 97% of information collection phases are within 64ms, and all are within 256 ms. In conclusion, the information collection takes very short time compared to the file replication and consistency maintenance latency as shown in Figure 13(a).

Figure 15(b) shows the latency for replication arrangement that determines which files should be replicated in which nodes. We see that the latency follows CORP>LAR>PAST. This is because CORP needs to order files based on their load, visit rate and update rate for both source file selection and replication arrangement, PAST only needs to find higher-load files to directly replicate to its neighbors, and LAR needs to find both higher-load files and query initiators that generate most load. We also see that CORP has similar time latency when SU$\leq$0.8, and it only generates 15% and 27% higher time latency than PAST when SU is 0.9 and 1, respectively. The results imply that CORP generates slightly longer latency for replica rearrangement, but can significantly reduce the cost for replication and consistency maintenance.

Figure 15(c) shows the total number of hops needed to report the information for file replication to the supernodes and the average hops per replica file. This figure indicates that the total reporting cost is proportional to the system utilization. This is because more nodes are overloaded when SU increases, and more files need to be replicated. The figure also shows that the reporting cost of each replica file decreases when the system utilization increases. With an increased system utilization, each overloaded node may have more than one file replica, which can be reported together, thus reducing the average hops of each replica file. We see that each replica file generates less than 1.5 reporting hops, which indicates that CORP does not produce a high cost in the information clustering stage.

Figure 15(d) shows the information reporting cost in CORP versus SU. Comparing it with Figure 12, we see that CORP's information reporting cost only constitutes a small percent of the consistency maintenance cost of LAR and PAST, which implies that CORP's extra step of information reporting generates a small amount of cost, but helps CORP significantly reduce the cost of consistency maintenance.

5.7 Node Utilization with Churn

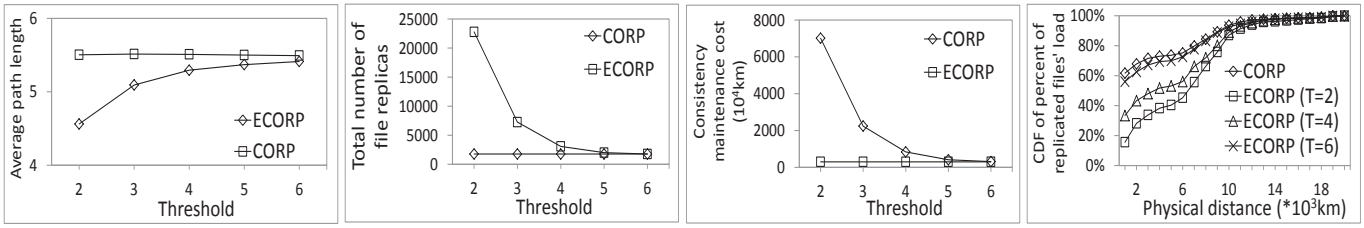
We run one round of file replication in churn. Figure 16 shows the 1st percentile, median and 99th percentile of maximum NU of a node versus SU. We see that the median increases as SU grows because higher SU leads to more replicas. We also see that the 99th percentiles of LAR and PAST always stay very high while that of CORP is always less than 1. This is because CORP avoids overloading nodes in file replication, while LAR and PAST do not pay attention to the available capacity of nodes. LAR replicates files to requester and PAST replicates files to a limited number of a server's neighbors. Thus, PAST generates higher 99th percentile than LAR.

5.8 Effect of Skewed Lookups

Figure 17(a) shows the total number of file replicas versus the range of query keys. We see that LAR and PAST generate more replicas than CORP due to the same reasons explained in Figure 8(a). Also, when the range of query keys is small, LAR and PAST generate much more replicas, while CORP only produces slightly more replicas. This is because the queries are on smaller number of nodes, which generates more overloaded nodes. Unlike LAR and PAST, CORP considers the available capacity of nodes chosen to replicate nodes in order to create fewer replicas. Also, it tries to constrain the number of replicas when choosing source files to replicate. Thus, it produces much fewer replicas than LAR and PAST. The number of replicas of PAST is larger than that of LAR, because a node in PAST randomly chooses a file to replicate rather than choosing high-load files as in simulation. The slower load release produces more replicas.

