IRM: Integrated File Replication and Consistency Maintenance in P2P Systems

Haiying (Helen) Shen, *Member*, *IEEE*

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without considering file replication dynamism, leading to inefficient file update and hence high possibility of outdated file response.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 low cost. Instead of passively accepting replicas and updates, 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.

Index Terms—File replication, consistency maintenance, peer-to-peer, distributed hash table.

1 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 percent 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. Fig. 1 demonstrates the interrelationship between file replication and consistency maintenance.

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. The methods were designed without considering the efficiency of subsequent file consistency maintenance. Specifically, these methods replicate files close to file owners [2], [3], [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. Large number of replicas needs more updates hence high consistency maintenance overhead and vice versa. Therefore, the methods 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], [11], [12], [13], [14] or message spreading [15], [16], [17]. Though these methods generally can be applied to all file replication methods, they cannot be exploited to their full potential without considering time-varying and dynamic replica nodes. Structure-based methods assume relatively stable replica nodes, which does not hold true in practice due to dynamic replica nodes caused by file replication. Replica nodes may be continuously and rapidly generated, deleted, and fail. Such 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 message. Therefore, without taking into account file replication dynamism, consistency maintenance generates unnecessary overhead and cannot help to guarantee the fidelity of replica consistency. Furthermore, as in file replication, passively

• *The author is with the Department of Electrical and Computer Engineering, Clemson University, 313-B Riggs Hall, Clemson, SC 29634. E-mail: shenh@clemson.edu.*

Manuscript received 6 Sept. 2008; revised 18 Feb. 2009; accepted 20 Feb. 2009; published online 5 Mar. 2009.
Recommended for acceptance by K. Hwang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-09-0338.
Digital Object Identifier no. 10.1109/TPDS.2009.43.

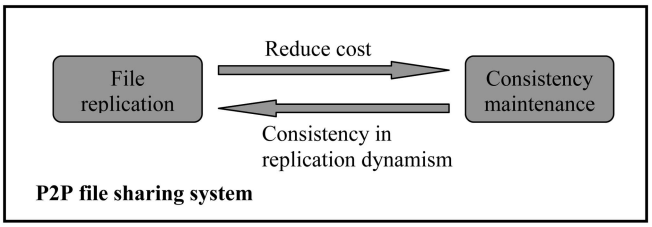


Fig. 1. Interrelationship between file replication and consistency maintenance.

accepting update messages makes it difficult to avoid unnecessary updates in order 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, on one hand, 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 harmonized and coordinated manner. Basically, each node actively decides to create or delete a replica and to 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 trade-off between overhead and query efficiency as well as consistency guarantees.

IRM is ideal for P2P systems due to a number of reasons. First, IRM does not require a file owner to keep track of replica nodes. Therefore, it is resilient to node joins and leaves, and thus suitable for highly dynamic P2P systems. Second, since each node determines its need for a file replication or replica update autonomously, the decisions can be made based on its actual query rate, eliminating unnecessary replications and validations. This coincides in spirit with the nature of node autonomy of P2P systems. Third, IRM enhances the guarantee of file consistency. It offers the flexibility to use different replica update rate to cater to different consistency requirements determined by the nature of files and user needs. Faster update rate leads to higher consistency guarantee, and vice versa. Fourth, IRM ensures high possibility of up-to-date file responses. A replica node does not serve a request right away during polling period. In contrast, in pushing, a node cannot guarantee the up-to-date status of its response file by passively accepting update. Within the knowledge of the author, there is no other work that addresses the problem of the interrelationship between the file replication and consistency maintenance. While this paper proposes a solution for the problem, the author hopes that this work will stimulate other advanced methods for this crucial problem in P2P file sharing systems.

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 2 presents a concise review of representative file replication and consistency maintenance approaches for P2P systems. Section 3 describes the IRM mechanism including file replication and consistency maintenance algorithms. Section 4 shows the performance of IRM in static as well as dynamic situations in comparison with other approaches by a variety of metrics. Section 5 concludes this paper.

2 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 rigidly determine the places of replicas and push the replicas to the places. Generally, the methods replicate files near file owners [2], [3], [4], file requesters [5], [6], or along a query path from a requester to a owner [16], [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. The caches 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], [22], [23], [24], [25], [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 underutilized replicas and ensure high utilization of existing replicas. Thus, unnecessary replicas lead to a waste of consistency maintenance. Yang et al. proposed Parity Replication in IP-Network Storages (PRINS) [27]. PRINS replicates the parity of a data block upon each write operation instead of the data block itself. The data block will be recomputed back at

the replica storage site upon receiving the parity. PRINS trades off high-speed computation for communication that is costly distributed storages. In our previous work, we proposed an efficient and adaptive decentralized file replication algorithm in P2P file sharing systems called EAD [28]. In the method, traffic hubs that carry more query load and frequently requesters are chosen as replica nodes. The nodes periodically compute their query load to create replicas and remove underutilized replicas. IRM is developed by leveraging EAD. It shares similarity with EAD in file replication strategies. The novelty of IRM lies in the observation of the interrelationship between the file replication and consistency maintenance technologies, and the harmonic integration of the two technologies. The autonomous feature of nodes in the file replication strategies facilitates IRM to flexibly minimize the number of replicas for low cost in subsequent consistency maintenance while maintaining the high effectiveness of file replication in releasing the load in hot spots and improving query efficiency.

Replication in a structured P2P system is to decrease file query time, while replication in an unstructured P2P system is to decrease the search time. Unstructured P2P systems allow for more proactive replications of objects, where an object may be replicated at a node even though the node has not requested the object. Lv et al. [21] and Cohen and Shenker [22] showed 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 method replicating files proportionally to the square root of their popularity is such that both average search size and capacity utilization rate vary per object, but the variance in utilization is considerably smaller than with uniform replication, and the variance in average search size is considerably smaller than with proportional replication. Tewari and Kleinrock [23], [24], [25] showed that proportional replication can optimize flooding-based search, download time, and workload distribution. They also showed that local storage management algorithms such as Least Recently Used (LRU) automatically achieve near-proportional replication and that the system performance with the replica distribution achieved by LRU is very close to optimal. APRE [29] adaptively expands or contracts the replica set of a file 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. Rubenstein and Sahu [26] also discussed the scalability achieved by the fact that user requests create additional replicas which improves system performance although they focused on the system's ability to find the newly created sources.

Most of the studies focused on protocol design and implementation to avoid hot spots and enhance file availability, but did not address the fundamental issues of how replicas should be managed. File consistency maintenance is actually indispensable to file replication. Along with file replication, numerous file consistency maintenance methods have been proposed. They generally can be classified into two categories: structure based [10], [11], [12], [13], [14] and message spreading based [15], [16], [17], [6]. Some methods [10], [11] build one structure for each replicated file. The work in [10] constructs a hierarchical structure with locality consideration, and SCOPE [11] constructs a tree for update propagation. Scribe [12] is a P2P-based publish/subscribe system that provides a decentralized event notification mechanism for publishing systems. The paths from subscribers to the publisher are recorded for update notifications. 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 et al. [17] proposed to use flooding-based push for static objects and adaptive poll for dynamic objects. In hybrid push/poll algorithm [15], flooding is substituted by rumor spreading to reduce communication overhead. At every step of rumor spreading, a node pushes updates to a subset of related nodes it knows, only providing partial consistency. In these methods, an update is not guaranteed to reach every replica, and redundant messages generate high propagation overhead. Raunak et al. proposed polling method for web cache consistency [30]. Its context is static, in which the proxies are always available. Thus, this method is not applicable for a P2P dynamic environment. Muthitacharoen et al. [31] proposed lease algorithm for consistency. The lease is a commitment on the part of the server to notify the client of any modifications made to that file during the term of the lease. When the lease has expired, a client must request the server to transfer the new contents to itself. rsync [32] is a software application for Unix systems which synchronizes files and directories from one location to another while minimizing data transfer using delta encoding when appropriate. An important feature of rsync not found in most similar protocols is that the mirroring takes place with only one transmission in each direction. IRM shares similarity with the work in [30], [17] in employing linear increase multiplicative decrease algorithm in polling for consistency. However, the focus of IRM is to integrate file replication and consistency maintenance to enhance their mutual interactions and reduce their overhead. IRM further considers both file update rate and file query rate for polling frequency determination to reduce overhead and avoid overloading file owner.

