GeWave: Geographically-aware Wave for File Consistency Maintenance in P2P Systems

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

Abstract

File consistency maintenance in P2P systems is a technique for maintaining consistency between files and their replicas. Most traditional consistency maintenance methods depend on either message spreading or structure for update propagation by pushing. Message spreading generates high overhead due to redundant messages, and cannot guarantee that every replica node receives an update. Structure-based pushing methods reduce the overhead but cannot guarantee timely consistency in churn. Moreover, most methods are unable to consider physical proximity to improve efficiency. To further reduce update overhead, enhance guarantee of consistency, and take proximity into account, this paper presents a geographically-aware Wave method (GeWave). Depending on adaptive polling in a dynamic structure, GeWave avoids redundant file updates by dynamically adapting to time-varying file update and query rates, and ensures the consistency of query results even in churn. Furthermore, it conducts update propagation between geographically close nodes in a distributed manner. Simulation results demonstrate the efficiency of GeWave in comparison with other representative consistency maintenance schemes. It dramatically reduces the overhead and yields significant improvements on efficiency and scalability of file consistency maintenance schemes.

1. Introduction

Over the past years, the immense popularity of Internet has produced a significant stimulus to peer-to-peer (P2P) file sharing systems. A recent survey [18] shows more than 75% of Internet traffic is generated by P2P file sharing systems. The percentage has increased significantly as P2Ps become more and more popular. To improve P2P system performance, file replication and caching are widely used. Shared files are replicated on several nodes to improve system reliability and availability. Moreover, query results are always cached along the routing path to reduce the query latency of subsequence queries.

Although files in some P2P file sharing systems (i.e. KaZaA [9] and Morpheus [13]) are always consistent, other P2P systems (i.e. OceanStore [10] and Publius [28]) permit users to modify their files, leading to inconsistency between a modified file and its replicas. On the other hand, with the tremendous development of P2P applications, newly-developed P2P applications require frequent content updates, such as trust management [33], remote collaboration, bulletin-board systems, and e-commerce catalogues. Therefore, file consistency maintenance to maintain the consistency between a file and its replicas is essential to improve service quality of existing P2P applications, and to meet the basic requirement of newly-developed P2P applications. Without effective consistency maintenance, a P2P system is limited to providing only static or infrequently-updated file sharing.

Due to the large-scale and churn of P2P systems, a highly-scalable and churn-resilient consistency maintenance method is highly demanded. In addition, the method is also required to have: (1) consistency guarantee of query results; (2) low overhead; (3) fast update propagation.

Centralized scheme is a straightforward way to maintain replica consistency. However, it lacks of scalability and suffers from single-point of failure. Some current decentralized consistency maintenance methods rely on message spreading [6, 11]. They generate very high overhead because of overwhelming number of messages. Furthermore, they cannot guarantee that all replica nodes can receive the update. To reduce their overhead, other decentralized methods depend on a structure [12, 2, 15, 31] to propagate updates by pushing. These methods produce less update messages, but node failures will break the structure intactness, and hence lead to unsuccessful update propagation. Update failure aside, these methods generate high overhead for structure maintenance, especially in churn. In addition,

most methods don't consider physical proximity in file update processing, though message transmission between geographically (physically and geographically are interchangeable terms in this paper) close nodes can improve update efficiency in terms of propagation overhead and speed.

To improve the consistency guarantee, enhance the efficiency, and reduce the overhead of current methods, this paper presents a geographically-aware Wave method (GeWave). In GeWave, each replica node has a polling frequency for its replica that ensures the consistency of the replica when requested. GeWave builds structure, in which a node's parent has higher polling frequency, and is geographically close to it. Nodes update their files by polling their parents in a top-down fashion. Due to its wavy pattern among geographically close node, we name this method as geographically-aware wave. GeWave structure is resilient to P2P churn and the creation and deletion of replica nodes.