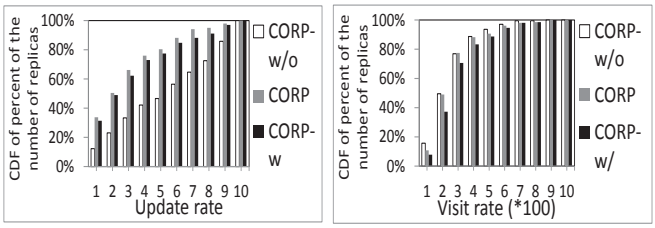
Figure 17(b) shows the median, 1st and 99th percentiles of NUs versus the query range. We see that the result is consistent with that in Figure 8(b) due to the same reasons. Also, PAST generates a much higher 99th percentile NU because of the same reason as in Figure 8. Finally, we observe that the median values of all methods are almost the same. This is because the loads of nodes not in the range of query keys are almost the same.

Figure 17(c) demonstrates the consistency maintenance cost with different query ranges. It shows that CORP produces significantly less maintenance cost than PAST and LAR due to its locality-awareness, fewer replicas and lower update rates of replicas. In contrast, LAR and PAST do not consider locality in file replication. Also, they do not try to reduce the file replicas and the update rates of the replicas. Therefore, they produce much higher consistency maintenance cost. PAST leads to higher maintenance cost than LAR because it has more replicas as shown in Figure 17(a).



(a) Lookup path length (b) Number of replicas (c) Consistency maintenance cost (d) CDF of total load of replicated files

Fig. 18. Performance of CORP and enhanced CORP.



(a) Update rate (b) Visit rate

Fig. 19. CDF of percentage of replicas.

5.9 Replication for Locality-aware Replica Retrieval

In this experiment, we test the effectiveness of the locality-aware replica retrieval policy described in Section 3.7. We use ECORP to represent CORP with this policy. Recall the system has $N = 350 \times 5$ nodes. Given each file's visit rate V in trace, we randomly selected a pool of $80\%N$ nodes and assigned $20\%V$ requests of the file to the $80\%N$ nodes randomly. We then assigned the remaining $80\%V$ requests to the remaining $20\%N$ nodes randomly. In the test, each node sends out its assigned requests. We call a node that replicates a file because its visit rate is $\geq$ T *replica client* and call such a source file *client replicated file*.

Figure 18(a) shows the average lookup path length of requests for all client replicated files versus the different values of threshold T. The average path length of ECORP is less than that of CORP. Also, the average path length of CORP always stays at 5.5, while that of ECORP increases as T increases. When T=6, ECORP and CORP achieve similar average lookup path length, which means there are few client replicated files in ECORP. Figure 18(b) plots the total number of replicas versus the different threshold values. We see that the number of replicas of ECORP drops sharply, while that of CORP remains nearly constant. Again, when T=6, ECORP and CORP generate almost the same number of replicas. When T is smaller, more requesters have replicas and they can retrieve the files from themselves, leading to shorter average path lengths. Figure 18(c) plots the consistency maintenance cost versus the threshold, which shows similar results as those in Figure 18(b). This is because more replicas produce more update propagations. These results show the effectiveness of the replication for locality-aware replica retrieval, and imply that the threshold should be set to an appropriate value so that the lookup path length can be greatly reduced with a moderate amount of additional replicas.

Figure 18(d) shows the CDF of the percentage of total load of replicated files. We see that CORP resolves more load within shorter physical distances, while ECORP resolves relatively less load within shorter distances. Also, smaller T generates less load in shorter distance and more load in long distance. This result is consistent with that in Figure 18(b). Since replicated files usually are not necessarily replicated to physically close nodes, more such replicas in ECORP lead to more load resolved in long distances.

