J. Parallel Distrib. Comput. 69 (2009) 429–440

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A proactive low-overhead file replication scheme for structured P2P content delivery networks

Haiying Shen [a,*], Yingwu Zhu [b]

[a] Department of Computer Science & Computer Engineering, University of Arkansas, Fayetteville, AR 72701, United States
[b] Department of Computer Science & Software Engineering, Seattle University, Seattle, WA 98122, United States

ARTICLE INFO

Article history:
Received 26 October 2008
Received in revised form
14 February 2009
Accepted 14 February 2009
Available online 12 March 2009

Keywords:
File replication
Peer-to-peer
Distributed hash table
Content delivery networks

ABSTRACT

File replication is a widely used technique for high performance in peer-to-peer content delivery networks. A file replication technique should be efficient and at the same time facilitates efficient file consistency maintenance. However, most traditional methods do not consider nodes' available capacity and physical location in file replication, leading to high overhead for both file replication and consistency maintenance. This paper presents a proactive low-overhead file replication scheme, namely Plover. By making file replicas among physically close nodes based on nodes' available capacities, Plover not only achieves high efficiency in file replication but also supports low-cost and timely consistency maintenance. It also includes an efficient file query redirection algorithm for load balancing between replica nodes. Theoretical analysis and simulation results demonstrate the effectiveness of Plover in comparison with other file replication schemes. It dramatically reduces the overhead of both file replication and consistency maintenance compared to other schemes. In addition, it yields significant improvements in reduction of overloaded nodes.

1. Introduction

Over the past years, the immense popularity of the Internet has produced a significant stimulus to peer-to-peer (P2P) content delivery overlay networks. A recent study shows that more than 49%–83% of Internet traffic is generated by P2P applications. The percentage of such traffic in the total aggregate traffic on the Internet has increased significantly and become almost pervasive [27]. Furthermore, the access to these objects is highly repetitive and skewed towards the most popular ones. Such objects can exhaust the capacity of a node. The capacity of a node means the ability of the node to handle queries in a given time interval. The capacity is determined by a nodes access bandwidth, processing power, disk speed, etc. If a node receives a large volume of requests for an object at one time, it becomes a hot spot, leading to delayed response. File replication techniques to replicate a hot file to some other nodes have been widely used to avoid such hot spots by distributing the file query load among a number of nodes.

In addition to file replication, an effective consistency maintenance method is also highly demanded by P2P content delivery overlay networks. Without effective replica consistency maintenance, a P2P content delivery network is limited to providing only static or infrequently-updated file sharing. On the other hand, newly-developed P2P applications need consistency support to deliver frequently-updated contents, such as directory service [42], online auction [12,26], and remote collaboration [49]. Therefore, a file replication technique should be efficient and farseeing enough to facilitate low-overhead and timely file replica consistency maintenance.

However, the two issues of file replication and file consistency maintenance have been typically addressed separately, despite the significant interdependency of file consistency maintenance on file replication. Most traditional file replication methods in structured P2P content delivery networks determine replica nodes based on node IDs [31,7,16,28,14] or query path [36,15,30,46]. ID-based methods determine replica nodes based on the relationship between the node ID and the file's ID, and path-based methods choose replica nodes in the file query path from the file requester to the file provider. Both groups of methods assume that replica nodes have available capacity for replicas. This assumption will make the problem of hot spots even more severe since replica nodes may be overloaded nodes. These methods also make file replication without considering node locality, which is a vital factor for efficiency of file replication and consistency maintenance.

This paper presents a *p*roactive *l*ow-*over*head file replication scheme, namely *Plover*. *Plover* not only achieves high efficiency in file replication but also supports low-overhead and timely consistency maintenance. It makes file replication among physically

* Corresponding address: University of Arkansas, Department of Computer Science & Computer Engineering, 311 Engineering Hall, 800, 72701 Fayetteville, AR United States.
E-mail addresses: hshen@uark.edu (H. Shen), zhuy@seattleu.edu (Y. Zhu).

0743-7315/$ – see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2009.02.008

close nodes based on node available capacities. With nodes' available capacity consideration, it avoids exacerbating the hot spot problem by choosing nodes which have sufficient capacity for the replicas. In addition, it determines the number of file replicas based on nodes' available capacity which eliminates unnecessary replicas, resulting in less consistency maintenance overhead. With locality consideration, it enables the file replication and consistency maintenance to be conducted among physically close nodes, leading to considerable reduction of overhead. *Plover* further adopts a lottery scheduling method to achieve file query load balance between replica nodes.

The rest of this paper is structured as follows. Section 2 presents a concise review of representative file replication approaches for structured P2P systems. Section 3 presents the *Plover* file replication scheme, corresponding consistency maintenance method and query redirection algorithm. Section 4 shows the performance of *Plover* in comparison with representative file replication schemes in terms of a variety of metrics, and analyzes the factors effecting file replication performance. Section 6 concludes this paper with remarks on possible future work.

2. Related work

Driven by tremendous advances of P2P file sharing systems, numerous file replication methods have been proposed for structured P2P content delivery networks. One group of file replication methods determines the replica node based on IDs [31,7,16,28]. In PAST [31], each file is replicated on a set number of network nodes whose IDs match most closely to the file's ID. The number is chosen to meet the availability needs of a file, relative to the expected failure rates of individual nodes. It has a load balancing algorithm for non-uniform storage node capacities and file sizes. CFS [7] 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 servers immediately after the block's successor on the Chord ring in order to increase availability. LessLog [16] determines the replicated nodes by constructing a lookup tree based on IDs to determine the location of the replicated node. In HotRoD [28], hot arcs of peers are replicated and rotated over the identifier space. Ghodsi et al. [14] proposed a symmetric replication scheme in which a number of IDs are associated with each other, and any item with an ID can be replicated in nodes responsible for IDs in this group.

Another group of file replication methods [36,15,30,46,4,37] chooses replica or caching nodes based on file query path. Stading et al. [36] proposed the Backslash system, in which a node pushes a cache to one hop closer to requester nodes when overloaded. LAR [15] let overloaded nodes replicate at the query initiator and create routing hints on the reverse path. CUP [30] and DUP [46] cache metadata along the lookup path with consistency support.

Rigid assumptions on how replications of objects happen in the system such as those based on ID, requester or path may even make the overload problem severe, as the replica nodes may be already overloaded. Allowing for more proactive replications of objects, where an object may be replicated at a node without restriction, avoids such problems. Oceanstore [18], objects are replicated and stored on multiple servers for security concerns without restricting the placement of replicas. It uses a different approach to handling updates by introducing some degree of centralization. In [6], a decentralized replication solution is used to achieve practical availability in unstructured P2P systems. It uses randomized decisions extensively together with a novel application of the Reed Solomon erasure codes, without considering replica consistency.

Another group of work connects file replicas number with file popularity. Lv et al. [23] and Cohen and Shenker [5] considers static replication in combination with a variant of Gnutella searching using random walkers in order to reduce random search times. The authors show that replicating objects proportionally to their popularity achieves optimal load balance but has varied search latency, while uniform replication has the same average search latency for all files but causes load imbalance. Square-Root replication replicating files proportionally to the squareroot of their popularity is such that both average search size and utilization rate vary per object, but the variance in utilization is considerably smaller than with Uniform, and the variance in average search size is considerably smaller than with Proportional.

Tewari and Kleinrock [39–41] showed that proportional replication in which the number of replicas is proportional to the file request rates can optimize flooding-based search, download time, and workload distribution. They also showed that local storage management algorithms like the least recently used policy (LRU) automatically achieve near-proportional replication and that the system performance with the replica distribution achieved by LRU is very close to optimal. APRE [43] adaptively expands or contracts the replica set of an object in order to improve the sharing process and achieve a low load distribution among the providers. To achieve that, it utilizes search knowledge to identify possible replication targets inside query-intensive areas of the overlay.

