# IRM: Integrated File Replication and Consistency Maintenance in P2P Systems

Haiying Shen
Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, AR 72701
Email: hshen@uark.edu
Telephone: (479) 575–4382
Fax: (479) 575-5339

***Abstract*—In peer-to-peer file sharing systems, file replication and consistency maintenance are widely used techniques for high system performance. Despite significant interdependencies between them, these two issues are typically addressed separately. Most file replication methods rigidly specify replica nodes, leading to low replica utilization, unnecessary replicas and hence extra consistency maintenance overhead. Most consistency maintenance methods propagate update messages based on message spreading or a structure without considering file replication dynamism, leading to inefficient file update and hence high possibility of outdated file response. This paper presents an Integrated file Replication and consistency Maintenance mechanism (IRM), that integrates the two techniques in a systematic and harmonized manner. It achieves high efficiency in file replication and consistency maintenance at a significantly lower cost. Instead of passively accepting replica and update, each node determines file replication and update polling by dynamically adapting to time-varying file query and update rates, which avoids unnecessary file replications and updates. Simulation results demonstrate the effectiveness of IRM in comparison with other approaches. It dramatically reduces overhead and yields significant improvements on the efficiency of both file replication and consistency maintenance approaches.**

**Keywords: File replication, Consistency maintenance, Peer-to-Peer, Distributed Hash Table.**

## I. Introduction

Over the past years, the immerse popularity of Internet has produced a significant stimulus to peer-to-peer (P2P) file sharing systems. A recent large scale characterization of HTTP traffic [1] has shown that more than 75% of Internet traffic is generated by P2P applications. The percentage of P2P traffic has increased significantly as files such as videos and audios have become almost pervasive. The study also shows that the access to these files is highly repetitive and skewed towards the most popular ones. Such objects can exhaust the capacity of a node, leading to delayed response. File replication is an effective method to deal with the problem of overload condition due to flash crowds or hot files. It distributes load over replica nodes and improves file query efficiency. File consistency maintenance to maintain the consistency between a file and its replicas is indispensable to file replication. Requiring that the replica nodes be reliably informed of all updates could be prohibitively costly in a large system. Thus, file replication should proactively reduce unnecessary replicas to minimize the overhead of consistency maintenance, which in turn provides guarantee for the fidelity of consistency among file replicas considering file replication dynamism. File replication dynamism represents the condition with frequent replica node generation, deletion and failures.

Despite the significant interdependencies between file replication and consistency maintenance, they have been studied separately. In most current file replication methods, file owners rigidly specify replica nodes, and the replica nodes passively accept replicas. These methods replicate files close to file owners [2–4], requesters [5, 6] or along a query path between a file requester and a file owner [7]. These methods make it difficult to adjust the number of replicas to the time-varying utilization of replicas, and to ensure that all replicas are fully utilized. The number of replicas has a significant impact on the overhead of file consistency maintenance. Therefore, they lead to high overhead for unnecessary file replications and consistency maintenance.

On the other hand, in addition to centralized methods [8, 9] which are not suitable to decentralized large-scale P2P systems, most consistency maintenance methods update files by relying on a structure [10–14] or message spreading [15–17]. Though these methods generally can be applied to all file replication methods, without considering time-varying and dynamic replica nodes, they cannot be exploited to their full potential. Structure-based methods assume relatively stable replica nodes, which does not hold true due to dynamic replica nodes caused by file replication. File replication dynamism will lead to unsuccessful update propagation, and significantly high overhead for structure maintenance. System-wide message spreading will generate tremendously unnecessary redundant messages. In addition, they cannot guarantee that all replica nodes can receive a update. Therefore, without taking into account file replication dynamism, consistency maintenance generates unnecessary overhead, and cannot guarantee the fidelity of replica consistency. Furthermore, as in file replication, passively accepting update messages makes it difficult to avoid unnecessary updates to reduce overhead.

