Plover: A Proactive Low-overhead File Replication Scheme for Structured P2P Systems

Haiying Shen
Dept. of Computer Science & Computer Engineering
University of Arkansas, Fayetteville, AR 72701
hshen@uark.edu

Yingwu Zhu
Dept. of Computer Science & Software Engineering
Seattle University, Seattle, WA 98122
zhuy@seattleu.edu

***Abstract*—File replication is a widely used technique for high performance in peer-to-peer file sharing systems. A file replication technique should be efficient and meanwhile facilitates efficient file consistency maintenance. However, most traditional methods don't consider node 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 replication among physically close nodes based on node 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. 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.**

I. Introduction

Over the past years, the immerse popularity of the Internet has produced a significant stimulus to peer-to-peer (P2P) file sharing systems. A recent study [1] shows that more than 75% 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 [2]. The median object size of these P2P systems is 4MB which represents a thousand-fold increase over the 4KB median size of typical web objects. 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, even under a low price of storage. 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 systems. Without effective replica consistency maintenance, a P2P system 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, online auction, and remote collaboration. 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 systems determine replica nodes based on node IDs [3–6] or query path [7–10]. 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 concentrate on flash crowds elimination and query efficiency by assuming 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 close nodes based on node available capacities. With node 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 node 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 lottery scheduling method to achieve file query load balance between replica nodes.

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

II. Related Work

Driven by tremendous advances of P2P file sharing systems, numerous file replication methods have been proposed for structured P2P systems. One group of file replication methods determines replica node based on IDs [3–6]. In PAST [3], 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 load balancing algorithm for non-uniform storage node capacities and file sizes. CFS [4] stores blocks of a file and spreads blocks evenly over the available servers to prevent large files from causing unbalanced use of storage. It uses distributed hash function to replicate each block on servers immediately after the block's successor on the Chord ring in order to increase availability. LessLog [5] determines the replicated nodes by constructing a lookup tree based on IDs to determine the location of the replicated node. In HotRoD [6], hot arcs of peers are replicated and rotated over the identifier space.

Another group of file replication methods [7–10] chooses replica nodes based on file query path. Stading *et al.* [7] proposed Backslash system, in which a node pushes cache to one hop closer to requester nodes when overloaded. LAR [8] let overloaded nodes replicate at the query initiator and create routing hints on the reverse path. CUP [9] and DUP [10] cache metadata along the lookup path with consistency support.

The rigid ID-based or path-based replica node determination 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 load balancing algorithm, and CFS adopts file division, these strategies come at a price of extra overhead and complexity. In addition to the node available capacity, these file replication schemes don't 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 node 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.

III. Plover: 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 highly capacity and fast connections. For simplicity, we define a node with capacity greater than a predefined threshold as supernode; otherwise a regular node. *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 [12]. 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 use the vector of distances $< d_1, d_2, \ldots, d_m >$ as its coordinate in Cartesian space. Two physically close nodes will have similar landmark vectors. Hilbert curve [12] 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 *Hilbert number* of the node.

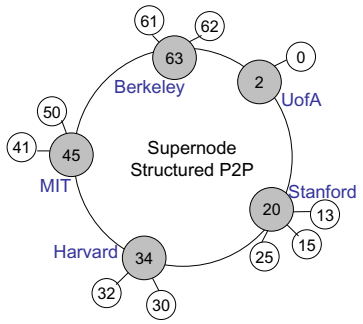

Fig. 1. Supernode structured P2P.

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 [13] and Cycloid [14]. With 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; that is, regular nodes are connected to their physically closest supernode since node ID represents node physical location closeness. 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 backups. 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 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.

Figure 1 shows an example of 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 *n*40 wants to join in the *pCluster*, *n*40 asks its known node *n*2 to find the supernode with ID closest to 40, which is *n*45. If *n*40 is a supernode, *n*45 moves *n*41 to *n*40. The maintenance of supernode network is the same as that of Chord. If *n*40 is a regular node, it becomes a client of *n*45. If a node, say *n*45, wants to leave the system, it moves *n*41 to *n*34, and *n*50 to *n*63. If *n*41 wants to leave the *pCluster*, it only need to disconnect its link to *n*45.

Plover replies on the supernode network for locality-aware file replication. Specifically, each overhead node reports the information of its hot files, and each lightly loaded node reports its *available capacity* $C - L$ to its supernode periodically, where C is a node's capacity and L is its actual load. As a result, the information of physically close nodes gather together in the supernode. The supernode conducts node mapping with node capacity consideration for file replication, and notify overloaded nodes to replicate files to lightly loaded nodes.

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 hot spot problem. In addition, nodes with higher available capacity have 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 transferred during a time unit. 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.

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. File replication may generate load imbalance among nodes. The load balancing algorithm in [15] is adopted into *Plover* to handle the load imbalance problem. Due to space limit, please refer to [15] for details.

TABLE I
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:25000, upper bound: 25000*10
Supernode threshold	50000
Number of items	20480
Existing item load	Bounded Pareto: shape: 2, lower bound: mean item actual load/2 upper bound: mean item actual load/2*10

File consistency maintenance. *Plover* eliminates the need for a specific file consistency maintenance method such as [9, 10, 16, 17] 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 structure maintenance costs overhead. On the other hand, push/pull or flooding method doesn't 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 the number of replica nodes of a file is larger than T, supernode builds a tree structure consisting of the replica nodes and it functions as the tree root. For a file update, *Plover* lets each node send its update message to its supernode, which will further broadcast the message to the file replica nodes if there is no tree structure for the file. Otherwise, supernode forwards the message to its children in the tree, and the message will be propagated downwards along the tree. *Plover* facilitates low-overhead and timely consistency maintenance as update messages travel along short physical distances.