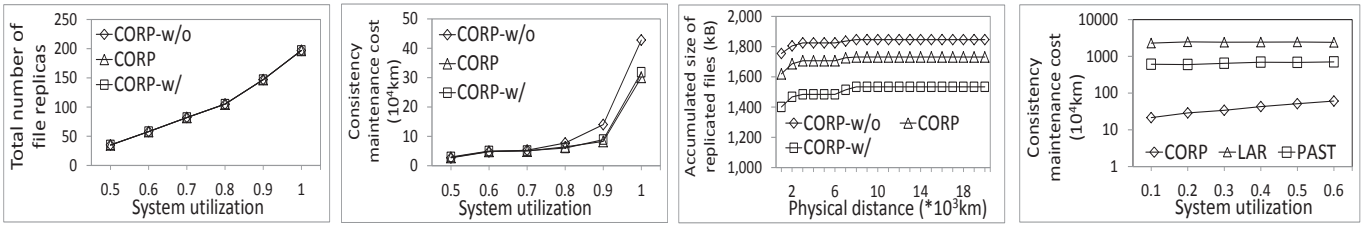
5.10 Integrated Consideration of Update and Visit Rates

In this test, we set the load of all files to 2000kB in order to see the effect of considering update rate and visit rate on the file replication and consistency maintenance performance. We use CORP-w/o to denote CORP without considering update rate or visit rate, and its source files is $\mathcal{F}_A$ after the first step in Algorithm 1. We use CORP-w/ to denote CORP with integrated consideration of update rate and visit rate as described in Section 3.3.

Recall that CORP reduces the replicas of frequently-updated or infrequently-visited files to decrease the overhead of file consistency maintenance and increase replica utilization. This experiment illustrates the effectiveness of CORP that considers visit rate and update rate. Figure 19(a) demonstrates the CDF of percent of the number of replicas. We see that CORP-w/o generates fewer replicas with low update rate than CORP-w/, which generates fewer replicas with low update rate than CORP. This is because CORP-w/o does not consider update rate when selecting files to replicate. In contrast, CORP-w assigns high priorities to barely-updated files to be replicated into high available capacity nodes in order to reduce consistency maintenance overhead. CORP gives higher priority to update rate than visit rate, while CORP-w/ gives relatively lower priority to the update rate than CORP since CORP-w/ integrates the update rate and visit rate. Consequently, CORP-w/ generates fewer replicas with low update rate than CORP.

Figure 19(b) plots the CDF of percent of the number of replicas versus the visit rate. We observe that CORP-w/o generates more replicas with low visit rate than CORP, which generates more replicas with low visit rate than CORP-w/. This is because CORP-w/o does not consider visit rate and CORP gives lower priority to visit rate than CORP-w/. From the figure, we conclude that compared to CORP-w/o, CORP and CORP-w/ generates fewer replicas for less popular files, and more replicas for more popular files, thus avoiding idle file replicas and enhancing replica utilization.

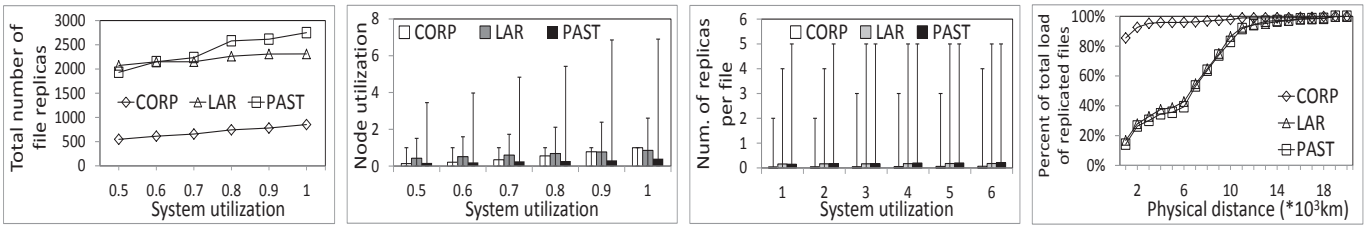
Figure 20(a) shows the total number of files replicas versus SU. We see that all three methods produce the same number of replicas. The three methods need to release the same amount of total load and all files have the same load. Also, the methods are all capacity-aware. As a result, they generate the same number of total replicas. This means that the integrated consideration of update and visit rates does not create more replicas. Figure 20(b) shows the consistency maintenance cost versus SU. It shows that CORP-w/o generates higher consistency maintenance than other two methods. This is because CORP and CORP-w/ try to replicate files with lower



