Consistency Maintenance In Peer-to-Peer File Sharing Networks[*]

Jiang Lan, Xiaotao Liu, Prashant Shenoy and Krithi Ramamritham[†]
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
jiang.lan@labs.gte.com, {xiaotaol,shenoy,krithi}@cs.umass.edu

Abstract

While the current generation of peer-to-peer networks share predominantly static files, future peer-to-peer networks will support sharing of files that are modified frequently by their users. In this paper, we present techniques to maintain temporal consistency of replicated files in a peer-to-peer network. We consider the Gnutella P2P network and present techniques for maintaining consistency in Gnutella even when peers containing replicated files dynamically join and leave the network. An experimental evaluation of our techniques shows that: (i) a hybrid approach based on push and pull achieves high fidelity in highly dynamic P2P networks and (ii) the run-time overheads of our techniques are small, making them a practical choice for P2P networks.

1. Introduction

The past few years have seen a dramatic increase in the popularity and use of peer-to-peer (P2P) file sharing networks. Current P2P systems are specifically designed to share static content such as music and video files. The utility of P2P systems goes beyond sharing of static files—future P2P applications (e.g., collaborative P2P applications) can be expected to share dynamic files. In such applications, files may be modified and updated during their lifetime. To handle such dynamic content, P2P networks must evolve from a predominantly read-only system to one where files can be both read and written. Since files may be widely replicated in a P2P system, handling dynamic files requires consistency techniques to ensure that all replicas of a file are temporally consistent with one another.

Consistency techniques have been widely studied in the context of web proxy caches [1, 2]. However, these techniques are not directly applicable to P2P systems due to their dynamic nature—peers can dynamically join and leave the network at any time, making consistency maintenance more challenging than in web environments where proxies are mostly available and failures are rare. Due to this crucial difference, novel consistency techniques specifically designed to handle unavailable peers are required. In this paper, we propose three consistency maintenance techniques for P2P networks. Our first two techniques are based on push and pull, respectively, and have complementary tradeoffs. Our results show that: (i) a flooding-based push approach can provide near-perfect fidelity but has high communication overheads and is suitable only for static networks; (ii) in contrast, a pull-based approach has lower communication overheads and is better suited for dynamic networks but provides weaker guarantees than push. Based on these observations, we propose a hybrid approach that combines the best features of push and pull and attempts to provide good fidelity in highly dynamic networks at a reasonable cost. We propose enhancements to the Gnutella protocol to incorporate our techniques and implement them into the public-source Gtk-Gnutella system. Our experiments show that the hybrid approach can provide good fidelity in highly dynamic environments. Our measurements from the prototype implementation indicate that this fidelity can be provided at a reasonable run-time cost.

The remainder of this paper is structured as follows. We present our consistency maintenance techniques in Section 2. Section 3 presents our experimental results, and finally, Section 4 presents our conclusions.

2. Consistency Techniques for P2P Gnutella File Sharing Networks

Peer-to-peer file sharing networks provide an infrastructure for communities to share storage and popular files. Our work is focused on the widely used decentralized system–Gnutella [4]. In Gnutella, peers are organized into an overlay network, where each peer is connected to some num-

[*]This research was supported by NSF grants CCR-9984030, CCR-0098060, CCR-0219520 and EIA-0080119

[†]Also with the Indian Institute of Technology Bombay

ber of neighboring peers over logical links; neighbors are discovered by running a peer discovery protocol. A peer can join or leave the network at any time. Queries propagate through the overlay network via flooding; the reach of a query message is limited by a time-to-live (TTL) value, which is decremented at each hop. The information (name, size, peer ID, etc) of matching files are transmitted back to the initiator through the reverse path. Files are download using the HTTP protocol.

With this background, we now present techniques for maintaining consistency of replicated files in a P2P network. We assume that each file in the system has a unique owner. Modifications to a file can only be made by its owner. While this assumption may seem overly restrictive, it is not—any user (peer) may modify a file, but upon doing so, it is required to transmit these modifications to the owner to "commit" the changes. This ensures that the owner always has the most up-to-date version of the file at all times. Each file is also associated with a version number; the version number is incremented by the owner upon each update. Since a file may be arbitrarily replicated at different peers, upon a modification, the replicas will need to be made consistent with the version at the owner peer. We present three different techniques for doing so in the remainder of this section.