Query redirection for load balance. If an overloaded node 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 node available capacity. An efficient query redirection algorithm should effectively allocate query load to replica nodes in balance. *Plover* adopts lottery scheduling [18] for query redirection. Lottery scheduling is a method that efficiently implements proportional-share resource management. In the scheduling, nodes has tickets, and its allocated resource is proportional to the number of tickets that they hold. *Plover* regards the visit rate of a file as tickets. The number of tickets a replica node holds equals to 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

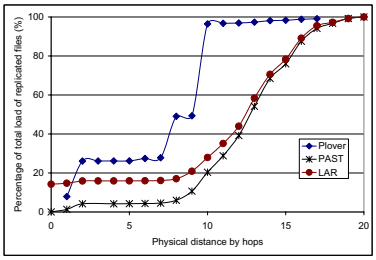

Fig. 2. CDF of total load distribution of replicated file.

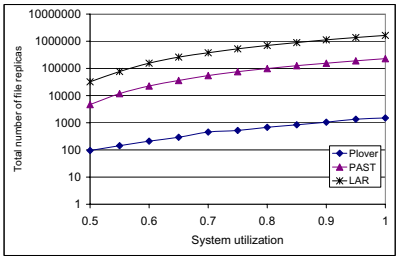

Fig. 3. Total replicas.

generated. The visit rate ratios among replica nodes is the expected ratios of load that are responsible for.

IV. 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 [3] and LAR [8] are representative of each 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 equals to its visit rate in PAST. Table I lists the parameters of the simulation and their default values. 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 a transit-stub topology generated by GT-ITM [19]: "ts5k-large". "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.

A. Locality-aware File Replication

In this section, we will show the effectiveness of *Plover* to achieve locality-aware file replication between physically close nodes. Figure 2 shows the cumulative distribution function (CDF) of total load of replicated files with system utilization approaches to 1 in "ts5k-large". We can see that *Plover* is able to replicate 95% of 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 file in long distances. The more file replicated in the shorter distance, the higher locality-aware performance of a file replication scheme. The results indicate that *Plover* performs superiorly 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.

B. Capacity-aware File Replication

Figure 3 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*. Figure 4 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 don't consider node 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 node available capacity consideration 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 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, *Plover* proactively takes into account node 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.

C. 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 travelled; 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. Figure 5 plots the file consistency maintenance cost of *Plover*, PAST and LAR in "ts5k-large". 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 replication of each file along its lookup path length, while PAST replicates file based on its availability. With node capacity consideration, *Plover* generates less replicas. Second, because LAR and PAST neglect locality in file replication, they

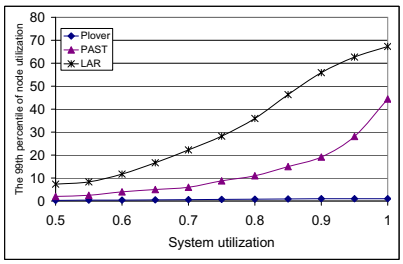

Fig. 4. Node utilization.

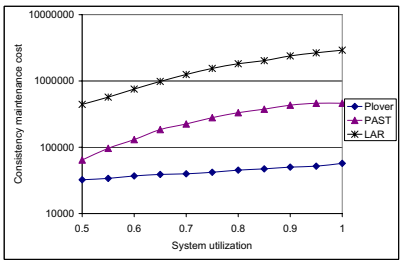

Fig. 5. File consistency maintenance cost.

render significantly high 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 less number of replicas result in low-overhead and timely file consistency maintenance.

V. Conclusions

File replication and file consistency maintenance are indispensable parts 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 the two techniques for high performance. This paper presents a proactive low-overhead file replication scheme called *Plover*. Unlike existing file replication methods, by taking node available capacity and physical locality into account, *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 node available capacity. Thus, it reduces file consistency maintenance overhead due to short communication distances and less file replicas. It also includes an efficient file query redirection algorithm for load balance in replica nodes. Simulation results demonstrate the effectiveness of *Plover* in comparison with other file replication schemes. We plan to explore the methods that help to fully exploit file replication for efficient lookups.

Acknowledgment

This research was supported in part by U.S. Acxiom Corporation.

References

[1] S. Saroiu and et al. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Pro. of MMCN*, 2002.

[2] M. Chesire, A. Wolman, and et al. Measurement and Analysis of a Streaming-Media Workload. In *Proc. of USITS*, 2001.

[3] 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.

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

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

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

[7] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer Caching Schemes to Address Flash Crowds. In *Proc. of IPTPS*, 2002.

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

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

[10] L. Yin and G. Cao. DUP: Dynamic-tree Based Update Propagation in Peer-to-Peer Networks. In *Proc. of ICDE*, 2005.

[11] F. Fessant, S. Handurukande, and et al. Clustering in Peer-to-Peer File Sharing Workloads. In *Proc. of IPDPS*, 2004.

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

[13] I. Stoica, R. Morris, D. Liben-Nowell, and et al. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *TON*, 1(1):17–32, 2003.

[14] H. Shen, C. Xu, and G. Chen. Cycloid: A Scalable Constant-Degree P2P Overlay Network. *Performance Evaluation*, 63(3):195–216, 2006.

[15] H. Shen and C. Xu. Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured Peer-to-Peer Networks. *TPDS*, 2007.

[16] Z. Li, G. Xie, and Z. Li. Locality-Aware Consistency Maintenance for Heterogeneous P2P Systems. In *Proc. of IPDPS*, 2007.

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

[18] C. Waldspurger and W. Weihl. Lottery Scheduling: Flexible Proportional-Share Resource Management. In *Proc. of OSDI*, pages 1–11, 1994.

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