In most of these file replication and consistency maintenance methods, nodes passively accept replicas and update messages. They are unable to keep track the utilization of replicas to determine the need of file replicas and replica updates. Minimization of the number of replicas helps to reduce unnecessary updates in consistency maintenance, but it should still keep the efficiency of file replication to release the load in hot spots and to improve query efficiency. More importantly, despite the significant interdependencies between file replication and file consistency maintenance, these two issues are typically addressed

separately. There has been a increasingly desire for a mechanism combining file replication and consistency maintenance that brings more benefits than overhead. IRM harmonically integrates file replication and consistency maintenance and determines the need of file replication and update based on the balance between profit and cost. In IRM, nodes determine the need of replication of a file and the frequency of update polling based on the file's query rate and update rate in order to provide high efficiency, and meanwhile avoid unnecessary overhead in both file replication and consistency maintenance. IRM shares the similarity with the file replication in unstructured P2P systems in that file popularity is taken into account in file replication. However, these file replication methods cannot be directly applied to structured P2P systems due to their basic protocol differences.

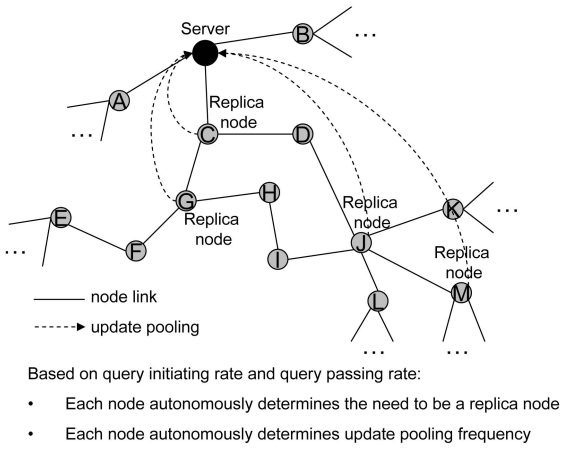


Fig. 2. IRM file replication and consistency maintenance.

3 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 autonomously determine the need for 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 underutilized 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, IRM ensures timely update operation and avoids unnecessary updates. As a result, IRM achieves high efficiency in both file replication and consistency maintenance.

The basic idea of IRM is to use file query and update rate to direct file replication and consistency maintenance. When a node receives queries for a file frequently or itself queries a file frequently, placing a replica in the node can improve the query efficiency and meanwhile make full use of replicas. When a replica node doesn't receive queries for its replica frequently or itself doesn't query its replica frequently, it removes the replica. IRM aims to guarantee that a file is the updated file when visited. Based on this principle, a node adaptively polls file owner for update based on file query rate and update date to avoid unnecessary overload.

Fig. 2 shows an example for file replication and consistency maintenance in IRM. The nodes C and G in the middle observed frequent queries for the file, and nodes J and M queried for the file frequently. Therefore, they made a copy of the file in themselves. The replica nodes periodically check the query rate. When their replicas are underutilized, they remove the replicas. For consistency maintenance, the replica nodes actively probe the file server for update. They probe the server approximately at the rate of file change rate. However, when their replica is visited at lower frequency than the file's change frequency, they probe the server at the frequency that the replica is visited. This strategy reduces the overhead of consistency maintenance while still guarantees the up-to-date status of the visited replicas. In the following, we first present the IRM mechanism in terms of file replication and consistency maintenance. We then discuss how IRM achieves harmonized integration of file replication and consistency maintenance.

3.1 Adaptive File Replication

IRM is developed by leveraging our previous work of EAD [28] file replication algorithm. IRM's file replication component shares similarity with EAD. The replication algorithm achieves an optimized trade-off between query efficiency and overhead in file replication. In addition, it dynamically tunes to time-varying file popularity and node interest, and adaptively determines replica nodes based on query traffic. In the following, we introduce IRM's file replication component by addressing two main problems in file replication: 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 underutilized file replicas so that the overhead for consistency maintenance is minimized?

3.1.1 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 enables frequent requesters of a file to 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 ends of a query path speeds up file query.

3.1.2 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, denoted by T_q. It could be the product of a constant factor and the normal query initiating rate 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, denoted by T_l. It could be the product of a constant factor and the normal query passing rate 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.

3.1.3 Replica Adaptation

Considering that file popularity is nonuniform and time-varying and node interest varies over time, some file replicas become underutilized 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.

3.2 File Consistency Maintenance

Maintaining consistency between frequently updated or even infrequently updated files and their replicas is a fundamental reliability requirement for a P2P system. P2P systems are characterized by dynamism, in which node join and leave continuously and rapidly. Moreover, replica nodes are dynamically and continuously created and deleted. The dynamism has posed a challenge for timely update in structured-based consistency maintenance methods. On the other hand, consistency maintenance relying on message spreading generate high overhead due to dramatically redundant messages. Rather than relying on a structure or message spreading, IRM employs adaptive polling for file consistency maintenance to cater to file replication dynamism. A poll approach puts the burden of consistency maintenance on individual nodes. Unlike push, poll approach can achieve good consistency for distant nodes and is less sensitive to P2P dynamism, network size, and the connectivity of a node.

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 two 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?

3.2.1 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. In this case, a file replica node can ensure that a replica is never 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. Specifically, like the works in [30], [17], IRM employs a *linear increase multiplicative decrease* algorithm. The algorithm has been effectively used in many systems to adapt to changing system conditions [33]. 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

$$TTR = TTR_{old} + \alpha, \tag{3}$$

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:

$$TTR = TTR_{old}/\beta, \tag{4}$$

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

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

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 $TTR = TTR_{min} = \triangle t$.

The probed server may depart or fail. In structured P2P systems, before a node leaves, it transfers its files to the files' new owners based on the P2P file allocation algorithm. Therefore, if a replica node has not received the response from its probed server during a certain period time, it probed the file's new owner. If it still has not received response from the new owner, it assumes that the server fails and removes the replica correspondingly. Otherwise, it updates its file according to the information from the new owner.