File systems other than P2P systems also replicate content at multiple sites to decrease access latency seen by end-users and optimal allocation of system storage (e.g. [20]). Sandhu and Zhou [33] proposed a cluster-based dynamic file replication system called Frolic. Instead of keeping copies of a widely shared file at each client workstation, these files are dynamically replicated onto the cluster file servers, so that they become locally available. In the Frolic system, the dynamic replication of unstable files can be effectively used for improving file access times in large-scale distributed file system environments. *Plover* is similar to the Frolic system in terms of relying on cluster structure. Unlike on the Frolic system that distributes replicas among servers, *Plover* distributes replicas among cluster nodes as well as cluster servers. Most importantly, it takes into account file popularity and update frequency for economical file replication. The Ivy system [24] stores all file data in a set of logs using the DHash distributed hash table with little control on data consistency. The Om system [47] is for replica regeneration, i.e., creating new replicas in response to replica failures for reducing management costs and improving the availability of large-scale distributed systems. The work in [29] developed an analytical optimization theory for benchmarking the performance of replication/replacement algorithms, including algorithms that employ erasure codes. FarSite [1,9,10] is a distributed file system with the strong persistence and availability of a traditional file system. The FarSite filesystem uses the same number of replicas (i.e., three) for each file. In contrast to a file system, the goal of a P2P community is not to provide strong file persistence, but instead, maximal content availability. Thus, in a P2P community, the number of replicas of a file depends on the popularity of the file. Pangaea [32] creates replicas aggressively to improve overall performance. By organizing all replicas of a file in a strongly-connected graph, it propagates an update from one server to the others through flooding, which does not scale well with a large number of replicas. In [13], Gedik et al. used a dynamic passive replication scheme to provide reliable service for a P2P Internet monitoring system, where the replication list is maintained by each Continual Queries owner. Coda [17] uses replication to improve availability at the expense of consistency and introduce specialized conflict resolution procedures. Sprite [25] also uses replication and caching to improve availability and performance, but has a guarantee of consistency that incurs a performance penalty in the face of multiple writers.

The rigid ID-based or path-based replica node determination in structured P2P systems may make the overloaded problem even more severe, since the replica nodes chosen might not have enough

available capacity for a replica. Although PAST employs a load balancing algorithm, and CFS adopts file division, these strategies come at a price of extra overhead and complexity. In addition to the nodes' available capacity, these file replication schemes do not take locality into account. Fessant et al. [11] indicated that geographical clustering is present and can be leveraged in order to yield significant performance improvements. Based on this principle, this paper presents the *Plover* file replication scheme. By considering nodes' available capacity and locality, *Plover* not only achieves high efficiency in file replication but also proactively avoids extra overhead and facilitates efficient file consistency maintenance.

Along with file replication, numerous file consistency maintenance methods have been proposed. Lan et al. [19] proposed to use flooding-based push for static files and polling for dynamic files. In a hybrid push/poll algorithm [8], flooding is substituted by rumor spreading to reduce communication overhead. Li et al. [21] presented a scheme that forms the replica nodes into a proximity-aware hierarchical structure (UMPT). SCOPE [2] builds a replica-partition-tree for each key based on its original P2P system. CUP [30] is a protocol for performing Controlled Update Propagation to maintain caches of metadata in peer-to-peer networks. To improve CUP, Yin and Cao proposed the Dynamic-tree based Update Propagation (DUP) scheme [46], which builds a dynamic update propagation tree on top of the existing index searching structure with very low cost. In Freenet [3], an update is routed to other nodes based on key closeness.

3. The design of Plover

3.1. Proactive low-overhead file replication

Plover realizes locality-aware file replication through building a supernode network which clusters physically close nodes. In general, supernodes are nodes with high capacity and fast connections. For simplicity, we define a node with capacity greater than a predefined threshold as supernode; otherwise a regular node. Based on hash-based proximity clustering [34], *Plover* assembles all supernodes into a self-organized structured P2P for file replication. The supernode network can also serve other purposes such as load balancing and express routing.

Before we present the details of the supernode network construction, let us introduce a landmarking method to represent node closeness on the Internet by indices. Landmark clustering has been widely adopted to generate proximity information [45]. It is based on the intuition that nodes close to each other are likely to have similar distances to a few selected landmark nodes. We assume m landmark nodes that are randomly scattered in the Internet. Each node measures its physical distances to the m landmarks, and uses the vector of distances $\langle d_1, d_2, \ldots, d_m\rangle$ as its coordinate in Cartesian space. Two physically close nodes will have similar landmark vectors. A Hilbert curve [45] is further used to map m-dimensional landmark vectors to real-numbers. That is, $R^m \longmapsto R^1$, such that the closeness relationship among the points is preserved. We call this number the *Hilbert number* of the node.

Algorithm 1: Pseudo-code for node joining in *Plover* supernode structured P2P network containing node n'.

```
n.join(n'){
  ID=n.Hilbertnum;
  //find the supernode closest to n
  s=n'.find_supernode(n.ID);
  if n's capacity<a predefined threshold then {
    //n is a regular node, taking s as its supernode
    supernode=s;
    supernode.addto_clientlist(n); }
  else
    //n is a supernode
    if n.ID==s.ID then
      s.addto_backuplist(n);
    else
      //join in supernode network, initialize neighbors
      predecessor=nil;
      //find its successor
      if s.ID%2^d >n.ID%2^d then
        successor=s;
      else
        successor=s.successor;
}
```

Algorithm 2: Pseudo-code for node n leaving from *Plover* supernode structured P2P network.

```
n.leave( ){
  if successor!=nil then
    //n is a client
    supernode=nil;
  else
    //n is a supernode
    if backuplist.size>0 then {
      s=backuplist.getone();
      //choose one backup, transfer supernode information to it
      s.clientlist=clientlist;
      s.backuplist=backuplist; }
    else
      //no backup supernode, transfer regular nodes accordingly
      for i = 0 up to clientlist.size do {
        client=clientlist[i];
        if predecessor is closer to client than successor then
          move client to predecessor;
        else
          move client to successor; }
}

//n is notified of supernode change
n.supernode_change_notify(s){
  supernode=s;
  supernode.addto_clientlist(n);
}
```

Plover directly uses a node's Hilbert number as its logical node ID, and let supernodes and regular nodes act as the nodes and keys in the top-level supernode structured P2P respectively. The supernode network can be any type of structured P2P such as Chord and Pastry. Based on the key assignment protocol that a key is stored in a node whose ID is the closest to the key, a regular node is assigned to a supernode whose ID is closest to the node's ID. Since node ID represents node physical location closeness, thus regular nodes are connected to their physically closest supernode. As a result, the physically close nodes will be in the same cluster or in nearby clusters with supernodes. In the case when a number of supernodes have the same Hilbert numbers, one supernode is chosen and others become its clients while acting as its backup nodes. The consistent hashing for key assignment protocol requires relatively little re-association of regular nodes to dynamically designated supernodes as nodes join and leave the system. As a result, nodes in one cluster are physically close to each other, close clusters/supernodes in logical ID space are also physically close

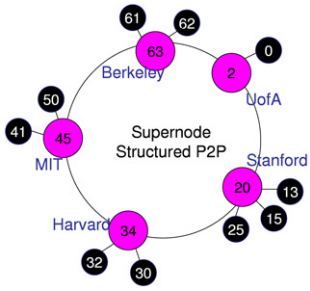


Fig. 1. Supernode structured P2P in Plover.

to each other, and the application-level connectivity between the supernodes in the top-level supernode network is congruent with the underlying IP-level topology.

To find a supernode responsible for an ID, a regular node forwards a query to its supernode, which uses structured P2P routing algorithm on supernode network. Structured P2P protocols dealing with node and item joins and departures can be directly used to handle supernode and regular node joins and departures in the supernode network. When a supernode or regular node joins the supernode structured P2P, it must know at least one node, and uses the supernode network routing algorithm to find its place. Algorithm 1 and 2 show the pseudocode of node join and departure in the supernode network respectively. The algorithms help to maintain the mapping between regular nodes and supernodes in dynamism where nodes join, leave and fail.

Fig. 1 shows an example of a supernode structured P2P in Chord. By taking advantage of Hilbert number and key assignment protocol, physically close nodes are grouped into a cluster with a supernode and all supernodes constitute Chord. Each supernode functions as a node in a flat Chord. If $n40$ wants to join in the system, $n40$ asks its known node $n2$ to find the supernode with ID closest to 40, which is $n45$. If $n40$ is a supernode, $n45$ moves $n41$ to $n40$. The maintenance of supernode network is the same as that of Chord. If $n40$ is a regular node, it becomes a client of $n45$. If a node, say $n45$, wants to leave the system, it moves $n41$ to $n34$, and $n50$ to $n63$. If $n41$ wants to leave the system, it only needs to disconnect its link to $n45$. The following theorems illustrate the performance of these operations.