In most current methods, all replica nodes get informed soon after a file is updated. The ultimate goal of consistency maintenance is to guarantee that query results are not out-of-date. Therefore, rather than trying to achieve strong consistency of replicas, GeWave novelly aims to achieve the consistency of query results by avoiding unnecessary updates. A significant feature of GeWave is that it achieves an optimized tradeoff between overhead and fidelity of consistency guarantees. Specifically, GeWave owns the following distinguishing features.

(1) *Consistency guarantee.* It uses adaptive polling to ensure that all replica nodes get updates, and to enhance the consistency guarantee of query results even in churn.

(2) *Low overhead.* It avoids redundant file updates by dynamically adapting to time-varying file update and query rates, and it generates much less overhead for structure maintenance.

(3) *Fast update.* It speeds up updates by conducting update propagation between geographically close nodes.

(4) *High scalability.* It conducts consistency maintenance in a decentralized manner, and a node receives update directly without relay nodes.

(5) *Churn-resilient.* It ensures that a replica node can always get an update even in churn.

The rest of this paper is structured as follows. Section 2 presents a concise review of representative file replication and consistency maintenance approaches. Section 3 and Section 4 present the overview and detailed design of GeWave consistency maintenance scheme. Section 5 shows the performance of GeWave in comparison with other approaches. Section 6 concludes this paper.

2. Related Work

Replication and caching have been widely deployed in P2P systems [14, 16, 5, 21, 25, 7, 4, 23, 24, 17, 8, 21, 3]. Most of these systems resort to a centralized method to maintain consistency, which is not scalable for P2P systems. For higher scalability, a number of file consistency maintenance methods have been proposed. One class of methods is based on message spreading. Lan and *et al.* [11] proposed to use flooding-based push for static files and polling for dynamic files. In hybrid push/poll algorithm [6], flooding is substituted by rumor spreading to reduce communication overhead. A replica node randomly selects a set of replica nodes and forwards the message to them with a probability. When a new node joins or a node gets connected again, it contacts online replica nodes to poll updated content. However, hybrid push/poll scheme only offers probabilistic guarantee of replica consistency. While suitable for unreliable P2P systems with churn, spreading-based category generates high propagation overhead due to redundant update messages. It cannot guarantee that an update reaches every replica node. In addition, not considering proximity prevents it from achieving higher efficiency.

Another class of methods is pushing based on structures. Li and *et al.* [12] presented a scheme that forms the replica nodes into a proximity-aware hierarchical structure (UMPT): the upper layer is P2P and a node in the lower layer attaches to a physically close node in the upper layer. An update tree is built dynamically when the upper layer propagates update messages. Though it takes proximity into account, clusters of physically close nodes are fragile in churn, which may lead to update propagation failures. Moreover, periodical message exchange for cluster maintenance leads to high overhead. SCOPE [2] builds a replica-partition-tree for each key based on its original P2P system. It keeps track of the locations of replicas and then propagates updates. CUP [15] and DUP [31] propagates an update along a routing path. In FreeNet [3], an update is routed to other nodes based on key closeness.

However, structure-based pushing methods also have a number of problems. First, they still generate redundant messages. Some non-replica nodes as in SCOPE, and some replica nodes that don't necessarily need updates due to absence of queries still receive update messages. Second, decentralized pushing cannot guarantee all replica nodes get the update with node departures and failures, hence cannot always ensure the consistency of query results. Third, proximity unawareness in some methods prevents further improvement in consistency maintenance efficiency.

The consistency of web proxy caching is studied in [27, 30, 26]. In their context, the proxies are always available. Thus, these methods are not applicable for a P2P dynamic environment. GeWave shares similarity with the

work in [26, 11, 19] in terms of polling employment. However, GeWave improves consistency guarantee in churn. It considers physical proximity to improve consistency maintenance efficiency. More importantly, rather than polling a single file owner, GeWave conducts consistency maintenance in a decentralized manner, thus achieving higher scalability.