3.2.2 Poll Reduction

In addition to the file change rate, file query rate is also a main factor to consider in consistency maintenance. Even when a file changes frequently, if a replica node does not receive queries 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. However, most current consistency maintenance methods neglect the important role that file query rate plays in reducing

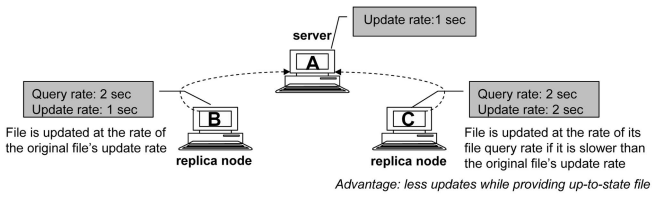


Fig. 3. An example of poll reduction in consistency maintenance.

overhead. 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. IRM assumes that in a certain period of time, the file query rate does not vary greatly. That is, the pattern of a user's file access behavior remains similar.

Algorithm 1. Pseudo-code for the IRM adaptive file consistency maintenance algorithm

```
//operation at time instant T_poll
if there is a query for the file then
    include a polling request into the query for file f
else
    send out a polling request
if get a validation reply from file owner then{
    if file is valid then
        TTR = TTR_old + α
    if file is stale then{
        TTR = TTR_old / β
        update file replica}
    if TTR > TTR_max or TTR < TTR_min then
        TTR = max(TTR_min, min(TTR_max, TTR))
    if TTR ≤ T_query then
        TTR_poll = T_query
    else
        TTR_poll = TTR}
```

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. This is because the ultimate goal of consistency maintenance is to guarantee the received file is up to state. If a replica is updated soon after its original file is changed but there is no query for this replica until after the next update, it is a waste to update the file this time. For example, a file changes every 1 second while it is visited every 2 seconds, then updating replica once every 2 seconds can guarantee that the response file from replica node is the up-to-date file.

Fig. 3 shows an example of update reduction in consistency maintenance. Node A has the original file of two replicas in node B and C, respectively. The original file is changed every 1 seconds. The query rate of this file in nodes B and C is once every 2 seconds. Node B updates its replica soon after its original file is updated. Node C updates its replica at the rate equals to the replica query rate. Thus, it guarantees that the provided file is up to date, and meanwhile reduces the poll operation and overhead. Therefore, the polling rate can be equal to file query rate in the case that replica query rate is slower than the original file change rate. 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, TTR_{poll} should be calculated based on the following formula:

$$TTR_{poll} = \begin{cases} T_{query} & TTR \leq T_{query}, \\ TTR & TTR > T_{query}. \end{cases}$$

Similar to file replication requests, replica nodes try to include all polling messages along with queries in order to save the polling overhead. Algorithm 1 shows the pseudo-code of the adaptive file consistency maintenance algorithm. 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.

3.3 Comparative Discussion of IRM

In IRM, the file replication component and the consistency maintenance component are integrated in a coordinated manner. IRM avoids the negative effect between the two components and enables them to be exploited to their fullest capacity. On the one hand, file replication helps to minimize the number of replicas in order to minimize the overhead of consistency maintenance. On the other hand, consistency maintenance helps to guarantee the fidelity of consistency among replicas in file replication dynamism. Briefly, IRM arranges each node to autonomously determine the need to replicate a file and update a replica based on its experienced traffic of the file. This harmonized integration helps IRM achieve high efficiency and effectiveness in both file replication and consistency maintenance.

3.3.1 The Impact of File Replication on Consistency Maintenance

Traditional file replication methods could generate much more replicas due to a variety of reasons. First, in some traditional methods [2], [3], servers keep track of the query forwarding rate of nodes and replicate files in nodes that frequently forward queries to the servers. The servers holding popular files already tend to be overloaded due to the processing of many file requests. Keeping track of the query forwarding rate generates more load on the servers, making the servers more likely to be overloaded. This will result in more replicas since overloaded servers tend to replicate their files in other nodes to release their load. Second, in the *ClientSide* file replication methods that replicate files in requesters [5], [6], the replicas have low utilization. A replica is used only when the replica node queries the file of the replica again, and the replica cannot be shared by other requesters. If a replica is not visited frequently, it has low utilization and hence cannot take over much load from the server. Therefore, the load of servers cannot be released by a low-utilization replica as much as a high-utilization replica. Consequently, servers need to generate more replicas to release their load in *ClientSide*. Third, the *ServerSide* methods replicate files near file owners

[2], [3], [4]. Since a server has limited number of neighbors, disseminating its extra load to the neighbors may also overload them. Thus, the overloaded neighbors will make more replicas to make themselves lightly loaded. Fourth, the *Path* methods replicate a file along a query path from a requester to a owner [16], [7], producing much more replicas. In this case, even though some nodes are not interested in the file or carry little query traffic of the file, they still host replicas of the file. Lastly, most traditional methods did not address the problem of underutilized replicas due to time-varying file popularity and node interest. More replicas consume more storage memory resource and lead to more overhead in subsequent consistency maintenance. In a large-scale P2P file sharing system, efficient consistency maintenance with low overhead is critical to achieving high scalability and efficiency of the system. Therefore, it is important to proactively reduce the number of replicas without compromising the efficiency and effectiveness of file replication in the replication phase.

IRM minimizes the number of replicas while maintaining high efficiency and effectiveness of file replication. First, without arranging the file server to keep track of the query rate of nodes in a centralized manner, IRM enables each node to autonomously keep track of its own load status. Thus, the file server won't be overloaded easily, leading to less replicas. Second, IRM replicates files in nodes with high query passing rate or query initiating rate. This guarantees that a request has high probability to encounter a replica node and every replica is highly utilized. High replica utilization helps to reduce replicas and the overhead of consistency maintenance. Third, IRM doesn't restrict the options of replica nodes to a small number of nodes, which makes it less likely that the replica nodes easily become overloaded nodes. This also helps to reduce replica nodes. Fourth, IRM doesn't replicate file in all nodes along a lookup path, thus avoiding generating redundant replicas and ensuring every replica is highly utilized. Finally, IRM addresses the problem of underutilized replicas by taking into account the time-varying file popularity and node interest. It arranges each node to periodically check the utilization of its replicas and remove underutilized replicas. With all these strategies, IRM produces much less replicas than the traditional file replication methods while still keeping high utilization of replicas. This significantly reduces the overhead of replica update in consistency maintenance phase, and hence enhances the scalability and efficiency of P2P file sharing systems.

3.3.2 The Impact of Consistency Maintenance on File Replication

Most of traditional consistency maintenance methods arrange the server to send update messages to replica nodes based on a structure or message spreading. In P2P dynamism and replication dynamism where nodes and replica nodes join, leave the system, and fail continuously and rapidly, the structure maintenance will generate high overhead especially in high dynamism. More importantly, the structure may not be able to recover in time which may lead to unsuccessful update notification. For example, in a tree structure [11], [13], [14], if a parent leaves or fails, its children cannot receive the update until the broken link is recovered. Message spreading methods will generate high volume of messages flooding in the network. The methods make nodes easily to be overloaded and only provide partial consistency. Consistency maintenance should help to guarantee the fidelity of replica consistency with low overhead.

In IRM, a replica node polls the file server for update without the need to construct and maintain a structure. It avoids the high cost due to structure construction and maintenance, and meanwhile prevents from generating redundant messages. In addition, with polling, the situation that some replica nodes cannot receive the update will not occur. In other words, all replica nodes can get the update successfully. Furthermore, IRM consistency maintenance is not negatively affected by P2P dynamism and replication dynamism. Although nodes or even the replica nodes join, leave, and fail continuously and rapidly, by probing server, every replica node can get the update successfully. Since replica nodes directly poll the server, there is no delay for replica updating because of dynamism. Moreover, a server does not need to keep track of replica nodes for updating and there is no redundant update message. Each node decides whether it needs to poll based on whether itself is a replica node or not.

