LSA: Lightweight and Self-Adaptive File Replication in Distributed P2P File Sharing Systems

Haiying Shen
Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, AR 72701

Abstract - *In peer-to-peer file sharing systems, file replication technology is widely used to reduce hot spots and improve file query efficiency. Most current file replication methods replicate files in all nodes on a client-server query path. However, these methods come at a cost of high overhead. This paper presents a Lightweight and Self-Adaptive file replication algorithm (LSA) that achieves high query efficiency and effectiveness at a significantly low overhead. LSA enhances the utilization of file replicas by selecting query traffic hubs and frequent requesters as replica nodes. LSA creates and deletes replicas in a decentralized self-adaptive manner while guarantees high replica utilization. Simulation results demonstrate the effectiveness of LSA in comparison with another approach. It dramatically reduces the overhead of file replication, and yields significant improvements on the effectiveness of file replication. overhead.*

Keywords: Distributed system, File replication, Peer to peer, File sharing system

1 Introduction

Over the past years, the immense popularity of Internet has produced a significant stimulus to peer-to-peer (P2P) file sharing systems, where a file requester's query will be forwarded to a file provider in a distributed manner. P2P file sharing systems can be used in video-on-demand service and shared digital library applications where individuals dedicate files which are available to others. Currently, the median file size of these P2P systems is very large, and the access to these files is highly repetitive and skewed towards the most popular ones. In such circumstances, if a server receives many requests at a time, it could become overloaded and consequently cannot reply the requests in a timely fashion. Therefore, highly-popular files (i.e. hot files) can exhaust the bandwidth capacity of the servers, and lead to low file sharing efficiency.

File replication is an effective method to deal with the problem of server overload by distributing load over replica nodes. It helps to achieve high query efficiency by reducing server response latency and lookup path length (i.e. the number of hops in a lookup path). A replica *hit* occurs when a file request is resolved by a replica node rather than a file owner. *Replica hit rate* is the percentage of the number of file queries that are resolved by replica nodes among total queries. Higher hit rate means higher effectiveness of a file replication method.

Recently, numerous file replication methods have been proposed [1, 2, 3, 4, 5, 6, 7, 8, 9], and most methods replicate or cache a file on the nodes along the query path from a requester to a file owner. However, though these methods can improve query efficiency but come at a cost of high overhead. We use *path* to denote these methods. Since more replicas lead to higher query efficiency and vice versa, a challenge for a replication algorithm is how to minimize replicas and still achieve high query efficiency. To deal with this challenge, this paper presents a Lightweight and Self-Adaptive file replication algorithm (LSA) that achieves high query efficiency at a significantly low cost. A significant feature of LSA is that it achieves an optimized tradeoff between query efficiency and overhead of file replication. Instead of creating replicas on all nodes on a client-server path, LSA chooses query traffic hubs (i.e. query traffic conjunction nodes) as replica nodes to ensure high replica hit rate. It achieves comparable query efficiency to *Path* but with much less replicas.

Another feature of LSA is that it adaptively adjusts file replicas to non-uniform and time-varying file popularity and node interest in a decentralized self-adaptive manner. Unlike other algorithms in which a file owner determines where to create or delete replicas in a centralized fashion, LSA lets nodes themselves decide whether to store or delete replicas based on the query traffic.

LSA is independent of P2P structure. We intend our results to be applicable to both structured P2P overlays [10, 11, 12, 13, 14, 15] and unstructured P2P overlays [6, 16, 17, 18].

The rest of this paper is structured as follows. Section 2 presents a concise review of representative file replication approaches for P2P systems. Section 3 presents the LSA file replication algorithm. Section 4 shows the performance of LSA in comparison with another approach with a variety of metrics, and analyzes the factors effecting file replication performance. Section 5 concludes this paper.

2 Related Work

Recently, numerous file replication methods have been proposed. PAST [1] replicates each file on a set number of nodes whose IDs match most closely to the file owner's ID. It uses file caching along the lookup path to minimize query latency and balance query load. Similarly, CFS [2] replicates blocks of a file on nodes immediately after the block's owner. It also caches a file location hint along a path to improve query efficiency. Stading *et al.* [3] also proposed to replicate a file in locality close nodes near the file owner. Overlook [4] places a replica of a file on a node with most incoming lookup requests for fast replica location. It needs to keep track of client-access history to decide the replica nodes.