Uncoordinated deployment of file replication and consistency maintenance techniques can negate each other's efforts

and lead to suboptimal or even low system performance. As a result, file replication is faced with a challenge to minimize the number of replicas to reduce the consistency maintenance overhead, without compromising its efficiency in hot spot and query latency reduction. On the other hand, consistency maintenance is faced with a challenge to guarantee the fidelity of replica consistency in a timely fashion with low overhead considering file replication dynamism. This makes it important to integrate the two techniques to enhance their mutual interactions and avoid their conflicting behaviors, ensuring that the two techniques can be exploited to their fullest capacities.

This paper presents an Integrated file Replication and consistency Maintenance mechanism (IRM) that achieves high efficiency in file replication and consistency maintenance at a significantly lower cost. IRM integrates file replication and consistency maintenance in a harmonic and coordinated manner. Basically, each node decides to create or delete a replica, and to actively poll for update based on file query and update rates, in a totally decentralized and autonomous manner. It replicates highly-queried files and polls at a high frequency for frequently updated and queried files. IRM avoids unnecessary file replications and updates by dynamically adapting to time-varying file query and update rates. It improves replica utilization, file query efficiency and consistency fidelity. A significant feature of IRM is that it achieves an optimized tradeoff between overhead and query efficiency as well as consistency guarantees.

We will introduce IRM's application in structured P2Ps though it is also applicable to unstructured P2Ps. The rest of this paper is structured as follows. Section II presents a concise review of representative file replication and consistency maintenance approaches for P2P systems. Section III describes the IRM mechanism including file replication and consistency maintenance algorithms. Section IV shows the performance of IRM in static as well as dynamic situations in comparison with other approaches by a variety of metrics. Section V concludes this paper.

## II. Related Work

File replication in P2P systems is targeted to release the load in hot spots and meanwhile decrease file query latency. Most traditional file replication methods replicate files near file owners [2–4], file requester [5, 6] or along a query path from a requester to a owner [7]. PAST [2], CFS [3] and Backslash [4] replicate each file on close nodes near the file's owner. Backslash also pushes cache one hop closer to requesters as soon as nodes are overloaded. In LAR [5] and Gnutella [6], overloaded nodes replicate a file at requesters. Freenet [16] replicates files on the path from a requester to a file owner. CFS, PAST, LAR [5] cache routing hints along the search path of a query. Cox *et al.* [7] studied providing DNS service over a P2P network as an alternative to traditional DNS. They cache index entries, which are DNS mappings, along search query paths. Overlook [18] deploys client-access history to place a replica of a popular file on a node with most lookup requests for fast replica location. LessLog [19] determines the replicated nodes by constructing a lookup tree based on IDs to determine the location of the replicated node. In OceanStore [20], files are replicated and stored on multiple servers for security concern without restricting the placement of replicas. OceanStore maintains two-tier replicas: a small durable primary tier and a large soft-state second tier. Other studies of file replication investigated the relationship between the number of replicas, file query latency and load balance [21–26] in unstructured P2P systems. In most of these methods, file owners rigidly determine replica nodes and nodes passively accept replicas. They are unable to keep track replica utilization to reduce under-utilized replicas and ensure high utilization of existing replicas.

Along with file replication, numerous file consistency maintenance methods have been proposed. They generally can be classified into two categories: structure based [10, 11, 13, 14] and message spreading based [15–17]. The work in [10] constructs a hierarchical structure with locality consideration, and SCOPE [11] constructs a tree for update propagation. CUP [13] and DUP [14] propagate update along a routing path. P2P systems are characterized by churn where nodes join and leave continuously and frequently. In these methods, if any node in the structure fails, some replica nodes are no longer reachable. Moreover, they need high overhead for structure maintenance especially in churn. In the category of message spreading based methods, FreeNet [16] routes an update to other nodes based on key closeness. Lan and *et al.* [17] proposed to use flooding-based push for static objects and adaptive poll for dynamic objects. IRM also uses adaptive polling, but it mainly focuses on the integration of two components. In hybrid push/poll algorithm [15], flooding is substituted by rumor spreading to reduce communication overhead. In these methods, an update is not guaranteed to reach every replica, and redundant messages generate high propagation overhead. Raunak and *et al.* proposed polling method for web cache consistency [27]. Its context is static, in which the proxies are always available. Thus, this method is not applicable for a P2P dynamic environment. IRM further considers file query rate for polling frequency determination to reduce overhead, and avoids overloading file owner.