(a) Total number of file replicas (b) Consistency maintenance cost (c) Accumulated size of replicated files

Fig. 20. The efficiency of integrated consideration of update and visit rates.

Fig. 21. Consistency maintenance cost with I/O trace.



(a) Number of replicas (b) Number of replicas (c) Replicas per file (d) CDF of load of replicated files

Fig. 22. Performance of file replication with I/O trace.

update rates, which decreases the consistency maintenance cost. CORP produces similar consistency maintenance cost as CORP-w/. This result verifies the effectiveness of the integrated consideration of update rate and visit rate.

Figure 20(c) shows the accumulated size of replicated files versus physical distance. It shows that the result follows CORP-w/o>CORP>CORP-w/. CORP-w/o does not consider file visit rate, CORP gives the lowest priority to visit rate, and CORP-w/ gives the same priority to both update rate and visit rate. Recall the three methods need to replicate the same amount of load, replicating files with higher visit rates leads to less total size of replicated files. As a result, file replicated size follows CORP-w/<CORP<CORP-w/o. These results imply that CORP-w/ replicates less amount of file sizes and hence has shorter latency when replicating files to remote nodes.

5.11 I/O Trade-driven Experiments

We then used a 4-hour I/O (read/write) trace for large scale applications known as CTH [55] in the previous network consisting of 350x5 nodes. The trace has 16,566 files, 3,972,284 file I/O calls and 3300 clients. The read and write block size varies in [1,2200]kB. The highest transaction read bandwidth is around 200MB/s. Then, the capacity of each node was assigned to 200/3300MB/s. The number of files held in a node was randomly chosen from [1,15], and the files are randomly chosen from the file pool. Figure 21 shows the consistency maintenance cost versus SU. Due to the same reason as in Figure 12, CORP saves much more consistency maintenance cost in other methods. Figure 22(a) shows the total number of replicas versus SU. Due to the same reason as in Figure 9(a), CORP generates fewer replicas than other methods. We also notice that the number of replicas of all three methods increases slowly as SU increases, and LAR and PAST generate relatively more replicas at low SUs. The trace shows that around 80% I/O operations are on 20% of the files. Thus, the nodes holding these 20% files easily become overloaded even when SUs are low, which is the reason for the above results. Figure 22(b) shows the median, 1st and 99th percentiles of NUs versus SU. Due to the same reason as in Figure 9(b), CORP always maintain the NUs no more than 1 and achieves better load balance than other methods. Figure 22(c) shows the median, 1st and 99th percentiles of the number of replicas per file versus SU. Due to the same reasons as in Figure 9(c), CORP produces fewer replicas per file than other methods. Figure 22(d) shows the CDF of load of replicated files versus physical distance. Due to the same reasons as in Figure 9(d), most of replicas in CORP are within short distances.

6 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 traditional file replication methods focus on hot spot elimination and query efficiency but neglect the efficiency of the subsequent file 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 considers file popularity, update rate, node available capacity and file load to reduce the number of replicas and the replicas for frequently-updated or infrequently-visited files. It further takes into account node locality 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 replication and consistency maintenance. Extensive experimental results on simulation and PlanetLab demonstrate the efficiency and effectiveness of CORP in comparison with other file replication protocols. In the future, we will study how to integrate the load, visit rate, and update rate into one metric for selecting files to replicate and replica nodes. We will also study how to set the threshold T to greatly reduce lookup path length at a moderate cost of additional replicas.