3. Overview

In this section, we present a brief overview of GeWave, deferring a detailed description to the next section.

GeWave employs polling for consistency maintenance mainly due to four reasons. First, polling helps reduce redundant updates. Since the ultimate goal of consistency maintenance is to offer up-to-date files, a node only needs to guarantee the updated status of its replica when responding to a query. Thus, a node does not necessarily need to update its replica soon after each update. Rather, it can poll the update based on its query rate. Instead of passively accepting updates, polling enables a node to actively adjust its update rate to reduce unnecessary updates. Second, pushing does not provide a way for a node to know whether its replica is up-to-date when the replica is queried. Active polling allows a replica node to hold the query during polling until it is certain that the replica is up-to-date, thus enhancing the fidelity of consistency guarantees of query results. Third, polling facilitates to enhance consistency guarantee in churn. Instead of passively waiting for updates which may lead to update failures due to node failures and departures, a node can actively poll to get timely update. Fourth, unlike decentralized pushing in which an update usually passes through a number of relay nodes before it reaches a replica node, polling enables an update be transmitted directly to a replica node, thus helping to guarantee timely update of all replicas.

In GeWave, each replica node has a periodical polling time interval for its replica, denoted by TTR. GeWave forms replica nodes of a file into a structure based on their TTR values and their physical locations. An example of GeWave structure is shown in Figure 1. The root node is the file owner, and the replica nodes are organized in ascending order of their TTR levels from the top to the bottom. That is, the polling rate of nodes in the d^{th} level is faster than that of nodes in the $(d+1)^{th}$ level. The children are physically close to their parents. In each level, a node connects to its predecessor and successor, and is relatively physically close to them.

The file update is conducted in the fashion of a wave. After the original file is updated, the nodes in the first level will poll the file owner for updates before their replicas are queried. Later, the nodes in the second level will poll the nodes in the first level before their replicas are queried, and so on. Therefore, a file's updating is like a wave from the top to the bottom between geographically nodes. By this way, GeWave is distinguishing in terms of a number of features: (1) Relying on polling with query rate consideration rather than updating all replicas once a file changes, GeWave significantly reduces the updates and overhead, and meanwhile provides enhanced consistency guarantee of replicas when queried. (2) Depending on active polling rather than pushing, GeWave lets a replica node respond to a query only when the replica is up-to-date, which improves the guarantee that all query results are current. (3) Polling within geographically close nodes dramatically improves the efficiency of consistency maintenance. (4) Relying on decentralized polling instead of centralized polling, GeWave achieves high scalability by distributing the overhead among replica nodes. (5) Compared to structured-based pushing methods, polling avoids update failures due to node departures and failures, and helps to ensure file consistency even in churn.

For simplicity, we assume that only the file owner can modify the file, or replica nodes are allowed to modify file but need to notify the file owner. The techniques for file replication are orthogonal to our study in this paper.

4. The Design of GeWave

4.1. Adaptive Polling

GeWave uses adaptive polling in which a replica node polls another node having an updated file to determine if its replica is stale. GeWave is developed based on the work in [19] that considers the scenario where replica nodes directly poll the file owner for update. We describe the polling method in the following.

The polling method lets a replica node dynamically vary its polling frequency based on the update rate of the file, i.e. frequently modified files are polled more often than relatively static files. A time-to-refresh (TTR) value is associated with each replica that represents the time between two successive polls. The polling begins using a TTR value of $\triangle$. It then uses a *linear increase multiplicative decrease (LIMD)* algorithm [26] to adapt the TTR value (and hence, the polling frequency) to the file update rate. This approach gradually increases the TTR value as long as there is no staleness, and reduces TTR value on detection of staleness. Thus, it probes the file owner for the rate at which the file changes, and sets the TTR value accordingly.