Theorem 3.1. *With high probability*[1]*, the number of nodes that must be contacted for a node to find its supernode in an N_s-supernode structured P2P network is $O(\log N_s)$.*

Proof. It was proved in [38] that in Chord, w.h.p, the number of nodes that must be contacted to find a successor in an N-node network is $O(\log N)$. In an N_s-supernode structured P2P network, a node first needs to contact an existing node which will forward the join message to its supernode. The number of supernodes that must be contacted to find the supernode of the newly-joined node is $O(\log N_s)$. Therefore, the total contacted nodes number is $O(\log N_s) + 1 \approx O(\log N_s)$. ■

Theorem 3.2. *With high probability, any supernode joining or leaving an N_s-supernode structured P2P network will use $O(\log^2 N_s)$ messages to re-establish the supernode structured P2P network routing information.*

Proof. It was proved in [38] that in Chord, w.h.p, any node joining or leaving an N-node Chord network will use $O(\log^2 N_s)$ messages to re-establish the Chord routing information. It can be applied to an N_s-supernode Chord network directly. ■

[1] An event happens with high probability (w.h.p.) when it occurs with probability $1 - O(n^{-1})$.

Theorem 3.3. *With high probability, if a successor list of length $r = O(\log N_s)$ in a supernode structured P2P network that is initially stable, and then if every supernode fails with probability 1/2, the expected time for a node to find its supernode in the failed network is $O(\log N_s)$.*

Proof. It was proved in [38] that in Chord, w.h.p, if a successor list of length $r = O(\log N)$ in a Chord network that is initially stable, and then every node fails with probability 1/2, then the expected time execute find_successor in the failed network is $O(\log N)$. It can be applied to the supernode structured P2P network. ■

Plover relies on the supernode network for locality-aware file replication. We assume that node i's capacity, C_i, is a quantity that represents the number of bits that node i can transfer during a given time interval T. We define the load of node i, L_i, as the number of bits needed to transfer due to file queries it receives over time T. We refer to node with load $L_i \leq C_i$ as a light node; otherwise a overloaded node. Specifically, each overloaded node reports the information of its hot files, and each lightly loaded node reports its *available capacity* $C - L$ to its supernode periodically. As a result, the information of physically close nodes gathers together in the supernode. For example, nodes $n61$ and $n62$ report their load information to $n63$ periodically, which conduct node mapping for file replication, and notify heavy nodes to replicate files to light nodes. The supernode conducts node mapping with node capacity consideration for file replication, and notifies overloaded nodes to replicate files to lightly loaded nodes.

Algorithm 3: Pseudo-code executed by node n for file replication in *Plover*.

```
n.replicate( ){
  if n is a regular node then{
    if C_n > L_n then{
    //n is a lightly loaded node
    report n's available capacity ΔC_n = C_n − L_n}
    else { //n is an overloaded node
      //find hot files and report their sizes and visit rates
      order its files in a descending order based on S × V in f[ ]
      L=0
      for each f[i++] do {
        L+=f[i].l //l represents the load of f[i]
        if L ≥ L_n − C_n then
          break}
      report the size and visit rate of files from f[0] to f[i]
      to its supernode } }
  else n is a supernode then{
    receive ΔC and S and V of hot files from n's regular nodes
    order ΔC in descending order in ΔC[ ]
    //top item is ΔC[m]
    order hot files in descending order based on V in f[ ]
    //assign hot files to lightly loaded nodes for file replication
    for each f[i++] do {
      restV=f[i].V
      while restV>0 {
      //restV is the visit rate has not been resolved
        v=restV //v is the visit rate to be resolved
        restV=0
        //find the number of visits that can be assigned to
        the node with ΔC[m]
        while f[i].S × f[i].V > ΔC[m] && v ≠ 0 {
          v−−
          restV++ }
        if v==0 then {
        //no node in n's cluster has sufficient capacity
```

```
        to replicate the file
          contact predecessors and successors for replica node
          get replica node
          assign the file to be replicated to the replica node }
        else{
        //assign the file to be replicated to a node in n's cluster
          assign f[i] to be replicated to the node with ΔC[m]
          put the ΔC[m] − f[i].l back to the ΔC[ ]
          and reorder ΔC[ ] }}}}
}
```

Algorithm 4: Pseudo-code executed by an original file owner n for query redirection in *Plover*.

```
n.redirect( ){
  //assume n has m replica nodes rn[0]-rn[m] for its file f
  receive file query for file f
  if n is not overloaded then
    reply the requester with file f
  else { //n is overloaded
    //get the total visit rates responsible by all replica nodes
    for each rn[i++] do
      totalV+=rn[i].V
    //choose a replica node using lottery scheduling
    //for query redirection
    randomly choose a value r within [0,totalV-1]
    i=0
    for each rn[i++] do {
      sum+=rn[i].V
      if sum≥ r then{
        //rn[i] is the replica node for query redirection
        redirect the file query to replica node rn[i]
        break  }}}
}
```

During the mapping, a node is chosen as a replica node for a file only if it has sufficient capacity for the file, which avoids exacerbating the hot spot problem. In addition, nodes with higher available capacity have a higher priority to be replica nodes, which helps to reduce unnecessary node replicas. The load caused by file access should be measured by the number of bits needed to be transferred during time interval T. Therefore, the load is determined by file size and file popularity. File popularity can be measured by file visit rate which is the number of visits during a certain time period, say one second. Let $V_{i,k}$ denote the visit rate of file k in node i, $L_{i,k}$ denote its load and $S_{i,k}$ denote its size. Then, $L_{i,k} = S_{i,k} \times V_{i,k}$. For instance, a supernode needs to find a replica node for a hot file with $V = 3$. If there are three options i, j and k, which can afford load of 3 visit rate, 2 visit rate and 1 visit rate of this file respectively, then node i will be selected as replica node for this file. Therefore, providing higher available capacity nodes higher priority to be replica nodes can reduce redundant replicas, and hence help to reduce file consistency maintenance overhead. In this node mapping phase, node location can also be considered, in which a node frequently receives queries for a specific file can have a replica of the file. These techniques are orthogonal to our study in this paper.

All hot files in a cluster may not be resolved in their cluster. As mentioned earlier, the distances between a supernode and its successors or predecessors in the supernode network represent their physical distances. The distances between a node and its sequential nodes are usually smaller than distances between the node and randomly chosen nodes in the entire ID space. Therefore, for locality consideration, a supernode probes its physically nearby supernodes by probing its successors or predecessors in sequence for unresolved hot files. The supernode structured P2P enables nodes to communicate and conduct file replication between physically close nodes. It not only enhances the efficiency of file replication but also file consistency maintenance. The load balancing algorithm in [35] can adopted into *Plover* to handle the load imbalance problem. Algorithm 3 demonstrates the pseudocode executed by a node for file replication in *Plover*.

Theorem 3.4 sheds insight into the subtleties of the *Plover* file replication scheme.

Theorem 3.4. *W.h.p., in a N-node structured P2P network, to replicate a file with S size and V visit rate, ID-based methods (e.g. PAST) will produce V replicas, path-based methods (e.g. LAR) will yield V* log *N replicas, and* Plover *will generate replicas* $\leqslant V$ *on average.*

Proof. In ID-based methods such as PAST, the number of replicas is chosen to meet the availability needs of a file, which is the visit rate. In path-based method such as LAR, a file is replicated along its routing path. Structured P2P networks have log N lookup path length on average, therefore LAR yields V log N replicas on average. *Plover* considers nodes' available capacity. If it can find a node whose available capacity is $\geqslant V \cdot S$, the node is the only replica node for the file. Hence, *Plover* generates replicas $\leqslant V$. ■

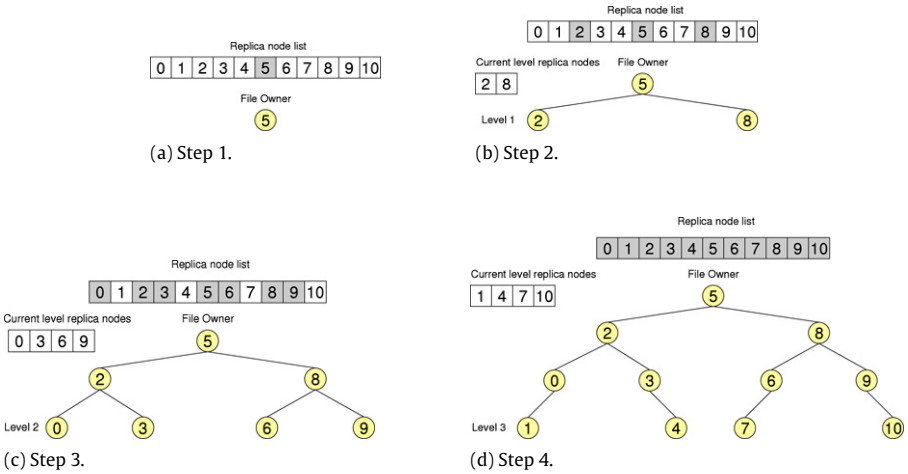
3.2. File consistency maintenance

Plover eliminates the need for a specific file consistency maintenance method such as [30,46,21,2,22] and facilitates efficient file consistency maintenance. Currently, most file consistency maintenance methods build a structure for each file. In this case, there is no unnecessary update message sent to non-replica nodes, but maintaining structure incurs cost overhead. On the other hand, push/pull or flooding method does not need structure maintenance, but more overhead is needed for propagation and some non-replica nodes may get update messages.