Acknowledgements

The authors are grateful to Dr. Jin Li at Microsoft Research for the trace data. This research was supported in part by U.S. NSF grants OCI-1064230, CNS-1049947, CNS-1156875, CNS-0917056 and CNS-1057530, CNS-1025652, CNS-0938189, Microsoft Research Faculty Fellowship 8300751, and Oak Ridge Award 4000111689. An early version of this work was presented in the Proc. of HiPC'09 [56].

REFERENCES

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, 2001.

[2] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *TON*, 1(1):17–32, 2003.

[3] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware*, pages 329–350, 2001.

[4] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: an infrastructure for fault-tolerant wide-area location and routing. *IEEE Journal on Selected Areas in Communications*, 12(1):41–53, 2004.

[5] P. Maymounkov and D. Mazires. Kademlia: a peer-to-peer information systems based on the xor metric. In *Proc. of IPTPS*, 2002.

[6] H. Shen, C. Xu, and G. Chen. Cycloid: a scalable constant-degree p2p overlay network. *Performance Evaluation*, 63(3):195–216, 2006.

[7] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proc. of PODC*, 2002.

[8] M. F. Kaashoek and R. Karger. Koorde: a simple degree-optimal distributed hash table. In *Proc. of IPTPS*, 2003.

[9] 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.

[10] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of SOSP*, 2001.

[11] K. Huang, T. Huang, and C. Y. Chou. LessLog: a logless file replication algorithm for peer-to-peer distributed systems. In *Proc. of IPDPS*, 2004.

[12] T. Pitoura, N. Ntarmos, and P. Triantafillou. Replication, load balancing and efficient range query processing in DHTs. In *Proc. of EDBT*, 2006.

[13] 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.

[14] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *Proc. of IPTPS*, 2002.

[15] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher. Adaptive replication in peer-to-peer systems. In *Proc. of ICDCS*, 2004.

[16] M. Theimer and M. Jones. Overlook: scalable name service on an overlay network. In *Proc. of ICDCS*, 2002.

[17] M. Roussopoulos and M. Baker. CUP: controlled update propagation in peer-to-peer networks. In *Proc. of USENIX*, 2003.

[18] L. Yin and G. Cao. DUP: dynamic-tree based update propagation in peer-to-peer networks. In *Proc. of ICDE*, 2005.

[19] J. Lan, X. Liu, P. Shenoy, and K. Ramamritham. Consistency maintenance in peer-to-peer file sharing networks. In *Proc. of WIAPP*, 2003.

[20] A. Datta, M. Hauswirth, and K. Aberer. Updates in highly unreliable, replicated peer-to-peer systems. In *Proc. of ICDCS*, 2003.

[21] Z. Li, G. Xie, and Z. Li. Locality-Aware consistency maintenance for heterogeneous P2P systems. *TPDS*, 2008.

[22] X. Chen, S. Ren, H. Wang, and X. Zhang. SCOPE: scalable consistency maintenance in structured P2P systems. In *Proc. of INFOCOM*, 2005.

[23] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *Proc. of the International Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2001.

[24] H. Shen. GeWave: geographically-aware wave for file consistency maintenance in P2P systems. In *Proc. of ICPP*, 2008.

[25] 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 *Proc. of the ASPLOS*, 2000.

[26] M Waldman, AD Rubin, and LF Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. of the 9th USENIX Security Symposium*, 2000.

[27] T. Hu, S. Ardon, and A. Sereviratne. Semantic-laden peer-to-peer service directory. In *Proc. of P2P*, 2004.

[28] M. Fontoura. Law-governed peer-to-peer auctions. In *Proc. of WWW*, pages 109–117, 2002.

[29] 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*, 2007.

[30] J. Y. Tham, S. L. Lee, C. E. Tan, and L. C. Tee. A distributed peer-to-peer platform for synchronized group collaboration and knowledge sharing. In *Proceeding of DCABES*, 2004.

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

[32] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache. In *Proc. of PODC*, 2002.