The ultimate objective of file consistency maintenance is to ensure a replica is not outdated when requested. Therefore, it is not necessary to update all file replicas once a file is changed. A consistency maintenance method is effective as along as files are up-to-date when requested. Even when a file changes frequently, if a replica node hardly receives

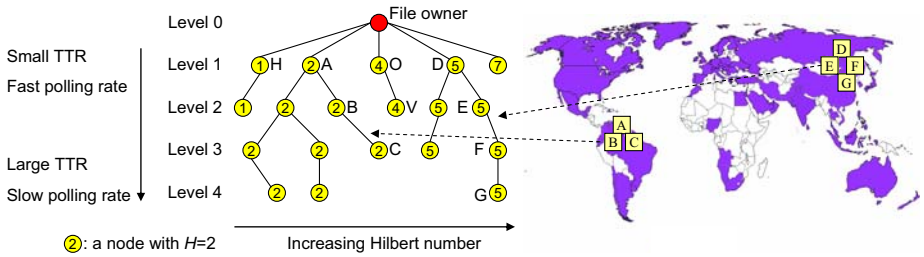


Figure 1. *An example of a four-level GeWave structure. Neighbor links in each level are omitted for simplicity. In vertical direction, nodes are ordered in descending order of their polling rate. In horizontal direction, nodes are ordered in increasing order of their Hilbert number. Physically close nodes are connected in both vertical and horizonal directions. Nodes in the $(d+1)^{th}$ level poll nodes in the d^{th} level for file updates.*

queries for the file or hardly queries for the file, it is a overhead waste to poll for consistency maintenance. Based on this, the polling method takes file query rate into account to further reduce the overhead of consistency maintenance. We use T_{query} to denote the time interval between two successive queries.

A node combines file query rate into consideration for TTR determination. That is:

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

For more details of the TTR determination, please refer to [19].

Polling helps GeWave enhance the consistency guarantee of query results. If a node receives a request during file polling process, the node holds the request until it receives the update message. By this way, GeWave provides enhanced guarantee that all query results are current even in churn.

Poll-based GeWave distributes the burden of consistency maintenance among individual nodes. In structure-based pushing and message spreading based methods, an update usually passes a number of relay nodes before arriving at a distant replica node. In contrast, using direct node-to-node communication, GeWave enables distant nodes to achieve consistency and is less sensitive to churn, network size and P2P structure. Typically, the polling method of GeWave has an advantage that it avoids providing stale file by active polling. In contrast, in push-based methods, a replica node passively receives update message, which increases the possibility of providing an outdated file, especially in a dynamic environment where nodes join, leave and fail.

4.2. Geographically-aware Wave

All replica nodes polling a file owner for consistency maintenance may overload the file owner, leading to delayed update message response. We propose a TTR-based and proximity-aware GeWave structure to conduct file consistency maintenance in a decentralized manner. Moreover, GeWave structure guides update messages to travel between physically close nodes to significantly improve the consistency maintenance efficiency.

Before we present the details of the GeWave structure, let us introduce a landmarking method to represent node closeness on the Internet by indices. Landmark clustering has been widely adopted to generate proximity information [29, 20]. We assume m landmark nodes that are randomly scattered in the Internet. Each node measures its physical distances to the m landmarks, and uses the vector of distances $< d_1, d_2, \ldots, d_m >$ as its coordinate in Cartesian space. Two physically close nodes have similar landmark vectors. We use space-filling curves [1], such as Hilbert curve [29], to map m-dimensional landmark vectors to real-numbers, called Hilbert numbers. That is, $R^m \longmapsto R^1$, such that physically close nodes will have similar Hilbert numbers. We use H_i to denote the Hilbert number of node i.

A GeWave structure is built dynamically based on the node geographical location and polling rate. It takes the file owner as its root, and constitutes nodes into different levels based on polling rate levels; nodes in the upper levels having high polling frequency than those in the lower levels. It connects geographically close nodes to enhance file update efficiency. An updated is propagated in the fashion of top-down wave between geographically close nodes.