## III. IRM: Integrated File Replication and Consistency Maintenance Mechanism

Instead of passively accepting replicas and update messages, IRM harmonically integrates file replication and consistency maintenance by letting each node determine file replication and update based on actual file query rate and update rates. IRM file replication places replicas in frequently-visited nodes to guarantee high utilization of replicas, and meanwhile reduce under-utilized replicas and overhead of consistency maintenance. IRM consistency maintenance in turn aims to guarantee file fidelity of consistency at a low cost with file replication dynamism consideration. Using adaptive polling, it ensures timely update operation and avoids unnecessary updates. As a result, IRM achieves high efficiency in both file replication and consistency maintenance.

### A. Adaptive File Replication

IRM achieves an optimized tradeoff between query efficiency and overhead in file replication. It dynamically tunes to time-varying file popularity and node interest, and adaptively determines replica nodes based on query traffic. IRM addresses two main problems in file replication to achieve its goal: (1) where to replicate files so that the file query can be significantly expedited and the replicas can be fully utilized? (2) how to remove under-utilized file replicas so that the overhead for consistency maintenance is minimized?

*1) Replica nodes determination :* In structured P2P systems, some nodes carry more query traffic load while others carry less. Therefore, frequent requesters of a file and traffic junction nodes (i.e. hot routing spots) in query paths should be the ideal file replica nodes for high utilization of file replicas. Based on this, IRM replicates a file in nodes that have been very interested in the file or routing nodes that have been carrying more query traffic of the file. The former arrangement lets frequent requesters of a file get the file without query routing, and the latter increases the possibility that queries from different directions encounter the replica nodes, thus making full use of file replicas. In addition, replicating file in the middle rather than in the end of a query path speeds up file query.

*2) Replica creation :* We define a requester's *query initiating rate* for file $f$, denoted by $q_f$, as the number of queries for $f$ sent by the requester during a unit time, say one second. A file requester records its $q_f$ for each file requested. IRM sets a threshold for query initiating rate, $T_q$, based on normal query initiating rates in the system. When a requester receives a file, it checks its $q_f$. If $q_f > T_q$, it makes a replication of the file. We define a node's *query passing rate* of file $f$, denoted by $l_f$, as the number of queries for file $f$ received and forwarded by the node during a unit time. IRM sets a threshold for query passing rate, $T_l$, based on normal query passing rates in the system. In IRM, when a routing node receives a query for file $f$, it checks $l_f$. In the case that $l_f > T_l$ and the node has available capacity for a file replica, it adds a file replication request into the original file request with its IP address. After the file destination receives the query, if it is overloaded, it checks if the file query has additional file replication requests. If so, it sends the file to the replication requesters in addition to the query initiator. Otherwise, it replicates file $f$ to its neighbors that forward the queries of file $f$ most frequently.

*3) Replica adaptation :* Considering that file popularity is non-uniform and time-varying and node interest varies over time, some file replicas become under-utilized when there are few queries for the files. To deal with this situation, IRM lets each replica node periodically update their query passing rate or query initiating rate of a file. If the rates are below their thresholds, the node removes the replica. Therefore, the determination of keeping file replicas is based on recently experienced query traffic due to file query rate. If a file is no longer requested frequently, there will be no file replica for it. The adaptation to query initiating and passing rate ensures that all file replicas are worthwhile and there is no waste of overhead for unnecessary file consistency maintenance.