In addition to reducing cost due to dynamism and redundant messages in the traditional consistency maintenance methods while guaranteeing the fidelity of file consistency, IRM further novelly reduces the consistency maintenance overhead. It looses the requirement of consistency maintenance while still achieving the same effectiveness of consistency maintenance in P2P file sharing. Unlike the previous consistency maintenance methods, which aim to update replicas soon after the original file is updated, the objective of IRM consistency maintenance is to guarantee that a replica file is up to date when being provided. Thus, a replica node does not need to update its replica that won't be queried before its next updating. Recall that for file replication, each node keeps track of a file's query rate for replica creation or deletion. The query rate is also used for consistency maintenance. Based on the query rate, a replica node can know if its replica should be updated or not. Hence, the query rate is measured for dual purposes: file replication and consistency maintenance, which shows the coordination of these two components in IRM.

4 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 even in churn and dramatically reduces consistency maintenance overhead. Table 1 lists the parameters of the simulation and their default values. In practice, a node has various capacities in terms of bandwidth, memory storage, processing speed, etc. We assume that different capacities can be represented by one metric. We assumed bounded Pareto distribution for node capacities. This distribution reflects the real world where

TABLE 1
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 lower bound: 500 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
Observation period	1 second
α	0.5
β	1.5

there are machines with capacities that vary by different orders of magnitude. The values of $\alpha(\alpha > 0)$ and $\beta(\beta > 1)$ were randomly chosen.

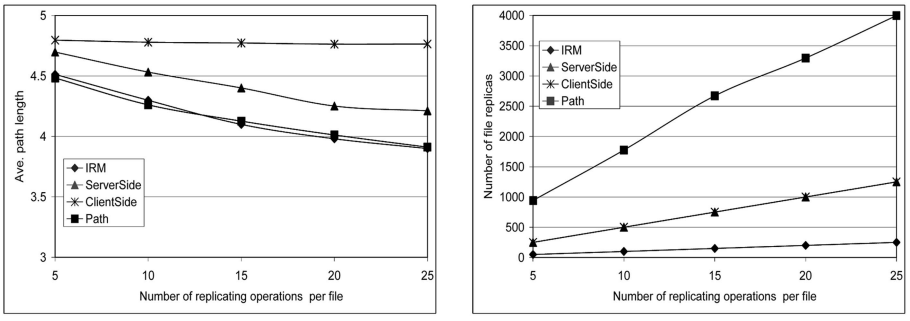
4.1 File Replication

We choose the works in [2], [5], and [7] as representative works of the three categories of file replication approaches, *ServerSide, ClientSide*, and *Path*, respectively. We compared the performance of IRM with *ServerSide* [2], *ClientSide* [5], and *Path* [7] in terms of average lookup path length, hot spot reduction, 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. To make the experiment results of different methods be comparable, we set the same number of replicating operations in IRM, *ServerSide* and *ClientSide*. The values of T_l and T_q were chosen for this purpose. The file requesters and queried file IDs were randomly chosen in the experiment, unless otherwise specified. We use *replica hit rate* to denote the percentage of the number of file queries that are resolved by replica nodes among total queries.

4.1.1 Effectiveness of File Replication Algorithms

Fig. 4a 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. More replicas in lookup path increase the probability that a file query meets a replica during its way traveling towards the file's owner. Therefore, *Path* produces high replica hit rate and shorter path length. However, this is outweighed by its prohibitively high overhead. First, *Path* needs to keep track of a query path so that it can replicate a file along the path. Second, *Path* needs higher cost for file replication than others because it replicates a file in all nodes along a path. Third, *Path* leads to higher overhead for consistency maintenance since it has much more replicas to maintain. Compared to *Path*, IRM can achieve almost the same lookup efficiency but with much less replicas. This shows the effectiveness of IRM to replicate files in nodes with high query initiating rate and query passing rate. Thus, most frequent file requesters can get the replicas of the requested file from themselves and most queries can encounter replica nodes on their way to the file owners. Therefore, IRM 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 cannot 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 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 and lookup path length.

In order to show the effectiveness of file replication methods in reducing hot spots, we randomly generated 100 files in different nodes as very popular files that make their owners become hot spots. We set the number of replicas for each file as five in *ServerSide, ClientSide*, and IRM. We regard load of a hot spot as the number of queries for its popular file. Fig. 5 shows the percent of reduced load in each of the hot spots. The figure demonstrates that *Path* can reduce around 90 percent-100 percent load, IRM can reduce 20 percent-40 percent load, and *ServerSide* can reduce 10 percent-20 percent load. In *ClientSide*, most



(a) (b)

Fig. 4. Performance of file replication algorithms. (a) Average path length and (b) number of replicas.

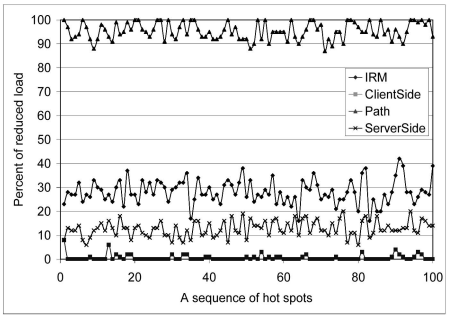


Fig. 5. Effectiveness of hot spot reduction.

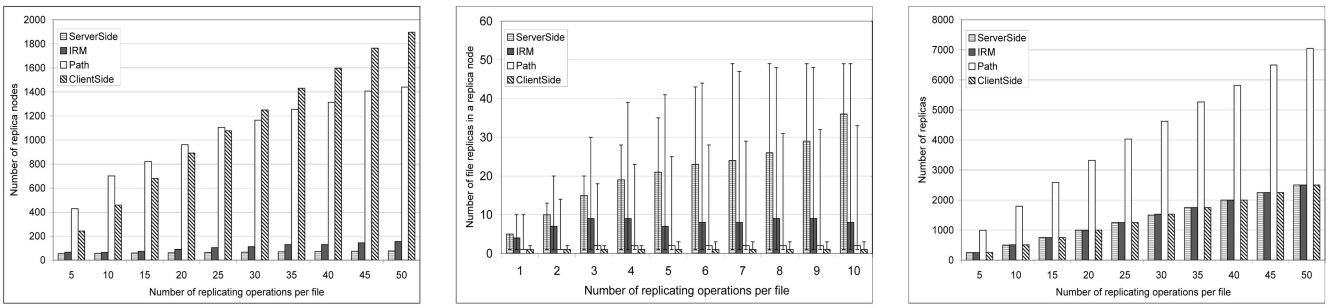
nodes' load is not reduced and some nodes' load is reduced by 1 percent-3 percent. The better performance of *Path* in reducing hot spots is because of its much more replicas. This is confirmed by the experiment results, which show that the total number of replicas in *Path* is 1904, while it is 500 in others. Though *ServerSide*, *ClientSide*, and IRM have the same number of replicas, IRM reduces more load of hot spots than *ServerSide*, while *ClientSide* hardly reduces the load of hot spots. This result implies the effectiveness of IRM by replicating files in traffic hubs. In IRM, queries toward hot spots are answered by the traffic hubs, reducing the load in hot spots. Since only five neighbors of a hot spot have replicas, some queries may not meet replica nodes. As a result, the load of hot spots reduced by *ServerSide* is not as much as IRM. In *ClientSide*, replicas can hardly be shared by other requesters. Thus, most queries are still forwarded to the hot spots, exacerbating the load status of hot spots.