A GeWave structure has L levels, which is determined by a file owner according to actual file query rate and estimated number of replica nodes. For instance, if the possible TTR is within [0,100] seconds and $L = 10$, then the nodes with TTR in [0,10) will be on level 1, the nodes with TTR in [10,20) will be on level 2, and so on. We use L_{TTR} to denote a node's TTR level. L should be an appropriate value. If L is too large, it will adversely affect the effectiveness of the GeWave structure for file updating. If it is too small, it may lead to load imbalance.

As shown in Figure 1, the root node is the file owner, and the replica nodes are organized in ascending order of their TTR levels from the top to the bottom. That is, the first level nodes are the nodes that have the least TTR, and the

TTR level of the d^{th} level nodes is smaller than that of the $(d+1)^{th}$ level nodes. In addition to children, most nodes have three links: a parent link and two neighbor links (i.e. predecessor and successor). The root node does not have a parent link and the first node and last node in each level only have one neighbor. The parent and neighbors of a node are physically close to it. For example, nodes A, B and C are physically close nodes, and node A is relatively close to its neighbors H and O. During file updating, after A polls the file owner, B polls A for update, and after that, C polls B for update.

The parent assignment algorithm in GeWave guarantees with high probability that a child's parent always has up-to-date file before the child polls it for update, and children are physically to their parent. Communication between geographically close nodes reduces the bandwidth consumption, and improves efficiency. From the perspective of the entire GeWave structure, the update process is modelled as a wave from the top to the bottom. Such a decentralized pattern significantly improves the scalability of the polling method.

4.3. Construction and Maintenance of GeWave Structure

In the following, we introduce how to build and maintain the GeWave structure in terms of the creation, departure and failure, and mobility of replica nodes.

Replica node creation. A file owner collects the TTR and H of its replica nodes into its directory when creating replicas. When a file owner creates replica node i, it compares node i's file query rate and the file's update rate to determine node i's TTR, and then its TTR level in the GeWave structure. Assume node i's TTR level is d, the file owner finds node i's parent in its directory, whose H is closest to H_i in the group of $L_{TTR} = d-1$. In addition, the file owner finds node i's predecessor and successor in the d level nodes using H_i. Its predecessor and successor may have the same H as itself, since nodes may have the same H. If node i is the first node or the last node in level d, it only has one neighbor. The file owner sends the information of the parent and neighbors to node i. Node i then begins periodically polling its parent at its TTR.

Recall that a child's parent is its physically closest node in the upper level. After node i joins in the GeWave structure in the d level, some nodes in the level $d+1$ may be physically closer to node i than to their current parents. Therefore, node i contacts its neighbors, which transfer their children to node i that are closer to node i. For example, in Figure 1, before node O joins in the GeWave structure, node V is connected to node D. After node O becomes a replica node, since V's Hilbert number is closer to O's, i.e. V locates physically closer to O, node V will be transferred to O.

Replica node departure and failure. A node leaves the GeWave structure when it leaves the P2P system, or it is no longer a replica node of the file. Moreover, a node could fail or depart without warning. In both cases, GeWave structure needs to be reorganized.

When a file owner removes file replica in node i, node i leaves the GeWave structure as a voluntary leaving node. Before a node leaves, it needs to notify its neighbors. Upon receiving a departure notifications, the leaving node's predecessor and successor become neighbors. In addition, the leaving node transfers its children to its neighbors based on the closeness of Hilbert numbers. For example, before node O leaves, it moves node V to D. To save overhead for structure maintenance, GaWave does not rely on periodical "keep-alive" message exchange to deal with node failure. It replies on backup owner-polling method. Specifically, GeWave sets a certain period of time T. If a leaving node has not received response from its neighbors during T after sending out departure notifications, it contacts the root for its new neighbors, then notifies the neighbors and transfers its children before leaving.