### B. File Consistency Maintenance

In IRM poll-based consistency maintenance, each replica node polls its file owner or another node to validate whether its replica is the up-to-date file, and updates its replica accordingly. IRM addresses three main issues in consistency maintenance. (1) how to determine the frequency that a replica node probe a file owner in order to guarantee timely file update? (2) how to reduce the number of polling operations to save cost and meanwhile provide the fidelity of consistency guarantees? (3) how to avoid overloading file owner in polling?

*1) Polling frequency determination :* Consider a file's maximum update rate is $1/\triangle t$, which means it updates every $\triangle t$ time units, say seconds, in the highest update frequency. Therefore, a file replica node can ensure that a replica will not be outdated by more than $\triangle t$ seconds by polling the owner every $\triangle t$ seconds. Since the rate of file change varies over time as hot files become cold and vice versa, a replica node should be able to adapt its polling frequency in response to the variations. In IRM, a replica node intelligently tailors its polling frequency so that it polls at approximately the same frequency of file change using a *linear increase multiplicative decrease* algorithm. The algorithm has been effectively used in many systems to adapt to changing system conditions [28, 17]. That is, frequently modified files are polled more frequently than relatively static files.

IRM associates a time-to-refresh (TTR) value with each replica. The TTR denotes the next time instant a node should poll the owner to keep its replica updated. The TTR value is varied dynamically based on the results of each polling. The value is increased by an additive amount if the file doesn't change between successive polls:

$$TTR = TTR_{old} + \alpha \qquad (3)$$

where $\alpha$, $\alpha > 0$, is an additive constant. In the event the file is updated since the last poll, the TTR value is reduced by a multiplicative factor:

$$TTR = TTR_{old}/\beta \qquad (4)$$

where $\beta$, $\beta > 1$, is the multiplicative decrease constant. The algorithm takes as input two parameters: $TTR_{min}$ and $TTR_{max}$, which represent lower and upper bounds on the TTR values. Values that fall outside these bounds are set to

$$TTR = max(TTR_{min}, min(TTR_{max}, TTR)) \qquad (5)$$

The bounds ensure that the TTR is neither too large nor too small. Typically, $TTR_{min}$ is set to $\triangle t$, since this is the minimum interval between polls necessary to maintain consistency guarantees. The algorithm begins by initializing $TTR = TTR_{min} = \triangle t$.

*2) Poll reduction and bottleneck avoidance :* In addition to file change rate, file query rate is also a main factor needed to be considered in consistency maintenance. Even when a file changes frequently, if a replica node does not receive queries

TABLE I
SIMULATED ENVIRONMENT AND ALGORITHM PARAMETERS.

| Parameter | Default value |
|---|---|
| Object arrival location | Uniform over ID space |
| Number of nodes | 4096 |
| Node capacity $c$ | Bounded Pareto: shape 2<br>lower bound: 500<br>upper bound: 50000 |
| Number of queried files | 50 |
| Number of queries per file | 1000 |
| Number of replicating operations per file | 5-25 |
| $T_l$, $T_q$ | 2 |
| $u$ | 0.5 |
| Observation period | 1 second |

for the file, or hardly queries for the file during a time period, it is an overhead waste to poll the file's owner for validation during the time period. IRM combines file query rate into consideration for poll time determination. We use $TTR_{query}$ and $TTR_{poll}$ to denote the next time instant of corresponding operation of a file.

Note that IRM polling algorithm uses TTR to approximately represent file change frequency. When $TTR \leq TTR_{query}$, that is, when the file change rate is higher than the file query rate, there is no need to update the replica at the rate of file change rate. Therefore, the polling rate can be equal to file query rate in this case. On the other hand, when $TTR > T_{query}$, that is, the file is queried at a higher rate than change rate, then the file should be updated timely based on $TTR$.