4.1.2 Efficiency of File Replication Algorithms

We set the number of hot files as 10 and tested the total number of replicas at a random time instant. Fig. 4b illustrates the number of replicas versus the number of replicating operations per file. The figure 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 lead to more replicas. The figure also shows that the number of replicas 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* generates much more replicas, and it needs much higher overhead for file replications and subsequent consistency maintenance. We can observe that IRM generates less replicas than *ServerSide* and *ClientSide*. IRM adjusts the number of file replicas adaptively based on file query initiating rate and query passing rate such that less popular or infrequently requested files have less file replicas. When a replica node no longer requests a file frequently or forwards a file query frequently, it will delete its replica. This reduces underutilized replicas and guarantees that all replicas have high utilization. The results confirm the effectiveness of IRM on removing unnecessary file replicas and keeping replicas worthwhile. From the perspective of file consistency maintenance, IRM saves consistency maintenance overhead by not having to maintain replicas for infrequently requested files.

4.1.3 Distribution of File Replicas

In this experiment, we test the distribution of file replicas in different file replication algorithms. To generate the 50 IDs of files to lookup, we first randomly chose an ID in the P2P ID space and then used its succeeding IDs as the lookup file IDs. Since the file owners locate successively in a small ID interval, the query traffic for the files flows toward the interval. We measured the number of replica nodes by regarding a node holding at least one replica as one replica node. Fig. 6a plots the number of replica nodes in different file replication algorithms. The figure shows that *ServerSide* generates less replica nodes than IRM, which produces significantly less replica nodes than *Path*, and *Path* produces fewer replicas than *ClientSide*. *ServerSide* replicates a file in the neighbors of the file's owner. Since the file owners in this experiment are neighbors, replica nodes are neighbors close to the file owner. Thus, it is very likely that many different files are replicated in the same node. It restricts the scope of replica nodes to a small group of neighbors of file owners. Therefore, *ServerSide* generates the least number of replica nodes. In contrast to *ServerSide, ClientSide* replicates a file at the clients that spread all over the network. It means that replicas are scattered over the network, which makes it less likely that different files share the same replica node. Consequently, *ClientSide* produces dramatically more replica nodes than *ServerSide*. Recall that *Path* replicates a file in multiple nodes along a path and others replicate a file in a single node in one replication operation. It is surprising to see that *Path* leads to less



(a) (b) (c)

Fig. 6. Distribution of file replicas in different file replication algorithms. (a) Number of replica nodes, (b) number of replicas in a node, and (c) total number of replicas.

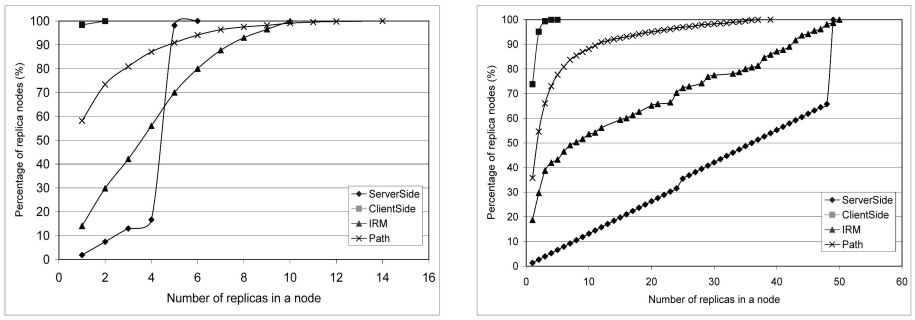


Fig. 7. CDF of total replica nodes in different file replication algorithms. (a) Five replicating operations per file and (b) 50 replicating operations per file.

replica nodes than *ClientSide* when the number of replicating operations per file is more than 30. *Path* replicates a file in nodes along lookup path, and *ClientSide* only generates one replica each time. Therefore, *Path* should generate much more replica nodes than *ClientSide*. Because the file destinations are gathered in a small ID space interval, the query traffic flows toward the same direction and different queries will meet at points of juncture. Therefore, many replicas will be placed in the same query forwarding node, especially when much more replicas are produced. This is the reason that *Path* creates less replica nodes than *ClientSide* when there are many replicas. IRM smartly takes advantage of the points of traffic juncture for replica nodes. Hence, many files take the same node as their replica nodes, leading to much less replica nodes. Although IRM produces slightly more replica nodes than *ServerSide*, by disseminating replicas among more nodes, it significantly reduces lookup path, enhances replica hit rate, and achieves more balanced load distribution.

We also recorded the number of replicas in each replica node. Fig. 6b plots the median, 1st and 99th percentile of the number of replicas in a replica node. We can observe that the median number of replicas of *ServerSide* is higher than IRM. The median number of replicas in IRM is higher than *Path*, and *ClientSide* generates the least median number of replicas. The results in the figure are consistent with those in Fig. 6a. With the same total number of replicas in *ClientSide*, *ServerSide*, and IRM, more replica nodes lead to less replicas in a node because the replicas are dispersed over more nodes. Although *ServerSide* generates more replicas, its dramatically large number of replica nodes share the responsibility for holding replicas and contribute to the low median number of replicas in a node.

We can also observe that the 1st percentiles of the number of replicas in all algorithms remain at one, the 99th percentiles of the number of replicas in *ServerSide* and IRM are higher than *Path*, and that of these algorithms are considerably higher than *ClientSide*. Due to the small set of nodes for replicating in *ServerSide*, different files share the same node for replicas, resulting in high 99th percentile. In IRM, since a file is replicated in a traffic hub and the query traffic for different files will pass through the same traffic hub, a replica node has a number of different replicas, leading to high 99th percentile. As explained that in *ServerSide*, many replicas will be in the same query forwarding nodes. Thus, *ServerSide* also generates high 99th percentile of the number of replicas. The notably low 99th percentile result of *ClientSide* is due to its widespread replica nodes and replicas. It is less likely that a replica node holds different replicas when replicas' nodes disperse over the network, leading to low replicas in a replica node. The results imply that *ServerSide*, IRM, and *Path* have high replica utilization, while *ClientSide* has low replica utilization though it is less likely to overload replica nodes. Without taking node load status into account, *ServerSide* and *Path* tend to generate load imbalance and overload replica nodes. In IRM, a file is replicated in a node with sufficient capacity to hold the replica and process its requests. Thus, IRM avoids load imbalance and meanwhile reduces the overhead for consistency maintenance by replicating files in traffic hubs.

Fig. 6c demonstrates the total number of replicas in different replication algorithms. In this experiment, we did not remove replicas in replica adaptation in IRM in order to make the test results be comparable between the algorithms. The figure shows that *ServerSide, ClientSide*, and IRM have the same number of total replicas, and *Path* generates much more replicas than these algorithms. It is because in one replicating operation, *Path* replicates a file in a number of nodes along a lookup path, while other algorithms only replicate a file in one node. The results imply that *Path* brings about much more overhead due to much more replica nodes. Moreover, more replica nodes lead to more overhead in the subsequent file consistency maintenance.