Gnutella [6] replicates files in overloaded nodes at the file requesters. In LAR [5], a node makes decisions of file replication based on its load. LAR specifies the overloaded degree of a server that a file should be replicated. In addition to replicating a file at the requester, it also has file location hint along the lookup path. Backslash [3] pushes cache to one hop closer to requester nodes as soon as nodes are overloaded.

Freenet [19] replicates files both on insertion and retrieval on the path from the requester to the target. PAST [1], CFS [2], LAR [5], CUP [7] and DUP [8] perform caching along the query path. Cox *et al.* [9] studied providing DNS service over a P2P network. They cache index entries, which are DNS mappings, along query paths.

LSA replicates a file at nodes in a path with high query load of the file, which improves replica hit rate and query efficiency and meanwhile reduces replicas. Unlike most of these methods, in which a file owner needs to keep track of replica nodes for replica creation and removal, LSA conducts the replica adjustment in a decentralized manner.

There are other studies for file replication in unstructured P2Ps [20, 21, 22, 23]. These works study the system performance such as successful queries and bandwidth consumption when the number of replicas of a file is proportional, is uniform and is square-root proportional of the query rate. These works focused on the relationship between the number of replicas, file search time and load balance, but they did not investigate the impact of replica location on file query efficiency. LSA mainly studies where to replicate files to achieve comparably query efficiency with less replicas.

Splitting a large file into small pieces can increase the service capacity of a large file rapidly. Replicating file location hint along query path can further improve file query efficiency. LAR can employ the techniques to further improve its performance. These techniques are orthogonal to our study in this paper.

3 LSA File Replication Algorithm

3.1 Lightweight File Replication

In P2P file sharing systems, some nodes carry more query traffic load while others carry less [24, 25]. There will be more query traffic along the query paths from the frequent file requesters and the file owners. Based on this observation, we can choose query traffic hubs or frequent requesters as replica nodes, so that the queries from different direction can encounter the replica nodes, increasing the hit rate.

We define *query rate* of a file f, denoted by q_f, as the number of queries initiated by a requester or forwarded by a node during unit time period T. LSA sets a threshold for query rate, T_q. If a node's $q_f > T_q$, it is a frequent requester or traffic hub for file f. When a node initiates or forwards a file request, it checks its q_f. If $q_f > T_q$, it incorporates a file replication request and its q_f into the file query.

In addition to the original owner of a file, a replica node can also replicate the file. We use *server* to denote both the original file owner and replica nodes. We use a server's visit rate to represent its query load denoted by l. We use c to denote a node's capacity represented by the number of queries it can respond during T. We use *node utilization* to denote the fraction of a node capacity that is used, represented by l/c. Each server i records its query load l_i over T periodically, and checks whether it is overloaded.

When overloaded, a server firstly orders the replication requesters based on their q_f in descending order. Then, it retrieves requesters in the list one at a time, and replicates f at the requester until the server is lightly loaded. This scheme guarantees that nodes with higher query rates have higher priorities to be replica nodes, which lead to higher replica hit rate.

Table 1. Simulated environment and parameters.

Parameter	Default value
File distributon	Uniform over ID space
Number of nodes	4096
Node capacity c	Bounded Pareto: shape 2 lower bound: 500 upper bound: 50000
Number of queried files	50
Number of queries per file	1000
Number of replication operations	5-25
T_q	5
T	1 second

3.2 Self-Adaptive File Replica Adjustment

In previous methods, a file server maintains information of its replica nodes to manage the replicas and disseminates information about new replica sets. Rather than depending on such a centralized method, LSA makes replica adjustment in a decentralized self-adaptive manner.

Typically, LSA lets each node periodically update its query rate of each file. If a node's $q_f > T_q$, it requests to have a replica as introduced in the previous section. If a replica node's $q_f < \delta T_q$, where δ is a under-loaded factor, which means the utilization of the replica is low, the replica node records the replica as infrequently-used replica. When the $q_f < \delta T_q$ condition occurs for a specified number of time periods, or when the node needs more space for other replicas, the node removes the replica. When a file is no longer requested frequently, there will be less file replicas for it. The adaptation to query rate ensures that all file replicas are worthwhile and there is no waste of overhead for unnecessary replica maintenance, thus ensuring high replica utilization. Due to space constraints, for more details of LSA, please see [26].