As a result, the number of polls and the overhead for file updates are reduced without compromising the file fidelity of consistency; that is, there is low possibility that a file is outdated when visited. IRM consistency maintenance guarantees file fidelity of consistency at a significantly lower cost.

## IV. PERFORMANCE EVALUATION

We designed and implemented a simulator for evaluating the IRM mechanism based on Chord P2P system [8]. We compared IRM with representative approaches of file replication and consistency maintenance. Experiment results show that IRM file replication algorithm is highly effective in reducing file query latency, the number of replicas, and file consistency maintenance overhead. IRM file consistency maintenance in turn provides a guarantee of file fidelity of consistency and dramatically reduces consistency maintenance overhead. Table I lists the parameters of the simulation and their default values.

### A. File Replication

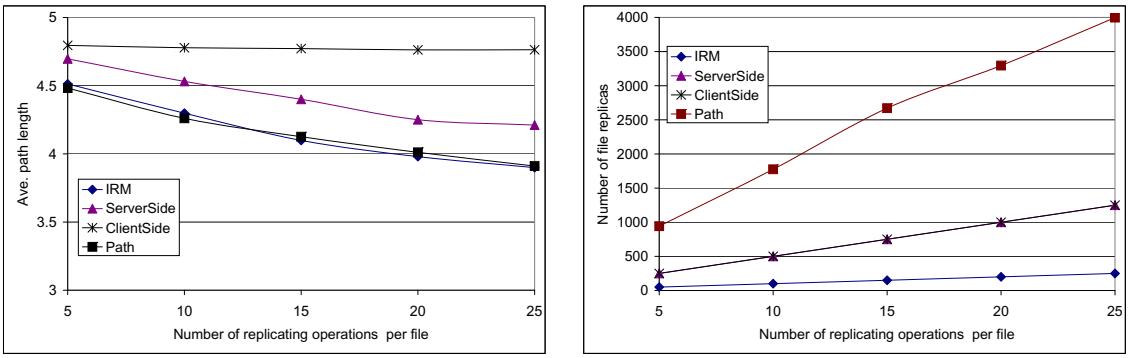
We compared the performance of IRM with *ServerSide* [2], *ClientSide* [5] and *Path* [7] in terms of average lookup path length, replica hit rate, and the total number of file replicas versus the number of replicating operations per file. In each replicating operation, IRM, *ServerSide* and *ClientSide* replicate a file to a single node while *Path* replicates a file to a number of nodes along a query path. The file requesters were randomly chosen in the experiment. We use *replica hit rate* to denote the percentage of the number of file queries that are resolved by replica nodes among total queries.

Figure 1(a) plots the average path length of different approaches. We can see that *Path* generates shorter path length than *ServerSide* and *ClientSide*, and IRM leads to approximately the same path length as *Path*. Unlike others that replicate a file only in one node in each file replicating operation, *Path* replicates a file in nodes along a query path. Therefore, it increases replica hit rate and produces shorter path length. IRM can achieve almost the same lookup efficiency with much less replicas. It shows the effectiveness of IRM to replicate files in nodes with high query rate, which enhances the utilization of replicas and hence reduces lookup path length. *ServerSide* replicates a file close to the file's owner, such that the file's request will encounter a replica node before arriving at the file owner, shortening lookup path length. However, since the replica nodes locate close to the file owner, the requests need to travel more hops than in *Path* and IRM, so that it is not able to significantly reduce lookup path length. *ClientSide* generates much longer lookup path length compared to others. It is because files are replicated in requesters, and the replica nodes may not request the same file later due to varied node interest. Consequently, *ClientSide* is not able to make full use of file replicas to reduce lookup path length. In contrast, IRM replicates a file in a requester only when the requester's query rate reaches a specified threshold, which increases the utilization of replicas.