To handle the problem of node failures, GeWave also replies on backup owner-polling method. If a node does not get response from its parent within T after polling, it assumes that its parent has left or failed, and resorts to the backup method in which it polls the file owner for update message, and meanwhile locates a new parent. Thus, GeWave does not need periodical message exchange to keep nodes' links up-to-date for structure maintenance, which brings about high overhead. Dynamic GeWave structure construction and recovery makes GeWave highly resilient to churn, while still proving high guarantee of consistency.

Replica node mobility. A node's TTR is not constant, thus its position in the GeWave structure should change correspondingly. A simple method to cope with node mobility is to execute a leave operation followed by a join operation. However, relying on root for position location in join operation may overload the root. GeWave deals with node mobility in a decentralized manner. If node i's TTR is increased, it first executes a leave operation. It then traverses upwards along the up links until reaching the level of its new TTR. Then, it traverses in the horizontal direction based on its Hilbert number, until arriving at the position where its successor's H is no less than itself, and its predecessor's H is no larger than itself. The process for TTR decrease is the same except that node i first traverses along the one of its child links.

To avoid being overloaded in dealing with node failures, a root can have a number of backups. Specifically, it chooses a number of nodes in level 1, whose polling rates are the highest. It copies its directory with collected TTR and H information to those nodes. When its load reaches

αC ($\alpha < 1$), where α is a percentage factor and C is its capacity, it transfers requests to its backups in a round-robin fashion.

Summary. In GeWave structure, each node always has a physically closest parent whose polling frequency is faster than it. This ensures that a node can get an update at a low overhead. Each node also has relatively physically close neighbor(s) whose polling rate is at the same level as itself. This helps to enhance the efficiency of structure maintenance when replica nodes join and leave.

Unlike traditional centralized polling in which a file owner is responsible for all update responses, GeWave distributes the update overhead among the replica nodes. It also dramatically improves consistency maintenance efficiency in terms of overhead and speed since the update messages are transferred between physically close nodes. GeWave is distinguished from the current tree-based file consistency maintenance methods [12, 2] in that it reduces structure maintenance overhead in churn by avoiding periodical message exchange. In addition, it has high capacity to deal with churn by resorting file owner polling. In contrast, in a tree structure, when a parent leaves, its children cannot get timely update messages until the tree is repaired.

5. Performance Evaluation

We designed and implemented a simulator in Java for evaluation of the *GeWave* based on Chord [22] with 4096 nodes. We compared the performance of *GeWave* with *SCOPE* [2], *UMPT* [12] and *Push/poll* [6] methods, in terms of proximity-aware performance, consistency maintenance cost, and the capacity to deal with churn. In *UMPT*, 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%, 0.15 sec), (2.5%, 7.5 sec), (7%, 30 sec) and (90%, 100 sec). The file query rate of each replica file was randomly generated at the rate of 0-500 seconds per query. We varied the number of replica nodes from 500 to 4000 with step size of 500.

We used transit-stub topology generated by GT-ITM [32]: "ts5k-large". It has 5 transit domains, 3 transit nodes per transit domain, 5 stub domains attached to each transit node, and 60 nodes in each stub domain on average. "ts5k-large" is used to represent a situation in which Chord consists of nodes from several big stub domains. To account for the fact that interdomain routes have higher latency, each interdomain and intradomain hop counts as 3 and 1 hops of latency units respectively.

5.1. Proximity-aware Performance

Figure 2(a) shows the cumulative distribution function (CDF) of the percentage of total update messages versus physical distance in "ts5klarge". We can see that *GeWave* and *UMPT* are able to transmit around 90% of total messages within 10 hops, while *SCOPE* and *Push/poll* move only up to 30% within 10 hops. Almost all messages in *GeWave* and *UMPT* travel within 15 hops, while 80% messages in *SCOPE* and *Push/poll* travel within 15 hops. The results show that most messages in *GeWave* and *UMPT* travel in short distances, while most messages in *SCOPE* and *Push/poll* travel in long distances. The more messages travelling in the shorter distances, the higher proximity-aware performance of a consistency maintenance method. The results indicate that proximity-aware schemes *GeWave* and *UMPT* are efficient in guiding update messages to travel between physically close nodes. Hence, they lead to faster and low-overhead update propagation.