To combine the advantages of both methods, *Plover* specifies a threshold T for the number of replica nodes. If there are replica nodes of a file in a group of clients of a supernode, the supernode chooses the highest-capacity replica node and notifies the file owner of this node. If the number of replica nodes of a file in the group of clients of the supernode is larger than T, the supernode builds a k-nary tree structure consisting of the replica nodes with the highest-capacity replica node as the tree root. Otherwise, the supernode notifies the highest-capacity replica node of all other replica nodes.

When the number of highest-capacity replica nodes received by a file owner exceeds T, the file owner forms these roots into a k-nary tree taking itself as the tree root. Thus, a hierarchical structure is constructed. In the upper level of the structure, a tree is formed by the highest-capacity replica nodes from different clusters of physically close nodes, and the file owner is the tree's root. In the lower level of the structure, a tree is formed by physically close replica nodes. For a file update, if there is a tree in the upper level, the file owner propagates the update message downwards along the tree. Otherwise, it uses broadcasting to send update messages to the highest-capacity replica nodes. After a highest-capacity node receives the update message, if it is a tree root, it further propagates the message downwards along the trees in the lower level. Otherwise, it broadcasts the messages to the replica nodes in its group.

Fig. 2 shows an example to build a 2-nary tree from a number of nodes. First, the nodes are put into a list with the tree root in the middle. This step is shown in Fig. 2(a). Node 5 becomes the tree root. Then, the index space [0, 10] of the list is partitioned into $k = 2$ parts: [0, 4] and [6, 10]. The node in the middle of each index



(a) Step 1. (b) Step 2.

(c) Step 3. (d) Step 4.

Fig. 2. Update tree construction.

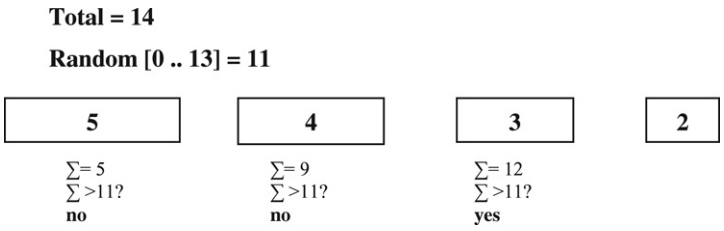


Fig. 3. Example of lottery scheduling.

space part becomes the child of node 5. The nodes are node 2 and node 8. This step is shown in Fig. 2(b). After that, index space [0, 4] is further partitioned into $k = 2$ parts: [0, 1] and [3, 4], and the nodes in the middle of the two parts become node 2's children. The two nodes are node 0 and node 3. Index space [6, 10] is also further partitioned into $k = 2$ parts: [6, 7] and [9, 10], and the nodes in the middle of the two parts become node 8's children. The two nodes are node 6 and node 9. This step is shown in Fig. 2(c). In the last step, the rest node in each part becomes the child of the previous level node. Finally, the tree is constructed as shown in Fig. 2(d).

Since the replica nodes in the same group are physically close nodes, message propagation among these nodes reduces update propagation overhead. Therefore, *Plover* facilitates low-overhead and timely consistency maintenance because update messages travel along short physical distances.

3.3. Query redirection for load balance

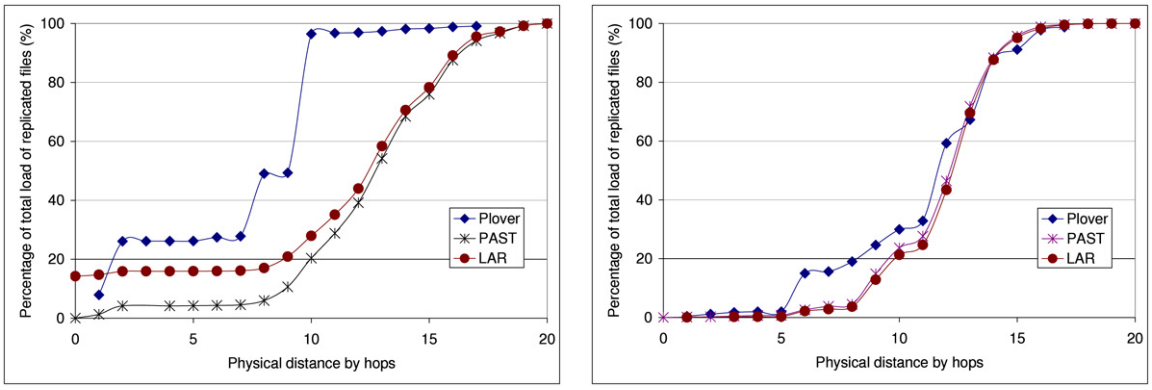
If an overloaded original file owner receives a file request, it will forward the request to one of the file's replica nodes. A question is how to choose the replica node so that the query load can be distributed based on nodes' available capacity. An efficient query redirection algorithm should effectively allocate the query load to replica nodes in balance. *Plover* adopts lottery scheduling [44] for query redirection. Lottery scheduling is a method that efficiently implements proportional-share resource management. In the scheduling, a node has tickets, and its allocated resource is proportional to the number of tickets that it holds. *Plover* regards the visit rate of a file as tickets. The number of tickets a replica node holds equals the visit rate it is responsible for. The overloaded node selects a replica node by picking a ticket from the replica nodes at random and chooses the replica node that holds this winning ticket which is randomly generated. The visit rate ratios among replica nodes is the expected ratios of load that are responsible for.

Fig. 3 shows an example [44] of lottery scheduling, where the third replica node is selected. In the example, the original file owner has four replica nodes for its file. The file's visit rates responsible by the replicas nodes are 5, 4, 3 and 2 respectively. Then, the total visit rate of the file responsible by all replica nodes is 14. When the original file owner receives a query for the file, if it is overloaded, it will choose a replica node to redirect the query. First, it randomly chooses a value within [0,13]. Let us say the random number is 11. It then accumulates the visit rates of replica nodes one by one until the sum is no less than 11. Specifically, because the visit rate of the first replica node is 5 which is less than 11, the original file owner adds the visit rate of the second replica node 4 to 5, getting 9. Because 9 is less than 11, the original file owner then adds the visit rate of the third replica node 3 to 9, getting 12. Because 12 is larger than 11, then the third replica node should be the replica node to redirect the file query. Algorithm 4 demonstrates the pseudocode executed by an original file owner for query redirection in *Plover*.

It was proved that scheduling by lottery is *probabilistically fair* [44]. The expected allocation of load to replica nodes is proportional to the number of tickets that they hold. The number of lotteries won by a client has a binomial distribution. The probability p that a client holding t tickets will win a given lottery with a total of T tickets is simply $p = t/T$. After n load assignment, the expected number of wins w is $E[w] = np$, with variance $\sigma_w^2 = np(1-p)$. The coefficient of variation for the observed proportion of wins is $\sigma_w^2 E[w] = \sqrt{(1-p)/np}$. Thus, a replica nodes load of the replica file is proportional to its ticket allocation, with accuracy that improves with $\sqrt{n}$.

4. Performance evaluation

We designed and implemented a simulator in Java for evaluation of the *Plover* based on Chord supernode structured P2P. There are mainly two classes for file replications: ID-based and path-based, and PAST [31] and LAR [15] are representative of each



(a) ts5k-large. (b) ts5k-small.

Fig. 4. CDF of total load distribution of replicated file.

Table 1
Simulated environment and algorithm parameters.