We set the number of hot files as 10, and tested the total number of replicas at a random time instant. Figure 1(b) shows that the number of replicas increases as the number of replicating operations per file increases. This is due to the reason that more replication operations for a file leads to more replicas. The result of *Path* is excessively higher than others. It is because in each file replication operation, a file is replicated in multiple nodes along a routing path in *Path*, but in a single node in *ServerSide*, *ClientSide* and IRM. Therefore, *Path* needs much higher overhead for file replications and subsequent consistency maintenance. The figure also shows that IRM has less replicas than *ServerSide* and *ClientSide*. IRM adjusts the number of file replicas adaptively based on file query rate, such that less popular or infrequently-requested files have less file replicas. The results confirm the effectiveness of IRM on removing unnecessary file replicas and keeping replicas worthwhile. Thus, IRM saves file consistency maintenance overhead by not having to maintain replicas for infrequently-requested files.

### B. File Consistency Maintenance

We use *Hierarchy* to denote the work in [10] that builds a hierarchical structure for file consistency maintenance. We compared the performance of IRM with *SCOPE* [11], *Hierarchy* [10] and *Push/poll* [15] methods in terms of file consistency maintenance cost, and the capability to keep fidelity of file consistency. In *Hierarchy*, we set the number of nodes in a cluster to 16. We assumed four types of file: highly mutable, very mutable, mutable and immutable. The




(a) Ave. path length (b) Number of replicas

Fig. 1. Performance of file replication algorithms.

percentage of the files in each category and their update rates were (0.5%, 0.15 sec), (2.5%, 7.5 sec), (7%, 30 sec) and (90%, 100 sec). File queries were successively generated. The query interval time was randomly chosen between 1 to 500 seconds.

The next experiment evaluated the performance of different consistency maintenance methods with churn in P2P systems. In the experiment, the number of replica nodes was set to 4000, and the failed nodes were randomly chosen. Figure 2(a) shows the average number of update messages per replica node versus the percentage of failed replica nodes. We can see that the number of update messages increases as the percentage of failed replica nodes increases in *SCOPE* and *Hybrid*, but remains constant in IRM and *Push/poll*. When a replica node fails, *SCOPE* and *Hybrid* need to recover tree structure and retransmit update messages, which generate more update messages. The failure of replica nodes has little effect on IRM and *Push/poll* due to their autonomous polling and message spreading nature, respectively. However, the number of update messages of *Push/poll* is much higher than other schemes, which means that its churn-resilience is achieved at the cost of high node communication cost. Without redundant messages and structure maintenance, IRM achieves churn-resilience at a dramatically lower cost.

If a replica is not updated in a timely fashion, file requesters may receive stale files. Figure 2(b) depicts the percentage of stale files received by requesters versus the percentage of failed nodes. It shows that *SCOPE* and *Hierarchy* incur much higher percentage rates than others. This is because they rely on tree structure for update propagation, and if a node fails, all the node's children cannot get the update message in time. In contrast, IRM and *Poll/push* do not depend on a structure. They respectively use autonomous polling and rumoring that are highly resilient to churn. The figure also shows that *Poll/push* leads to higher percentage of stable file responses than IRM. It is because rumoring cannot guarantee that all replica nodes can receive a update message. Moreover, the high churn-resilience of *Poll/push* is outweighed by its high cost of redundant update messages. In IRM, when a replica node receives a request during replica validation, it can wait until it receives the update message before it replies to the requester. The experiment results imply that IRM outperforms other methods with regards to the consistency maintenance overhead and the capability to guarantee fidelity of consistency in churn.

## V. Conclusions

In spite of the efforts to develop file replication and file consistency maintenance methods in P2P systems, there has been very little research devoted to tackling both challenges simultaneously. This is unfortunate because they are intimately connected: file replication needs consistency maintenance to keep the consistency between a file and its replicas, and on the other hand, the overhead of consistency maintenance is determined by the number of replicas. Connecting the two important components will greatly enhance system performance. In addition, traditional file replication and consistency maintenance methods either are not sufficiently effective or incur prohibitively high overhead.