4 Performance Evaluation

We designed and implemented a simulator for evaluating the LSA algorithm based on Chord P2P system [10]. We use *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 compared the performance of LSA with *Path*. Experiment results show that LSA achieves high file query efficiency and high hit rate with less file replicas. To be comparable, we used the same number of replication operations when a server is overloaded in all replication algorithms. In a replication operation, the server chooses all nodes in a lookup path in *Path* to replicate a file. As a result, LSA replicates a file to a single node while *Path* replicates a file to a number of nodes in one replication operation.

We assumed bounded Pareto distribution for node capacities. This distribution reflects the real world where there are machines with capacities that vary by different orders of magnitude. The file requesters and requested files in the experiment were randomly chosen. File lookups were generated according to a Poisson process at a rate of one per second as in [10]. Table 1 lists the parameters of the simulation and their default values.

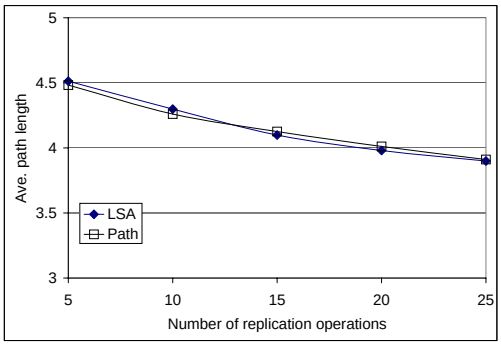

Figure 1. Average path length .

4.1 Effectiveness of Replication Algorithm

One ultimate goal of a file replication algorithm is to improve the file lookup efficiency. With an effective file replication algorithm, a file request can meet a replica node earlier in routing, leading to shorter lookup path length. This experiment is to show the effectiveness of LSA in reducing file lookup path length. Figure 1 plots the average lookup path length of different approaches versus the number of replication operations when a server is overloaded. We can see that LSA leads to approximately the same path length as *Path*. Unlike LSA that replicates a file only in one node in each file replication, *Path* replicates in nodes along a query path. Therefore, it increases replica hit rate and produces shorter path length. LSA can achieve almost the same lookup efficiency with much less replicas, since it replicates a file only in one node every time. This result implies the effectiveness of LSA to replicate files in nodes with high query rate including traffic hubs and frequent requesters, which enhances the utilization of replicas and hence reduces the lookup path length. LSA is almost as

effective as *Path* in improving file lookups, but generates much less overhead in file replication than *Path*.

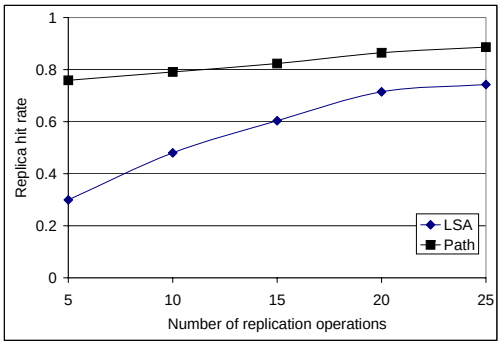

Figure 2. Replica hit rate.

An effective file replication algorithm should enable a file requester to have a high probability to encounter a replica node. That is, it should lead to a high replica hit rate. Figure 2 demonstrates the replica hit rate of different file replication approaches. We can observe that *Path* leads to higher hit rate than LSA. *Path* replicates files at nodes along the routing path. More replica nodes render higher possibility for a file requester of meeting a replica node. However, its efficiency is outweighed by its prohibitively cost of overhead for keeping track of query paths and maintaining much more file replicas. Though LSA leads to slightly lower replica hit rate than *Path*, it achieves comparable performance in improving lookup efficiency as shown in Figure 1.

4.2 Overhead of Replication Algorithm

Figure 3 illustrates the total number of replicas in different approaches. It shows that the number of replicas increases as the number of replication operations increases. This is due to the reason that more replication operations for a file produce more replicas. The number of replicas of *Path* is excessively higher than LSA. It is because in each file replication operation, a file is replicated in a single node in LSA, but in multiple nodes along a routing path in *Path*. Therefore, *Path* needs much higher overhead for file replication and replica maintenance. This result show that LSA generates much less overhead in file replication than *Path*, while achieving comparable effectiveness in improving file lookup efficiency. It implies that LSA is effective to replicate files in traffic hubs or frequent requesters.