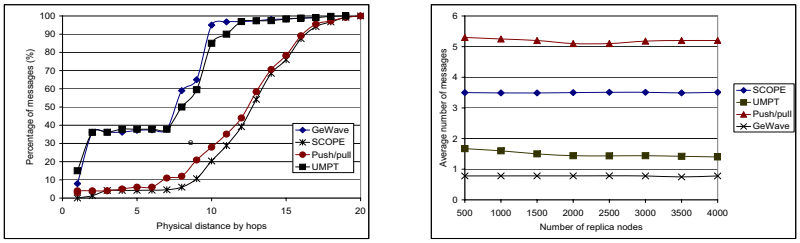
5.2. Consistency Maintenance Cost

Figure 2(b) illustrates the average number of messages per replica node for an update operation of a file. Messages for structure building are also included. From the figure, we can see that the *Push/poll* scheme has the highest number of update messages. It is because a replica node may receive multiple update messages from several other replica nodes. *SCOPE* leads to messages less than *Push/poll*, but twice more than that of *UMPT*. This is mainly because *SCOPE* constructs a tree with all nodes in the system, while *UMPT* builds a tree only with replica nodes of a file. Therefore, more messages are needed for tree construction in *SCOPE*. *GeWave* produces the least average number of messages. Recall that *GeWave* reduces messages for updating infrequently-queried file. Thus, GeWave is more cost-effective than others in terms of the number of update messages.

5.3. Performance in Churn

This experiment tests the capability of different consistency maintenance methods to deal with churn in P2P systems. In the experiment, the number of replica nodes was set to 4000, the percentage of failed replica nodes was varied from 5% to 35% with step size of 5%, and the failed nodes were randomly chosen.

Communication cost constitutes a main part of file consistency maintenance overhead. The cost is directly related with message size and physical path length of the message travelled. We use the product of these two factors of all messages to represent the communication cost. It is assumed that the size of a message is 1 unit. Figure 3(a) shows the



(a) CDF in "ts5k-large" (b) Ave. num. of messages

Figure 2. Performance of file consistency maintenance schemes.

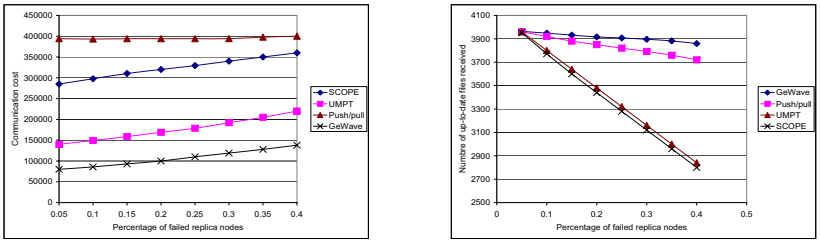
communication cost versus the percentage of failed replica nodes. We can see that the communication cost increases as the percentage of failed replica nodes increases in all methods except *Push/poll*. The failures of replica nodes have little effect on *Push/poll* due to its message spreading. However, its communication cost is still much higher than other schemes due to its message spreading. We can also observe that *GeWave* and *UMPT* incur much less communication cost than *SCOPE* and *Push/poll*. *SCOPE* and *Push/poll* do not take proximity into account in update propagation, resulting in long message travel distances. Moreover, considerably more messages further increase communication cost. In contrast, *GeWave* and *UMPT* consider proximity by guiding update message to travel between physically close nodes.