The results in Fig. 6a, 6b, and 6c confirm the motivation of this work. *ServerSide* tends to overload the replica nodes near the file owner, resulting in more replicas. *ClientSide* disseminates replicas widely in the network. Its low replica utilization leads to more replicas. *Path* generates many replicas by replicating a file along a lookup path. Therefore, a high effective and efficient file replication algorithm like IRM is needed. It reduces replicas, maintains high replica utilization, and avoids overloading replica nodes. Less replicas help to reduce the overhead in subsequent consistency maintenance.

4.1.4 Distribution of Replica Nodes

Fig. 7a shows the cumulative distribution function (CDF) of the percentage of total replica nodes versus the number of replicas in a node when the number of replicating operations

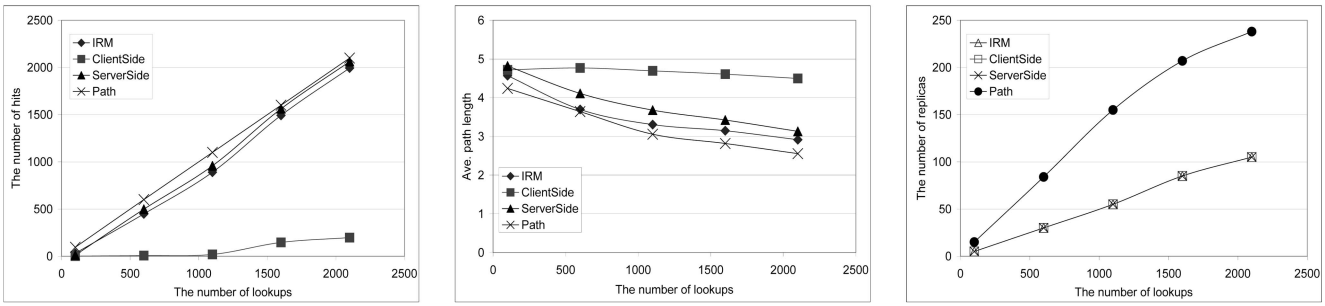


Fig. 8. The effect of file popularity on file replication methods. (a) Number of hits, (b) average path length, and (c) total number of replicas.

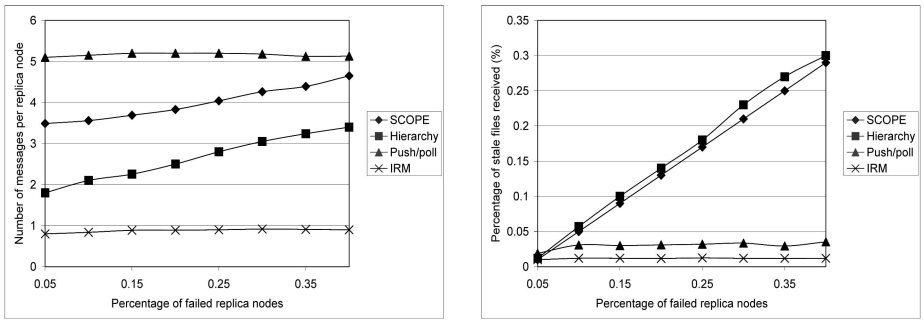
per file is five. In *ClientSide*, 2 percent of replica nodes have two replicas and 98 percent of replica nodes have only one replica. This result is in expectation because *ClientSide* replicates a file in nodes widely spread over the entire network. Only when a node requests more than one different files frequently, it has more than one replicas. In *Path*, the percent of nodes holding one replica is 58 percent, the percent of nodes holding two replicas is approximately 15 percent, and that of three or more replicas is less than 7.5 percent. *Path* replicates a file in nodes along a lookup path and has more replicas. Thus, the replicas are hosted in many nodes that forward requests to their destination. Since one file has five replicas, many replica nodes have only one copy. In IRM, replicas are more evenly distributed. Each group of nodes with $x(x \leq 5)$ replicas constitutes around 14 percent of the total replica nodes. The group of nodes with six and seven replicas constitutes around 10 percent and 8 percent of the total replica nodes, respectively. Thus, replicating files in traffic hubs with node load status consideration helps to distribute the replication overhead in balance. The curve of *ServerSide* jumps when the number of replicas in a node is five. The figure shows that 98 percent of replica nodes have five replicas. The results mean that by depending on a small set of owner neighbors for file replication when there are many requests for files in close owners, the replica nodes will be overloaded by considerable number of replicas.

Fig. 7b shows the CDF of the percentage of total nodes versus the number of replicas in a node when the number of replicas per file is 50. In *ClientSide*, 73 percent of nodes have only one replica and the number of replicas in a node does not exceed five. In *ServerSide*, the percents of nodes having one replica, two replicas, and three replicas are 36 percent, 19 percent, and 11 percent, respectively. Other replica nodes with different number of replicas are almost evenly distributed. In IRM, the percents of nodes having one replica, two replicas, and three replicas are 19 percent, 11 percent, and 9 percent, respectively. Other replica nodes with different number of replicas are also evenly distributed. In *ServerSide*, the percents of nodes with different number of replicas are almost the same except one node which has 49 replicas in the jumping point in the curve. The results mean that many nodes need to hold many replicas. *ServerSide* leads to load imbalance among replica nodes, and it sometimes seriously overloads a few nodes. The relative performance between the replication algorithms is consistent with that in Fig. 7a due to the same observed reasons in Fig. 7a. The results show that IRM distributes replicas in a more balanced manner than other algorithms, leading to less overloaded replica nodes.

4.1.5 The Effect of File Popularity

This experiment tested the performance of the file replication methods with difference degrees of file popularity. We randomly chose one file and a number of nodes to query for this file. The number of lookups is varied from 100 to 2,100 with increase of 500 in each step. More lookups mean higher popularity of the file. Fig. 8a illustrates the number of hits versus the number of lookups. We can observe that *ClientSide* produces dramatically less hits than others. Only when the number of lookups exceeds 1,500, *ClientSide* leads to more than 100 hits. Recall that the replicas in *ClientSide* are useful only when the replica nodes query for the replica file again, and they are hardly shared by other nodes. Therefore, in *ClientSide*, a hit occurs only when a replica node requests its replicas, leading to low total number of hits. Comparing *Path, ServerSide*, and IRM, we can find that *Path* generates a little more number of hits than IRM, and IRM produces slightly less number of hits than *ServerSide*. By replicating the file in all nodes along a path, *Path* increases the probability that a query meets a replica node. *ServerSide* replicates the file in the neighbors of the file's owner. Because a query always passes the file owner's neighbors before arriving at the owner, *ServerSide* leads to more hits. IRM replicates the file in traffic hubs and frequently requesters. It cannot provide hits as high as *ServerSide*. We also observe that the number of hits of the three methods grows sharply as file popularity increases. Higher popularity makes the file owner more easily to be overloaded, resulting in more replicas and hence more hits. More number of hits implies more load is released from the file owner of the popular file. Thus, we can conclude that *Path, ServerSide*, and IRM are highly effective than *ClientSide* in reducing hot spots due to popular file visits.