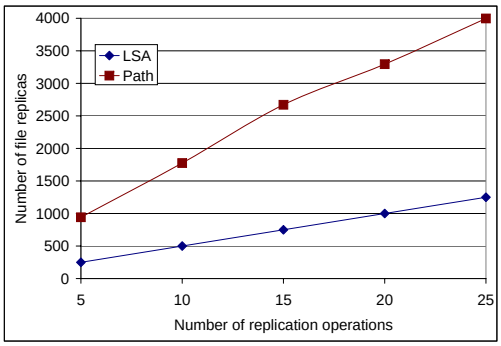

Figure 3. Number of replicas.

5 Conclusions

This paper proposes a Lightweight and Self-Adaptive file replication algorithm (LSA) that chooses query traffic hubs and frequent requesters as replica nodes to guarantee high utilization of replicas and high query efficiency. It leads to higher scalability and guarantees high replica utilization. Simulation results demonstrate the superiority of LSA in comparison with another file replication approach. It dramatically reduces the overhead of file replication and produces significant improvements in lookup efficiency. Its low overhead and high effectiveness are particularly attractive to the deployment of large-scale P2P file sharing systems.

Acknowledgements

This research was supported in part by the Acxiom Corporation.

References

[1] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-peer Storage Utility. In *Proc. of the 18th ACM Symp. on Operating Systems Principles (SOSP-18)*, 2001.

[2] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stocia. Wide-area cooperative storage with CFS. In *Proc. of the 18th ACM Symp. on Operating Systems Principles (SOSP-18)*, 2001.

[3] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer Caching Schemes to Address Flash Crowds. In *Proc. of the International workshop on Peer-To-Peer Systems (IPTPS)*, 2002.

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

the International Conference on Distributed Computing Systems (ICDCS), 2002.

[5] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher. Adaptive Replication in Peer-to-Peer Systems. In *Proc. of the International Conference on Distributed Computing Systems (ICDCS)*, 2004.

[6] Gnutella home page. http://www.gnutella.com.

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

[8] L. Yin and G. Cao. DUP: Dynamic-tree Based Update Propagation in Peer-to-Peer Networks. In *IEEE International Conference on Data Engineering (ICDE)*, 2005.

[9] R. Cox, A. Muthitacharoen, and R. T. Morris. Serving DNS using a Peer-to-Peer Lookup Service. In *Proc. of the International workshop on Peer-To-Peer Systems (IPTPS)*, 2002.

[10] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 1(1):17–32, 2003.

[11] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[12] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: An Infrastructure for Fault-tolerant wide-area location and routing. *IEEE Journal on Selected Areas in Communications*, 12(1):41–53, 2004.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, pages 329–350, 2001.

[14] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proc. of the USENIX Annual Technical Conference*, 2004.

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

[16] Fasttrack product description, 2001. http://www.fasttrack.nu/index_int.html.

[17] Morpheus home page. http://www.musiccity.com.

[18] Kazaa, 2001. Kazaa home page: www.kazaa.com.

[19] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Proc. of the International Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2001.

[20] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. of ACM SIGCOMM*, 2002.

[21] S. Tewari and L. Kleinrock. Analysis of Search and Replication in Unstructured Peer-to-Peer Networks. In *Proc. of ACM SIGMETRICS*, 2005.

[22] S. Tewari and L. Kleinrock. Proportional Replication in Peer-to-Peer Network. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2006.

[23] D. Rubenstein and S. Sahu. Can Unstructured P2P Protocols Survive Flash Crowds? *IEEE/ACM Trans. on Networking*, (3), 2005.

[24] P. Godfrey and I. Stoica. Heterogeneity and Load Balance in Distributed Hash Tables. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2005.

[25] H. Shen and C. Xu. Elastic Routing Table With Provable Performance for Congestion Control in DHT Networks. In *Proc. of the 26th International Conference on Distributed Computing Systems (ICDCS)*, 2006. Under review of TPDS.

[26] H. Shen. A Lightweight and Self-Adaptive File Replication in P2P File Sharing Systems. Technical Report TR-2008-01-87, Computer Science and Computer Engineering Department, University of Arkansas, 2008.