In addition to the proximity, churn is another factor affecting the communication cost. *SCOPE* and *UMPT* use periodical message exchange for structure maintenance, which generate more much messages. In addition, they need more messages for structure recovery, leading to the increase of the messages. *GeWave* incurs much fewer messages than *SCOPE* and *UMPT*. Unlike *SCOPE* and *UMPT* that need periodical message exchange, and need multiple messages for structure recovery in a node failure, *GeWave* only needs one more polling. Re-polling is the reason that *GeWave* incurs slightly higher messages as failed replica nodes increase. Thus, less messages in *GeWave* helps to further reduce its communication cost. Consequently, *GeWave* outperforms others in terms of communication cost.

If a replica is not updated timely, file requesters may receive stale files. The total number of file requests was set to 4000. Figure 3(b) depicts the number of up-to-date files received by requesters versus the percentage of failed nodes. It shows that *SCOPE* and *UMPT* lead to less number of up-to-date files received 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. Since *SCOPE* constitutes all nodes rather than only replica nodes in a tree, more replica nodes will not receive update timely than *UMPT*. In contrast, *GeWave* and *Pull/push* do not depend on a structure. They use polling and message spreading respectively that are highly resilient to churn. We can also observe that the number of up-to-date files received in *Pull/push* is marginally lower than *GeWave*. In *GeWave*, when a replica node receives a request during polling, it can wait until it receives the update message before it replies to the requester. If a node's parent leaves, the node can poll the file owner for update message instantly. Therefore, it always enhances consistency of files when queried. However, depending on message spreading, *Pull/push* cannot ensure that every replica node receives an update. Furthermore, the high churn-resilience of *Pull/push* is outweighed by its high cost of redundant update messages. The figure also demonstrates that *GeWave* still returns some stale files though the number of stale files is very small. This is due to the reason that some files may suddenly change, but replica nodes still poll update based on their usual change rate.

6. Conclusions

Traditional file consistency maintenance methods relying on structure-based pushing lead to high cost for structure construction and maintenance, and they cannot guarantee consistency in P2P churn. Other methods depending on message spreading gain high churn-resilient capacity at the cost of redundant messages. Furthermore, most current methods do not take physical proximity into account. In addition, decentralized push cannot guarantee timely update of all replicas due to indirect communication. In this paper, we propose a GeWave file consistency maintenance scheme that conducts consistency maintenance efficiently. Without relying on a static structure, GeWave is highly resilient to P2P churn by adaptive polling with direct node-to-node communication. It avoids unnecessary file updates by dynamically adapting to time-varying file update and query rates. Furthermore, it transmits update messages in a decentralized manner between physically nodes. Simulation results demonstrate the effectiveness of GeWave in comparison with other consistency maintenance approaches. Its low overhead, high efficiency and churn-resilience are particularly attractive to the deployment of large-scale and dynamic P2P file sharing systems.



(a) Communication cost (b) The num. of up-to-date file responses

Figure 3. Performance of file consistency maintenance schemes with churn.

Acknowledgments

This research was supported in part by the Acxiom Corporation.

References

[1] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier. Space filling curves and their use in geometric data structure. *Theoretical Computer Science*, 181(1):3–15, 1997.

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

[3] 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.

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

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

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

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

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

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

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

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

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

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

[14] 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.

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

[16] 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.

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

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

[19] H. Shen. IRM: Integrated File Replication and Consistency Maintenance in P2P Systems. In *Proc. of ICCCN*, 2008.

[20] H. Shen and C.-Z. Xu. Hash-based Proximity Clustering for Efficient Load Balancing in Heterogeneous DHT Networks. *JPDC*, 2008.

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

[22] 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.

[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. Proportional Replication in Peer-to-Peer Network. In *Proc. of INFOCOM*, 2006.

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

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

[27] P. S. V. Duvvuri and R. Tewari. Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web. In *Proc. of IEEE INFOCOM*, 2000.

[28] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. of the 9th USENIX Security Symposium*, 2000.

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

[30] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Hierarchical cache consistency in a WAN. In *Proc. of USITS*, 1999.

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

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

[33] R. Zhou and K. Hwang. PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing. *IEEE Transactions on Parallel and Distributed Systems*, 2008.