[33] B. Knutsson, M. M. Games, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Proc. of INFOCOM*, 2004.

[34] 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.

[35] H. Shen. EAD: an efficient and adaptive decentralized file replication algorithm in p2p file sharing systems. In *Proc. of P2P*, 2008.

[36] D. Rubenstein and S. Sahu. Can unstructured P2P protocols survive flash crowds? *IEEE/ACM Trans. on Networking*, (3), 2005.

[37] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache. In *Proc. of PODC*, 2002.

[38] H. Shen. IRM: integrated file replication and consistency maintenance in P2P systems. *TPDS*, 2009.

[39] H. Shen. Plover: proactive low-overhead file replication in structured p2p systems. *Proc. of JPDC*, 2009.

[40] J. Zhou, L. N. Bhuyan, and A. Banerjee. An effective pointer replication algorithm in P2P networks. In *Proc. on IEEE IPDPS*, pages 1–11, 2008.

[41] T. Liu, M. Bao, G. Chang, and Z. Tan. The improved research of Chord based on file-partition replication strategy. In *Proc. of IEEE HIS*, 2009.

[42] W. K. Lin, C. Ye, and D. M. Chiu. Decentralized replication algorithms for improving file availability in P2P networks. In *Proc. of IWQoS*, 2007.

[43] J. Ni, S. J. Harrington, and N. Sharma. Designing file replication schemes for peer-to-peer file sharing systems. In *Proc. of ICC*, 2008.

[44] H. Shen and C.-Z. Xu. Hash-based proximity clustering for efficient load balancing in heterogeneous DHT networks. *JPDC*, 2008.

[45] P. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *Proc. of INFOCOM*, 2005.

[46] M. Bienkowski, M. Korzeniowski, F. M. auf der Heide, and F. M. Heide. Dynamic load balancing in distributed hash tables. In *Proc. of IPTPS*, 2005.

[47] Y. Zhu and Y. Hu. Efficient, Proximity-aware load balancing for DHT-based P2P systems. *TPDS*, 16(4), 2005.

[48] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica. Load balancing in dynamic structured P2P systems. *Performance Evaluation*, 63(3):217–240, 2006.

[49] Z. Xu, M. Mahalingam, and M. Karlsson. Turning heterogeneity into an advantage in overlay routing. In *Proc. of INFOCOM*, 2003.

[50] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier. Space filling curves and their use in geometric data structure. *Theoretical Computer Science*, 181(1):3–15, 1997.

[51] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common API for structured P2P overlays. In *Proc. of IPTPS*, 2003.

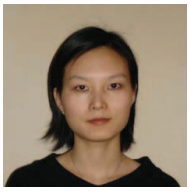
[52] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of INFOCOM*, 1996.

[53] PlanetLab. http://www.planet-lab.org/.

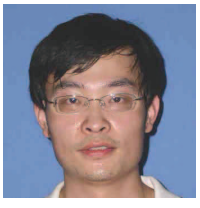
[54] C. Huang, J. Li, and K. W. Ross. Can internet video-on-demand be profitable. In *Proc. of SIGCOMM*, 2007.

[55] E. S. Hertel, Jr., R. L. Bell, M. G. Elrick, and et al. CTH: A Software Family for Multi-Dimensional Shock Physics Analysis. In *Sandia National Laboratories, Albuquerque, New Mexico, USA*.

[56] H. Shen. Corp: A cooperative file replication protocol for structured p2p networks. In *Proc. of HiPC*, 2009.



Haiying Shen Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Assistant Professor in the ECE Department at Clemson University. Her research interests include P2P networks, mobile computing, and cloud computing. She is a Microsoft Faculty Fellow of 2010 and a member of the IEEE and ACM.



Guoxin Liu Guoxin Liu received the BS degree in BeiHang University 2006, and the MS degree in Institute of Software, Chinese Academy of Sciences 2009. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include distributed networks, with an emphasis on Peer-to-Peer, data center and online social networks. He is a student member of IEEE.