Parameter	Default value
System utilization	0.5–1
Object arrival location	Uniform over ID space
Number of nodes	4096
Node capacity	Bounded Pareto: shape 2 lower bound:25 000, upper bound: 25 000*10
Supernode threshold	50 000
Number of items	20 480
Existing item load	Bounded Pareto: shape: 2, lower bound: mean item actual load/2 upper bound: mean item actual load/2*10

class. We compared the performance of *Plover* with PAST, and LAR in Chord in terms of locality-aware file replication performance, load balance performance and file consistency maintenance cost. In the experiment, we set the number of replicas of a file equal to its visit rate in PAST. We assume LAR replicates a file along a path. Table 1 lists the parameters of the simulation and their default values. This bounded Pareto distribution reflects the real world where there are machines with capacities that vary by different orders of magnitude. We used *node utilization* to represent the fraction of its capacity that is used, which is L/C, and used *system utilization* to represent the fraction of the system's total capacity that is used, which equals to $\sum_{i=1}^{n} L_i / \sum_{i=1}^{n} C_i$.

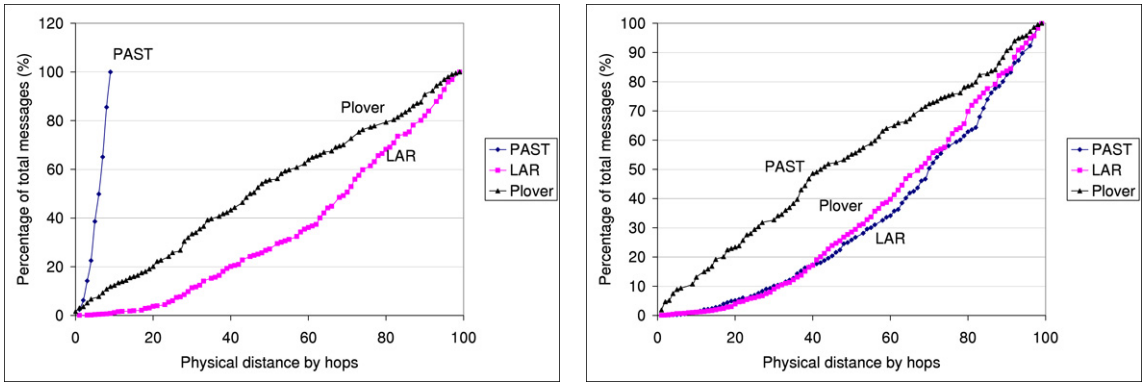
We used two transit-stub topologies generated by GT-ITM [48]: "ts5k-large" and "ts5k-small". "ts5k-large" has 5 transit domains, 3 transit nodes per transit domain, 5 stub domains attached to each transit node, and 60 nodes in each stub domain on average. "ts5k-small" has 120 transit domains, 5 transit nodes per transit domain, 4 stub domains attached to each transit node, and 2 nodes in each stub domain on average. "ts5k-large" has a larger backbone and sparser edge network (stub) than "ts5k-small". "ts5k-large" is used to represent a situation in which P2P consists of nodes from several big stub domains, while "ts5k-small" represents a situation in which P2P consists of nodes scattered in the entire Internet and only few nodes from the same edge network join the overlay. To account for the fact that interdomain routes have higher latency, each interdomain and intradomain hop counts as 3 and 1 hops of latency units respectively.

4.1. Locality-aware file replication

In this section, we will show *Plover*'s effectiveness in achieving locality-aware file replication between physically close nodes. Fig. 4(a) and (b) show the cumulative distribution function (CDF) of the total load of replicated files with system utilization approaching 1 in "ts5k-large" and "ts5k-small" respectively. We can see that in "ts5k-large," *Plover* is able to replicate 95% of the total load of replicated files, while LAR replicates about 30% and PAST replicates only about 20% within 10 hops. Almost all replications in *Plover* are within 15 hops, while LAR and PAST scheme replicate only 80% of the total file load within 15 hops. The results show that *Plover* replicates most files in short distances but LAR and PAST replicate most files at long distances. From Fig. 4(b), we can have the same observations as in "ts5k-large," although the performance difference between schemes is not so significant as in "ts5k-large". The more a file is replicated in a shorter distance, the higher the locality-aware performance of a file replication scheme. The results indicate that *Plover* performs better than LAR and PAST with regards to locality-aware file replication either when nodes are from several big sub domains or when nodes are scattered in the entire Internet.

Fig. 5(a) and (b) show the CDF of the distribution of total messages for file replication with system utilization approaching 1 in "ts5k-large" and "ts5k-small" respectively. We can see that in "ts5k-large", PAST restricts the physical distance by hops traveled by a message to 9, while the maximum physical distance of *Plover* and LAR is 99. In PAST, a file owner only needs to contact with its neighbor, which takes one logical hop. Thus, the maximum physical distance that a message travels is the maximum physical distance between two nodes. In contrast, in *Plover*, physically nodes report the information of their hot files and available capacity to the same supernode which is not necessarily physically close to them. A message of the information is routed to the supernode by DHT lookup algorithm, which takes a number of logical hops ($\log n$ in the average case). The physical distance a message travels is the sum of physical distances of all pairs of nodes in the lookup path. Therefore, in *Plover*, although files are replicated among physically close nodes, a report message needs to travel long physical distance. LAR replicates a file along a lookup path. Thus, the logical distance that a message travels is the number of hops in the lookup path. The physical distance is the sum of the physical distances of all pairs of nodes in the lookup path. As a result, as *Plover*, LAR leads to long physical distance for message traveling. We can also observe that the curve of *Plover* is above that of LAR, which means that most messages in *Plover* travel a shorter physical distance than in LAR. For instance, 43% of messages in total travel within 40 hops in *Plover*, while 20% of messages in total travel within 40 hops in LAR. *Plover* enables 64% of messages in total to travel within 60 hops, while LAR allows 36% of messages in total to travel within 60 hops.

From Fig. 5(b), we can have the same observations as in "ts5k-large". Compared to Fig. 5(a), we find that the maximum a message travels in PAST is 99 rather than 9 in "ts5k-large". In "ts5k-large", nodes are from several big stub domains, while in "ts5k-small", nodes are scattered in the entire Internet and only few nodes



(a) ts5k-large. (b) ts5k-small.

Fig. 5. CDF of the distribution of total messages.

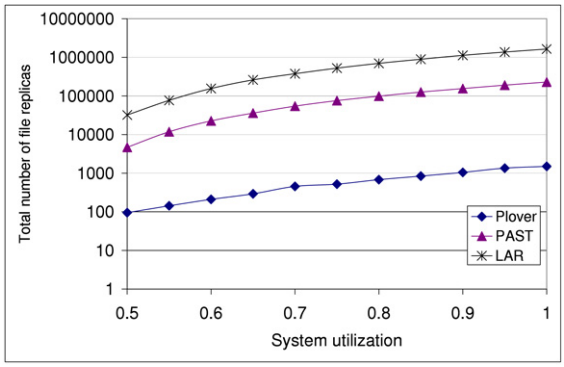


Fig. 6. Total replicas.

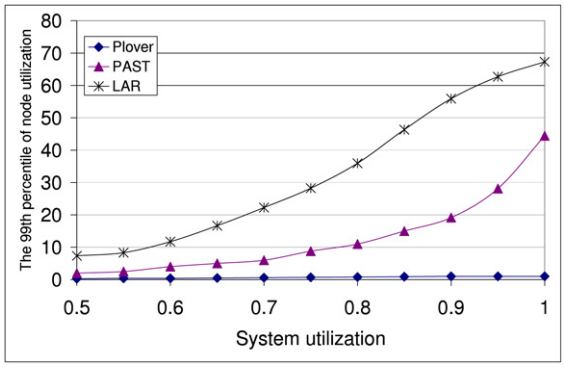


Fig. 7. Node utilization.

are from the same edge network. Therefore, the physical distance between a pair of arbitrary nodes is short in "ts5k-large", but is much longer in "ts5k-small". This is the reason that PAST generates a much longer physical distance for a message traveling in "ts5k-small" than in "ts5k-large". The curve of *Plover* is still above that of LAR, although the performance difference between the schemes is not so significant as in "ts5k-large". It implies that more messages travel in shorter physical distances in *Plover* than LAR. The results indicate that PAST enables messages to travel along short physical distances due to its one-hop replication distance. *Plover* performs better than LAR with regards to the overhead caused by message traveling.