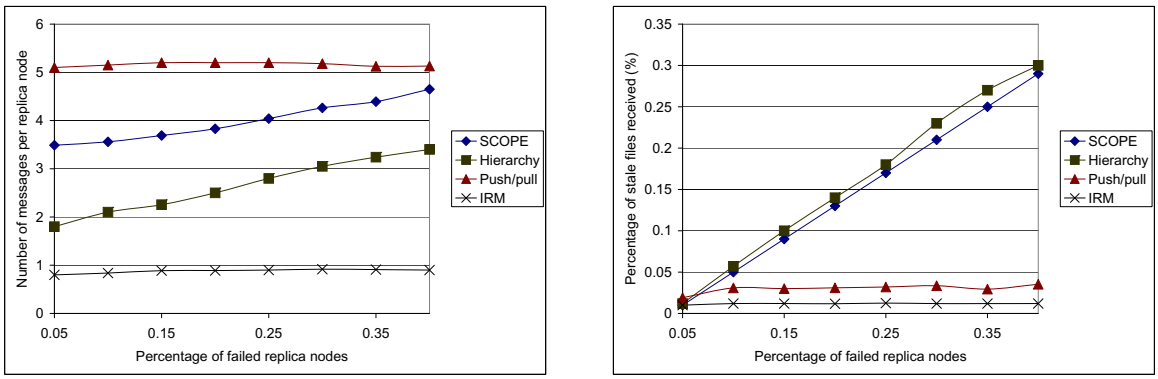
This paper proposes an Integrated file Replication and consistency Maintenance mechanism (IRM) that achieves high efficiency at a significantly lower cost. Instead of passively accepting replicas and updates, nodes autonomously determine the need for file replication and validation based on file query rate and update rate. It guarantees the high utilization of file replicas, high query efficiency and fidelity of consistency. At the same time, IRM reduces redundant file replicas, consistency maintenance overhead, and unnecessary file updates. Simulation results demonstrate the effectiveness of IRM in comparison with other file replication and consistency maintenance approaches. Its low overhead and high effectiveness are particularly attractive to the deployment of large-scale P2P systems.

## Acknowledgment

This research was supported in part by the Acxiom Corporation.

## References

[1] J. Liang, R. Kumar, and K. W. Ross. The FastTrack overlay: A measurement study. *Computer Networks*, (6), 2006.




(a) Num. of messages with churn

(b) Stale file responses with churn

Fig. 2. Performance of file consistency maintenance algorithms.


[2] 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.

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

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

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

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

[7] R. Cox, A. Muthitacharoen, and R. T. Morris. Serving DNS using a Peer-to-Peer Lookup Service. In *Proc. of IPTPS*, 2002.

[8] 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.

[9] 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.

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

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

[12] P. Druschel, M. Castro, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *J-SAC*, 2002.

[13] M. Roussopoulos and M. Baker. CUP: Controlled Update Propagation in Peer to Peer Networks. In *Proc. of USENIX*, 2003.

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

[15] A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *Proc. of ICDCS*, 2003.

[16] I. Clarke, O. Sandberg, and et al. 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.

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

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

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

[20] J. Kubiatowicz, D. Bindel, and et al. OceanStore: An architecture for global-scale persistent storage. In *Proc. of the ASPLOS*, 2000.

[21] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proc. of ICS*, 2001.

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

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

[24] S. Tewari and L. Kleinrock. On Fairness, Optimal Download Performance and Proportional Replication in Peer-to-Peer Networks. In *Proc. of IFIP Networking*, 2005.

[25] S. Tewari and L. Kleinrock. Proportional Replication in Peer-to-Peer Network. In *Proc. of INFOCOM*, 2006.

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

[27] B. Urgaonkar, A. Ninan, M. Raunak, and et al. Maintaining Mutual Consistency for Cached Web Objects. In *Proc. of ICDCS*, 2001.

[28] W. R. Stevens. *TCP/IP Illustrated Volume 1*. Addison-Wesley, 1994.