Fig. 8b shows the average path length of the file replication methods versus the number of lookups. The figure demonstrates that the average path length of *ClientSide* is longer than others, and it maintains almost constant. The reason for this observation is due to the same reason in Fig. 8a. *ClientSide*'s less hits lead to longer lookup path length since less queries can be resolved by replica nodes before they reach the file owner. The average path lengths of *Path, ServerSide*, and IRM decrease as the popularity of the file increases. This is because higher file popularity leads to more replicas and more hits, which results in shorter path length. *ServerSide* produces longer average



(a) (b)

Fig. 9. Performance of file consistency maintenance algorithms. (a) Number of messages with churn and (b) stale file responses with churn.

path length than IRM, and IRM generates marginally longer average path length than *Path*. *Path* has the highest number of hits, hence it has the shortest path length. It is intriguing to see that though IRM has slightly less hits than *ServerSide*, it produces shorter path length than *ServerSide*. This is due to the locations of replicas. The replicas in *ServerSide* are in the neighbors of the file owner. Thus, queries are resolved when they are approaching the file owner. Instead, IRM enables queries to be resolved in the middle of a path by replicating the file in traffic hubs.

Fig. 8c shows the number of replicas in each file replication method. We can see that *ClientSide, ServerSide*, and IRM generate the same number of replicas, while *Path* produces significantly more replicas than others. This is due to the reason that it replicates a file in all nodes along a path. Though it achieves more hits and least average path length, it brings about more overhead of more replicas. It implies the superiority of IRM that achieves slightly less effectiveness than *Path*, but generates much less cost.

4.2 File Consistency Maintenance

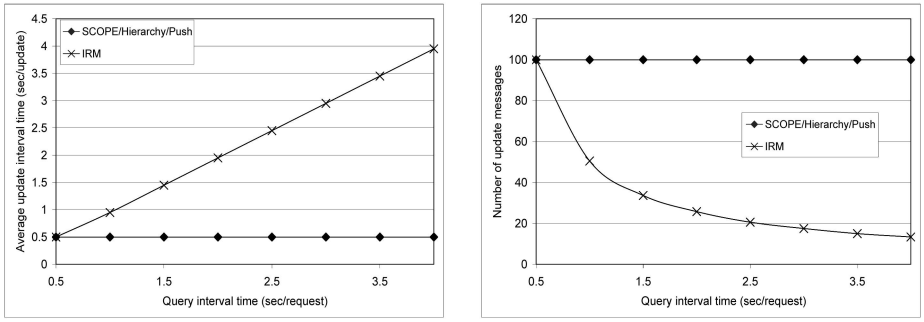
We use *Hierarchy* to denote the work in [10] that builds a hierarchical structure for file consistency maintenance. We compared the performance of IRM with *SCOPE* [11], *Hierarchy* [10], and *Push/poll* [15] methods in terms of file consistency maintenance cost and the capability to keep the fidelity 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 percentage of the files in each category and their update rates were (0.5 percent, 0.15 sec), (2.5 percent, 7.5 sec), (7 percent, 30 sec), and (90 percent, 100 sec). File queries were successively generated. The query interval time was randomly chosen between 1 and 500 seconds.

4.2.1 Efficiency of File Consistency Maintenance

This 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 4,000 and the failed nodes were randomly chosen. Fig. 9a 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*. *SCOPE* constitutes nodes into a tree structure for file updating. *Hybrid* forms nodes into a hierarchical structure. In the lower level of the structure, a group of regular nodes connects to a super node. In the upper level, super nodes constitute a tree structure for file updating. Therefore, when a replica node fails, *SCOPE* and *Hybrid* need to recover their tree or hierarchical 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. Without the need of maintaining a structure, IRM and *Push/poll* do not need extra messages for structure recovery and update retransmission. However, the number of update messages of *Push/poll* is much higher than other schemes. *Push/poll* relies on message spreading, which means that its churn resilience is achieved at the cost of high node communication overhead. IRM enables replica nodes to poll the file owner approximately at the rate of file update rate. Polling will not generate update failures and hence update retransmissions. Thus, IRM does not produce more update messages except polling message. Without redundant messages and structure maintenance, IRM achieves churn resilience at a dramatically lower cost.

4.2.2 Effectiveness of File Consistency Maintenance

In file consistency maintenance, if a replica is not updated in a timely fashion, file requesters may receive stale files. The efficiency of file consistency maintenance schemes is important for keeping file replicas' temporal consistency. This experiment tested the effectiveness of file consistency maintenance methods in maintaining the fidelity of file consistency. Fig. 9b depicts the percentage of stale files received by requesters versus the percentage of failed nodes. We can see that the percentages of stale files received in *SCOPE* and *Hierarchy* increase rapidly as the failed replica nodes grow, while the percentages of stale files received in IRM and *Push/poll* keep almost constant regardless of the percentage of failed replica nodes. The figure also demonstrates that *SCOPE* and *Hierarchy* incur much higher percentage rates than IRM and *Push/poll*. *SCOPE* relies on tree structure for update propagation, and if a node fails, all the node's children cannot get the update message in time until the tree is fixed. *Hierarchy* is a hierarchical structure with super nodes on the upper level and regular nodes on the lower level. A cluster of regular nodes connects with a super node, and super nodes constitute a tree structure. Therefore, if a super node fails, this super node's child super nodes and their child's regular nodes are unable to receive the updated message successfully. Update failures make some file requesters receive stale files. In contrast, IRM and *Push/poll* do not depend on a structure for file update. They respectively use



(a) (b)

Fig. 10. Effectiveness of IRM in overhead reduction. (a) Update rate and (b) number of update messages.

autonomous polling and rumoring that are highly resilient to churn. The figure also shows that *Push/poll* 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. In addition, an update message may traverse a number of hops before it arrives a replica node. Such indirect updating cannot provide timely update. Therefore, it is very likely that a replica node cannot provide up-to-date file upon a request in *Push/poll*. Moreover, the high churn resilience of *Push/poll* is outweighed by its high cost of redundant update messages. In IRM, direct updating by polling the file owner provides quick replica update once a file is update. Furthermore, 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 regard to the consistency maintenance overhead and the capability to guarantee fidelity of consistency in churn.

4.2.3 Overhead of File Consistency Maintenance

Recall that IRM considers file query rate to reduce the number of file updates in file consistency maintenance. This experiment tested the effect of IRM in poll reduction. Specifically, we tested the update rate and the number of update messages versus file query rate. We chose a file and set its update rate to 0.5 second per update. We varied the query rate of a replica of this file from 0.5 to 4 seconds per query with 0.5 second increment in each step. The query rate changes every 5 seconds. That is, the replica is queried every 0.5 second at first, and after 5 seconds, i.e., after 10 queries, it is queried every 1 second, and so on. The update rate is set to 0.5 second per update. Fig. 10a shows the average update rates of the replica in each consistency maintenance algorithm with different file query rates. We can observe that the update interval time of *SCOPE*, *Hierarchy*, and *Push/poll* remains at 0.5, while that of IRM increases as the query interval time increases. In *SCOPE*, *Hierarchy*, and *Push/poll*, a file owner sends updates to replica nodes once the file is changed. Thus, the replica is updated at the rate of the file update rate regardless of the replica's query rate. IRM employs polling for replica update. When a replica's query rate is slower than its update, the replica node polls the file owner according to its query rate. This is based on the fact that even though the replica is updated, if there is no query for it, the update is not necessary. As a result, the replica's update interval time increases as its query interval time grows. We find that the replica's update interval time doesn't equal to its query interval time. This is because there is a latency before the replica node observes the query rate change.