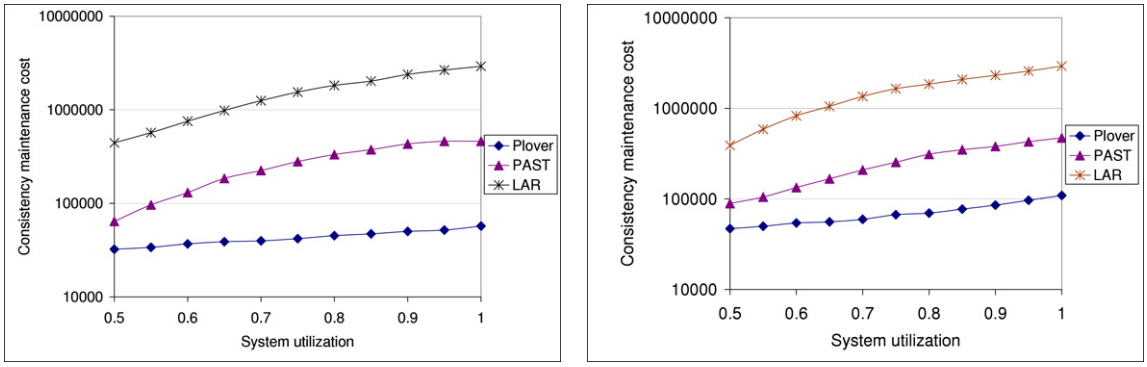
4.2. Capacity-aware file replication

Fig. 6 shows the total number of file replicas versus system utilization. We can observe that LAR generates dramatically more replicas than PAST, which produces much more replicas than *Plover*. We also measured the maximum node utilizations of all nodes, orders the values, and took the 99.9th percentile as the maximum 99.9th percentile node utilization. Fig. 7 plots the maximum 99.9th percentile node utilizations of different schemes. It illustrates that the utilization rate of LAR is higher than PAST, and that of PAST is higher than *Plover*. In addition, the rate of *Plover* is around 1. The results imply that LAR and PAST incur much more overloaded nodes, while *Plover* can keep nodes lightly loaded.

Recall that PAST and LAR do not consider nodes' available capacity during file replication. LAR makes replications along the lookup path once a node is overloaded, and PAST determines the number of replicas based on file availability. Determining the number of file replicas without considering nodes' available capacity will result in unnecessary file replication. For instance, if a hot file has 3 visit rate load and a node has available capacity to handle the 3 visit rate load, then one replica is enough and the file does not need to be replicated in three nodes. Though PAST uses load balancing afterwards, it will generate extra overhead. The neglect of nodes' available capacity in file replication leads to more replicas, more overloaded nodes, and extra overhead for load balancing and file consistency maintenance. In contrast, *Plover* proactively takes into account nodes' available capacity during file replication. It not only avoids unnecessary file replication, but also avoids exacerbating the hot spot problem by choosing nodes with enough 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.

4.3. Low-overhead file consistency maintenance

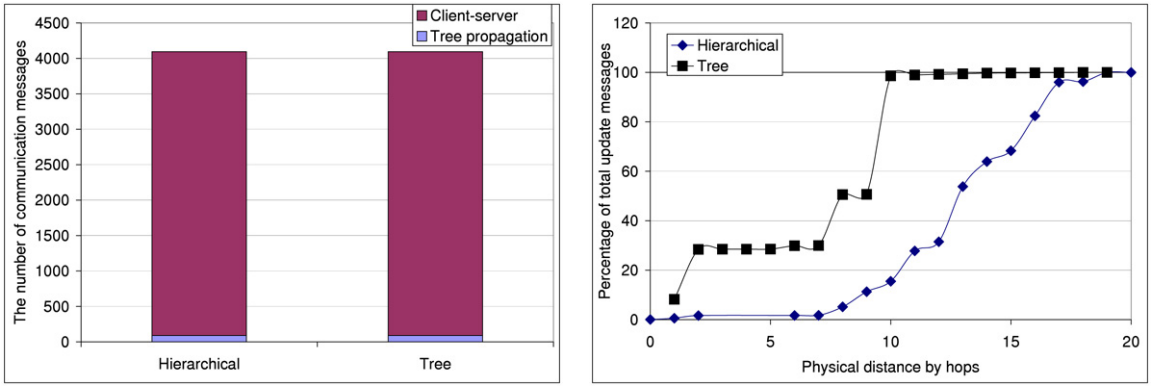
File consistency maintenance cost constitutes a major portion of structured P2P system overhead. The cost is directly related with message size and physical path length of the message traveled; we use the product of these two factors of all file update messages to represent the cost. It is assumed that the size of a update message is 1 unit. In the experiment, we updated every file once. To make results of file replication algorithm comparable, we used broadcasting for file update in all approaches. This experiment is to show the effect of file replication on the cost of file consistency maintenance caused by the number of replicas and physical distances for message propagation. Fig. 8(a) and (b) plot the file consistency maintenance cost of *Plover*, PAST and LAR in "ts5k-large" and "ts5k-small" respectively. From these figures, we can see that the cost increases with system load, LAR needs dramatically higher cost for file consistency maintenance than the others, and *Plover* incurs the least cost. There are two reasons for the results. First, LAR makes a replicate of each file along its lookup



(a) ts5k-large.

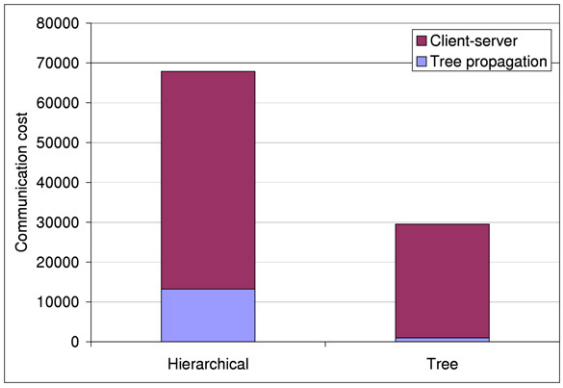
(b) ts5k-small.

Fig. 8. File consistency maintenance cost.



(a) The number of messages.

(b) CDF of message distribution.



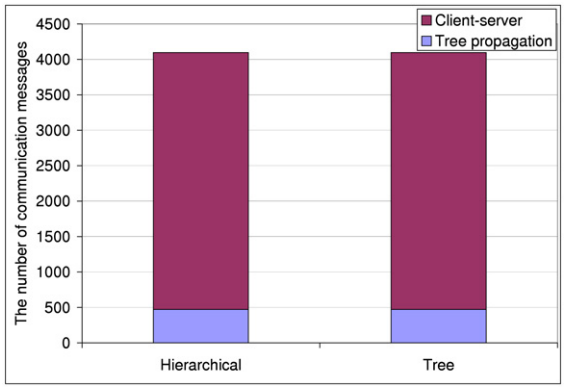
(c) Physical bandwidth.

Fig. 9. Performance of file consistency maintenance in "ts5k-large".

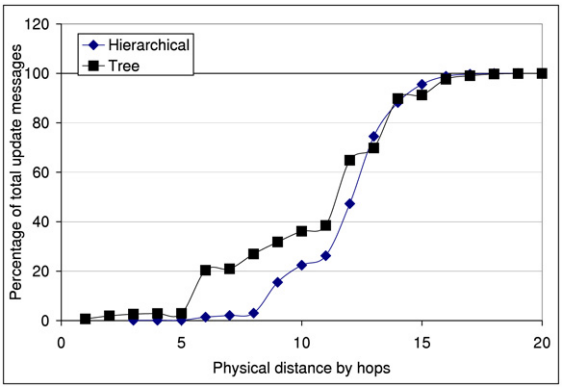
path length, while PAST replicates file based on its availability. With node capacity consideration, *Plover* generates fewer replicas. This result is consistent with Theorem 3.4. Second, because LAR and PAST neglect locality in file replication, they render a significantly higher cost for file update since messages travel long physical distances. In contrast, *Plover* proactively considers locality in file replication, such that the update messages only travel among physically close nodes. Its short physical distance for update messages and lower number of replicas result in low-overhead and timely file consistency maintenance.

The next experiment is to show the effectiveness and efficiency of the proposed consistency maintenance algorithm in *Plover*, denoted by *Tree*. We evaluated the performance of a consistency maintenance algorithm, denoted by *Hierarchical*. In *Hierarchical*, supernodes constitute a d-nary tree. Whenever a node updates its file, it notifies its supernode (client-server step). The supernodes propagate the update messages based on the d-nary tree. After a supernode receives the update message, it broadcasts the message to all of its children (tree propagation step). We evaluate the performance of *Tree* in comparison with *Hierarchical*.

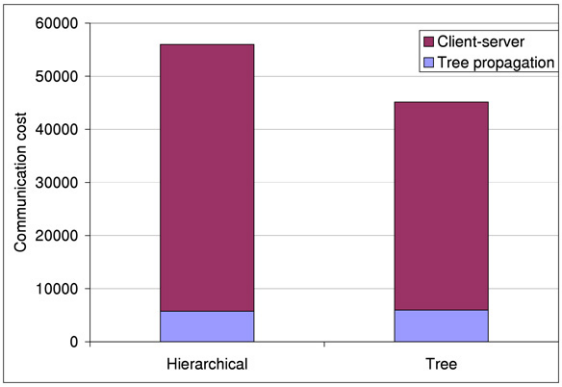
We assume that every node in the system has a replica of the updated file. Fig. 9(a) depicts the number of messages generated in the process of consistency maintenance in "ts5k-large". The number of messages is decomposed into that in the "client-sever" step in the upper level and in the "tree propagation" step in the lower level in the hierarchical structure. The figure shows that *Hierarchical* and *Tree* generate the same number of messages in each step. This is because in these two methods, the numbers of nodes in the upper level are the same and the numbers of nodes in the lower level are also the same. Therefore, they need the same



(a) The number of messages.



(b) CDF of message distribution.



(c) Physical bandwidth.

Fig. 10. Performance of file consistency maintenance in "ts5k-small".

number of messages for the message propagation in the upper-level tree and lower-level tree.

Fig. 9(b) plots the CDF of the distribution of update messages in "ts5k-large". We can see that *Tree* and *Hierarchical* limit the physical distance that a message travels within 20 hops. *Tree* enables 51% messages to travel within 8 hops, while *Hierarchical* only allows 5.1messages to travel within 8 hops. All messages travel within 12 hops in *Tree*, while only 30% messages travel within 12 hops in *Hierarchical*. Recall *Plover* replicates a file in nodes physically close to the file owner. *Tree* builds physically close replica nodes into a tree for update propagation. Therefore, update messages are propagated among physically close nodes in *Tree*. The supernodes are not necessarily physically close nodes, and a supernode also is not necessarily a physically close node to its clients. Hence, messages do not travel between physically close nodes in *Hierarchical*, leading to long physical distances of message traveling.

Fig. 9(c) shows the communication cost of all updates in "ts5k-large". *Tree* leads to a much lower communication cost than *Hierarchical*. Though the numbers of messages are the same in two algorithms, messages in *Tree* travel shorter distances than in *Hierarchical*. Therefore, the sum of the product of the size of a message and its traveled distance in *Tree* is less than that in *Hierarchical*.

Fig. 10(a) depicts the number of messages generated in the process of consistency maintenance in "ts5k-small". For the same reasons observed in Fig. 9(a), *Tree* leads to the same number of messages as *Hierarchical*. We can also find that *Hierarchical* produces much more messages in the "tree propagation" step in "ts5k-small" than in "ts5k-large". Because nodes are scattered in the entire Internet in "ts5k-small", while nodes are from several big stub domains in "ts5k-large", the information of hot files and available capacity is collected in more supernodes in "ts5k-small" than in "ts5k-large", resulting in more propagation messages between supernodes. Since every node should receive the update message once, the total numbers of messages in "ts5k-large" and "ts5k-small" are the same. Therefore, the number of messages in the "client-server" step in both *Hierarchical* and *Tree* in "ts5k-small" is less than that in "ts5k-large".

Fig. 10(b) plots the CDF of the distribution of update messages in "ts5k-small". We can have the same observations as in Fig. 9(b), although the performance difference between *Tree* and *Hierarchical* is not so significate due to different topologies. In "ts5k-small", the maximum physical distance of both *Tree* and *Hierarchical* is 20. *Tree* enables 27% messages to travel within 8 hops, while *Hierarchical* only allows 3% messages to travel within 8 hops. 65% of all messages travel within 12 hops in *Tree*, while only 47% messages travel within 12 hops in *Hierarchical*. These results confirm that *Tree* enables update messages to travel between physically close nodes, saving communication cost in consistency maintenance. Fig. 10(c) shows the communication cost of all updates in "ts5k-small". *Tree* leads to a much lower communication cost than *Hierarchical*. This is due to the same reason observed in Fig. 9(c).

5. Effectiveness of query redirection for load balance

This experiment tests the effectiveness of the query redirection algorithm based on lottery scheduling for load balance. We did an experiment on a file with 5 replica nodes. We tested the *Percent of queries received/Percent of responsible visit rate* (*percent rate* in short). The percent of responsible visit rate is the ratio between a replica node's responsible visit rate and the sum of visit rates of all replica nodes. The percent of queries received is the ratio between the number of queries received by a replica node and the

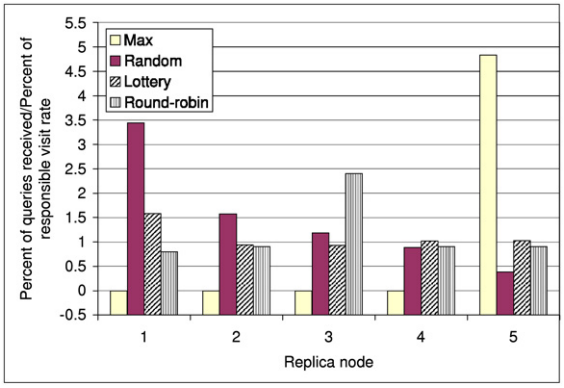


Fig. 11. Effectiveness of query redirection for load balance.

total number of queries. If the metric is close to 1, it means the query redirection algorithm can distribute file queries according to node responsible visit rates, achieving load balance. We tested the metric of the lottery scheduling based query redirection algorithm, denoted by *Lottery*. We also tested the metric of *Max*, *Random* and *Round-robin*. In *Max*, a file owner always forwards file queries to the replica node with the highest responsible visit rate. In *Random*, a file owner randomly chooses a replica node among all replica nodes to redirect a file query. In *Round-robin*, a file owner redirects file queries in a round-robin manner.

Fig. 11 demonstrates the percent rates of each replica node in all consistency maintenance algorithms. We can observe that in *Max*, one node receives all file queries and all other replica nodes do not receive any queries. This node is the replica node with the highest responsible visit rate. Obviously, *Max* is not able to achieve load balance since other replica nodes do not receive queries, while the node with the highest responsible visit rate may be overloaded. The figure shows that *Random* generates a percent rate within [0.4, 3.4]. It means that some nodes receive much more queries than they are responsible for, while some other nodes receive less queries than what they are responsible for. *Random* also cannot achieve a very good load balance. In *Round-robin*, one node has about a 2.4 percent rate, all other nodes' percent rates are round 1. *Round-robin* leads to a better load balance than *Max* and *Random*. In *Lottery*, except one node which has a 1.5 percent rate, all other nodes' percent rates are around 1. The results imply that *Lottery* can achieve a better load balance than other algorithms. Thus, *Lottery* outperforms other algorithms in distributing file queries among replica nodes in balance.

6. Conclusions

File replication and file consistency maintenance are indispensable parts for high performance in structured P2P file systems. Currently, these two issues are typically addressed separately, despite significant interdependency of file consistency maintenance on file replication. A growing need persists with regards to integrating file replication and consistency maintenance techniques for high performance. This paper presents a proactive low-overhead file replication scheme called *Plover* for structured P2P file sharing systems. Unlike existing file replication methods, by taking into account nodes' available capacity and physical locality, *Plover* not only achieves highly efficient file replication, but also proactively facilitates efficient file consistency maintenance. It makes file replication among physically close nodes based on nodes' available capacity. Thus, it reduces file consistency maintenance overhead due to short communication distances and fewer file replicas. It also includes an efficient file query redirection algorithm for load balance in replica nodes. Theoretical analysis and simulation results demonstrate the effectiveness of *Plover* in comparison with other file replication schemes.