Fig. 10b plots the number of update messages versus the replica query interval time. In this test, the query rate changes every 50 seconds. The figure shows that the number of messages in *SCOPE*, *Hierarchy*, and *Push/poll* remains at 100, while that of IRM decreases as the replica's query interval time increases. Because the file is updated every 0.5 second, thus there are $50/0.5 = 100$ updates during the 50 second period in these algorithms, resulting in 100 update messages. Unlike these algorithms, IRM adapts its update rate to replica query rate in consistency maintenance. Slow query rate leads to slow update rate by polling. Therefore, IRM's update rate decreases with the increase of the query interval time. Hence, the number of IRM's update messages decrease as the query interval time increases. The experiments confirm the effectiveness of IRM in reducing consistency maintenance overhead by flexibly adjusting a replica's update rate to its query rate.

5 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 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 replicas, high query efficiency and fidelity of consistency. Meanwhile, 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.

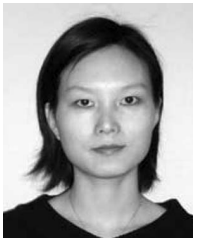
We find that IRM relying on polling file owners still cannot guarantee that all file requesters receive up-to-date files, although its performance is better than other consistency maintenance algorithms. We plan to further study and explore adaptive polling methods to fully exploit file popularity and update rate for efficient and effective replica consistency maintenance.

ACKNOWLEDGMENTS

This research was supported in part by US National Science Foundation (NSF) grants CNS-0834592 and CNS-0832109. An early version of this work [34] was presented in the Proceedings of ICCCN '08.

REFERENCES

[1] R. Kumar, J. Liang, and K.W. Ross, "The FastTrack Overlay: A Measurement Study," *Computer Networks,* vol. 50, no. 6, pp. 842-858, 2006.
[2] A. Rowstron and P. Druschel, "Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility," *Proc. ACM Symp. Operating Systems Principles (SOSP),* 2001.
[3] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stocia, "Wide Area Cooperative Storage with CFS," *Proc. ACM Symp. Operating Systems Principles (SOSP),* 2001.
[4] T. Stading, P. Maniatis, and M. Baker, "Peer-to-Peer Caching Schemes to Address Flash Crowds," *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2002.
[5] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher, "Adaptive Replication in Peer-to-Peer Systems," *Proc. 24th Int'l Conf. Distributed Computing Systems (ICDCS),* 2004.
[6] Gnutella Home Page, http://www.gnutella.com, 2008.
[7] R. Cox, A. Muthitacharoen, and R.T. Morris, "Serving DNS Using a Peer-to-Peer Lookup Service," *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2002.
[8] 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 Trans. Networking,* vol. 11, no. 1, pp. 17-32, Feb. 2003.
[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM,* pp. 329-350, 2001.
[10] G. Xie, Z. Li, and Z. Li, "Efficient and Scalable Consistency Maintenance for Heterogeneous Peer-to-Peer Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 19, no. 12, pp. 1695-1708, Dec. 2008.
[11] X. Chen, S. Ren, H. Wang, and X. Zhang, "SCOPE: Scalable Consistency Maintenance in Structured P2P Systems," *Proc. IEEE INFOCOM,* 2005.
[12] P. Druschel, M. Castro, A.-M. Kermarrec, and A. Rowstron, "Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," *IEEE J. Selected Areas in Comm.,* vol. 20, no. 8, pp. 1489-1499, Oct. 2002.
[13] M. Roussopoulos and M. Baker, "CUP: Controlled Update Propagation in Peer to Peer Networks," *Proc. USENIX Ann. Technical Conf.,* 2003.
[14] L. Yin and G. Cao, "DUP: Dynamic-Tree Based Update Propagation in Peer-to-Peer Networks," *Proc. 21st Int'l Conf. Data Eng. (ICDE),* 2005.
[15] A. Datta, M. Hauswirth, and K. Aberer, "Updates in Highly Unreliable, Replicated Peer-to-Peer Systems," *Proc. 23rd Int'l Conf. Distributed Computing Systems (ICDCS),* 2003.
[16] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Proc. Int'l Workshop Design Issues in Anonymity and Unobservability,* pp. 46-66, 2001.
[17] J. Lan, X. Liu, P. Shenoy, and K. Ramamritham, "Consistency Maintenance in Peer-to-Peer File Sharing Networks," *Proc. Third IEEE Workshop Internet Applications (WIAPP),* 2003.
[18] M. Theimer and M. Jones, "Overlook: Scalable Name Service on an Overlay Network," *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS),* 2002.
[19] K. Huang, T. Huang, and J. Chou, "LessLog: A Logless File Replication Algorithm for Peer-to-Peer Distributed Systems," *Proc. 18th Int'l Parallel and Distributed Processing Symp. (IPDPS),* 2004.
[20] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* 2000.
[21] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. 16th Int'l Conf. Supercomputing (ICS),* 2001.
[22] E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks," *Proc. ACM SIGCOMM,* 2002.
[23] S. Tewari and L. Kleinrock, "Analysis of Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. ACM SIGMETRICS,* 2005.
[24] S. Tewari and L. Kleinrock, "On Fairness, Optimal Download Performance and Proportional Replication in Peer-to-Peer Networks," *Proc. IFIP Networking,* 2005.
[25] S. Tewari and L. Kleinrock, "Proportional Replication in Peer-to-Peer Network," *Proc. IEEE INFOCOM,* 2006.
[26] D. Rubenstein and S. Sahu, "Can Unstructured P2P Protocols Survive Flash Crowds?" *IEEE/ACM Trans. Networking,* vol. 13, no. 3, pp. 501-512, June 2005.
[27] Q. Yang, W. Xiao, and J. Ren, "PRINS: Optimizing Performance of Reliable Internet Storages," *Proc. 26th Int'l Conf. Distributed Computing Systems (ICDCS),* p. 32, 2006.
[28] H. Shen, "EAD: An Efficient and Adaptive Decentralized File Replication Algorithm in P2P File Sharing Systems," *Proc. Eighth Int'l Conf. Peer-to-Peer Computing (P2P '08),* 2008.
[29] D. Tsoumakos and N. Roussopoulos, "APRE: An Adaptive Replication Scheme for Unstructured Overlays," *Proc. 14th Int'l Conf. Cooperative Information Systems (CoopIS),* 2006.
[30] M. Raunak, P. Shenoy, B. Urgaonkar, A. Ninan, and K. Ramamritham, "Maintaining Mutual Consistency for Cached Web Objects," *Proc. 21st Int'l Conf. Distributed Computing Systems (ICDCS),* 2001.
[31] A. Muthitacharoen, B. Chen, and D.M. Eres, "A Low-Bandwidth Network File System," *Proc. ACM Symp. Operating Systems Principles (SOSP),* pp. 174-187, 2001.
[32] rsync, http://en.wikipedia.org/wiki/Rsync, 2009.
[33] W.R. Stevens, *TCP/IP Illustrated,* vol. 1, Addison-Wesley, 1994.
[34] H. Shen, "IRM: Integrated File Replication and Consistency Maintenance in P2P Systems," *Proc. IEEE 17th Int'l Conf. Computer Comm. and Networks (ICCCN),* 2008.



Haiying (Helen) Shen received the BS degree in computer science and engineering from Tongji University, China, in 2000, and the MS and PhD degrees in computer engineering from Wayne State University in 2004 and 2006, respectively. She is currently an assistant professor in the Department of Electrical and Computer Engineering, and the director of the Pervasive Communications Laboratory of Clemson University. 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 mining. Her research work has been published in top journals and conferences in these areas. She was the program cochair for a number of international conferences and member of the Program Committees of many leading conferences. She is a member of the IEEE, the IEEE Computer Society, and the ACM.