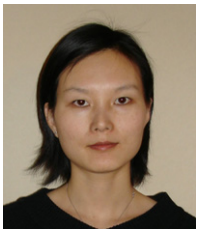
Acknowledgments

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

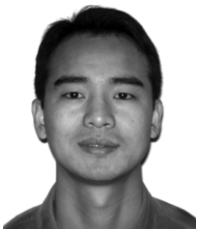
References

[1] A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer, R.P. Wattenhofer, Farsite: Federated, available, and reliable storage for an incompletely trusted environment, in: Proc. of Symposium on Operating Systems Design and Implementation, 2002.
[2] X. Chen, S. Ren, H. Wang, X. Zhang, SCOPE: Scalable consistency maintenance in structured P2P systems, in: Proc. of INFOCOM, 2005.
[3] I. Clarke, O. Sandberg, B. Wiley, 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, 2001, pp. 46–66.
[4] F. Cl(̂e)venot, P. Nain, A simple model for the analysis of the squirrel peer-to-peer caching system, in: Proc. of IEEE INFOCOM, 2004.
[5] E. Cohen, S. Shenker, Replication strategies in unstructured peer-to-peer networks, in: Proc. of ACM SIGCOMM, 2002.
[6] F.M. Cuenca-Acuna, R.P. Martin, T.D. Nguyen, Autonomous replication for high availability in unstructured P2P systems, in: Proc. of IEEE SRDS, 2003.
[7] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, I. Stocia, Wide-area cooperative storage with CFS, in: Proc. of SOSP, 2001.
[8] A. Datta, M. Hauswirth, K. Aberer, Updates in highly unreliable, replicated peer-to-peer systems, in: Proc. of ICDCS, 2003.
[9] J.R. Douceur, A. Adya, W.J. Bolosky, D. Simon, M. Theimer, Reclaiming space from duplicate files in a serverless distributed file system, in: Proc. of International Conference on Distributed Computing Systems, 2002.
[10] J.R. Douceur, R.P. Wattenhofer, Optimizing file availability in a secure serverless distributed file system, in: Proc. of IEEE Symposium on Reliable Distributed Systems, 2001, pp. 4–13.
[11] F. Fessant, S. Handurukande, A. Kermarrec, L. Massoulie, Clustering in peer-to-peer file sharing workloads, in: Proc. of IPDPS, 2004.
[12] Marcus Fontoura, Law-governed peer-to-peer auctions, in: In Proc. of the Eleventh International World Wide Web Conference, WWW, 2002, pp. 109–117.
[13] B. Gedik, L. Liu, Reliable peer-to-peer information monitoring through replication, in: Proc. of IEEE SRDS, 2003.
[14] A. Ghodsi, L. Alima, S. Haridi, Symmetric replication for structured peer-to-peer systems, in: Proc. of International Workshop on Databases, Information Systems and Peer-to-Peer Computing, 2005, p. 12.
[15] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, P. Keleher, Adaptive replication in peer-to-peer systems, in: Proc. of ICDCS, 2004.
[16] K. Huang, T. Huang, J. Chou, LessLog: A logless file replication algorithm for peer-to-peer distributed systems, in: Proc. of IPDPS, 2004.
[17] J. Kistler, M. Satyanarayanan, Disconnected operation in the Coda file system, ACM Transactions on Computer Systems (1) (1992).
[18] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, OceanStore: An architecture for global-scale persistent storage, in: Proc. of the ASPLOS, 2000.
[19] J. Lan, X. Liu, P. Shenoy, K. Ramamritham, Consistency maintenance in peer-to-peer file sharing networks, in: Proc. the IEEE Workshop on Internet Applications, WIAPP, 2003.
[20] N. Laoutaris, V. Zissimopoulos, I. Stavrakakis, On the optimization of storage capacity allocation for content distribution, Computer Networks (3) (2005).
[21] Z. Li, G. Xie, Z. Li, Locality-aware consistency maintenance for heterogeneous P2P systems, in: Proc. of IPDPS, 2007.
[22] Z. Li, G. Xie, Z. Li, Efficient and scalable consistency maintenance for heterogeneous peer-to-peer systems, TPDS (2008).
[23] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured peer-to-peer networks, in: Proc. of ACM International Conference on Supercomputing, ICS, 2001.
[24] A. Muthitacharoen, R. Morris, T.M. Gil, B. Chen, Ivy: A read/write peer-to-peer file system, in: Proc. of OSDI, 2002.
[25] M. Nelson, B. Welch, J. Ousterhout, Caching in the sprite network file system, IEEE/ACM Transactions on Networking (1) (1988).
[26] E. Ogston, S. Vassiliadis, A peer-to-peer agent auction, in: Proceeding of ACM International Conference on Autonomous Agents and Multiagent Systems, AAMAS, 2002.
[27] P2P traffic is booming, BitTorrent The Dominant Protocol. http://torrentfreak.com/p2p-traffic-still-booming-071128/.
[28] T. Pitoura, N. Ntarmos, P. Triantafillou, Replication, load balancing and efficient range query processing in DHTs, in: Proc. of EDBT, 2006.
[29] J. Kangasharju, K. Ross, D. Turner, Optimizing file availability in peer-to-peer content distribution, in: Proc. of INFOCOM, 2007, pp. 1973–1981.
[30] M. Roussopoulos, M. Baker, CUP: Controlled update propagation in peer to peer networks, in: Proc. of the USENIX 2003 Annual Technical Conf., 2003.
[31] A. Rowstron, P. Druschel, Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, in: Proc. of SOSP, 2001.
[32] Y. Saito, C. Karamanolis, M. Karlsson, M. Mahalingam, Taming aggressive replication in the pangaea wide-area file system, in: Proc. of the 5th USENIX Symposium on Operating Systems Design and Implementation, December 2002.

[33] H. Sandhu, S. Zhou, Cluster-based file replication in large-scale distributed systems, in: Proc. of ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, 1992, pp. 91–102.
[34] H. Shen, C.-Z. Xu, Hash-based proximity clustering for efficient load balancing in heterogeneous DHT networks, Journal of Parallel and Distributed Computing (JPDC) (2008); An early version appeared in IPDPS 2006.
[35] H. Shen, C. Xu, Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured Peer-to-Peer Networks, IEEE Transactions on Parallel and Distributed Systems (2007).
[36] T. Stading, P. Maniatis, M. Baker, Peer-to-peer caching schemes to address flash crowds, in: Proc. of IPTPS, 2002.
[37] D. Starobinski, D. Tse, Probabilistic methods for web caching, Performance Evaluation (1) (2003).
[38] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Transactions on Networking 1 (1) (2003) 17–32.
[39] S. Tewari, L. Kleinrock, Analysis of search and replication in unstructured peer-to-peer networks, in: Proc. of ACM SIGMETRICS, 2005.
[40] S. Tewari, L. Kleinrock, On fairness, optimal download performance and proportional replication in peer-to-peer networks, in: Proc. of IFIP Networking, 2005.
[41] S. Tewari, L. Kleinrock, Proportional replication in peer-to-peer network, in: Proc. of INFOCOM, 2006.
[42] Tim Hsin ting Hu, Sebastien Ardon, Aruna Sereviratne, Semantic-laden peer-to-peer service directory, in: The Proceedings of the 4th IEEE International Conference on P2P Computing, P2P, 2004.
[43] D. Tsoumakos, N. Roussopoulos, APRE: An adaptive replication scheme for unstructured overlays, in: Proc. of CoopIS, 2006.
[44] C. Waldspurger, W. Weihl, Lottery scheduling: Flexible proportional-share resource management, in: Proc. of OSDI, 1994, pp. 1–11.
[45] Z. Xu, M. Mahalingam, M. Karlsson, Turning heterogeneity into an advantage in overlay routing, in: Proc. of INFOCOM, 2003.
[46] L. Yin, G. Cao, DUP: Dynamic-tree based update propagation in peer-to-peer networks, in: Proc. of ICDE, 2005.
[47] H. Yu, A. Vahdat, Consistent and automatic replica regeneration, IEEE/ACM Transactions on Networking (1) (2005).
[48] E. Zegura, K. Calvert, S. Bhattacharjee, How to model an internetwork, in: Proc. of INFOCOM, 1996.
[49] J. Zhu, J. Gong, W. Liu, T. Song, J. Zhang, A collaborative virtual geographic environment based on P2P and Grid technologies, Information Sciences: An International Journal Archive (21) (2007).



Haiying Shen received the B.S. degree in Computer Science and Engineering from Tongji University, China in 2000, and the M.S. and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Assistant Professor in the Department of Computer Science and Computer Engineering of the University of Arkansas. Her research interests include distributed and parallel computer systems and computer networks, with an emphasis on peer-to-peer and content delivery networks, wireless networks, resource management in cluster and grid computing, and data searching. She is a PC member of many conferences and a member of IEEE and ACM.



Yingwu Zhu received his Ph.D. degree in Computer Science & Engineering from the University of Cincinnati in 2005. He obtained his BS and MS degrees in Computer Science from the Huazhong University of Science and Technology, China, in 1994 and 1997, respectively. He is an assistant professor of Computer Science and Software Engineering at the Seattle University. His research interests include operating systems, file and storage systems, peer-to-peer systems, distributed systems, and computer networks.