2.1. Push: Owner-initiated Consistency

In the owner-initiated approach, the owner broadcasts an invalidation message upon each update to a file (alternately, the new version of the file may be broadcast). The broadcast message propagates through the P2P overlay like query or ping messages—the owner forwards the message to its neighbors, who then propagate the message to their neighbors and so on until the TTL limit is reached. Upon receiving an invalidation message, a peer checks its shared cache and invalidates the file if the version number of the cached copy is smaller than the version number specified in the invalidation message.

The main advantage of such a push-style approach is its simplicity and stateless nature. The limitation though is that the broadcast nature of the technique increases the control message overhead substantially, especially for objects cached only at a few peers. While a push based approach is suitable for a static P2P network, the following limitations arise in dynamic networks:

1. Not all the peers in the network may receive the broadcast messages, since network dynamics can temporarily partition or increase the diameter of the network.

2. Peer that are disconnected don't receive invalidation messages and will share stale files upon reconnection.

Based on the above observations, we conclude push alone is not sufficient for maintaining consistency in a large-scale Gnutella network. Next, we present a pull-based approach for maintaining consistency.

2.2. Pull: Peer-initiated Consistency

A pull approach puts the burden of consistency maintenance on individual peers. In this approach, a peer polls the owner to determine if a file is stale. Different policies can be used to determine when and how frequently to poll the owner; we outline one such policy in this section. Unlike push, distant peers can still achieve good consistency in the pull approach and the approach is less sensitive to the choice of TTL values, network size and the connectivity of a peer [11].

The following information must be stored at each peer for pull-based consistency maintenance: (i) the version number and/or last modification time of a file, (ii) the IP address of the owner and (iii) the consistency status. The consistency status of a file can take one of three values: (i) *valid*, indicating the file is consistent with the version at the owner, (ii) *stale*, indicating that the file is older than the version at the owner, and (iii) *possibly stale*, indicating that the file could possibly be stale but the peer is unable to determine the actual status since the owner peer is unavailable (i.e., has left the P2P network). Observe that a pull-based approach is more resilient to dynamic joins and leaves. Upon rejoining the network, a peer can poll the owners of all cached files to check if these files were updated in the interim, and thereby ensure consistency of shared files.

Adaptive Polling: In this policy, a peer dynamically varies the polling frequency based on the update rate for the file—frequently modified files are polled more frequently than relatively static files. The notion of adaptive polling has been explored in the context of web cache consistency [6, 7] and we use a similar idea here.

A time-to-refresh (TTR) value is associated with each file. The TTR denotes the next time instant the peer must poll the owner, and thus, determines the polling frequency. The TTR value is varied dynamically based on the results of each poll message. The TTR value is increased by an additive amount if the file doesn't change between successive polls. In the event the file is updated since the last poll, the TTR value is reduced by a multiplicative factor. In essence, an additive increase multiplicative decrease (AIMD) algorithm is used to probe for the update rate. A key advantage of the technique is that it can adapt to changing update rates by recomputing the TTR value after each poll.

To precisely define this technique, if a file has not changed between two polls, we set

$$TTR = TTR_{old} + C \tag{1}$$

where C, $C > 0$, is an additive constant. If the file was modified, then

$$TTR = TTR_{old}/D \tag{2}$$

where D, $D > 1$ is the multiplicative decrease constant. After the above computation, the TTR is bound by a maximum and minimum value to prevent the TTR from becoming very large or very small, both of which can be problematic. Thus,

$$TTR = \max(TTR_{min}, \min(TTR_{max}, TTR)) \tag{3}$$

This TTR value is used to determine the time of the next poll. Such an adaptive TTR technique has the following advantages: (1) A peer can tune parameters C and D to determine how quickly the TTR is increased or decreased after each poll; (2) Only the most recent TTR and the last modification time (i.e., version number) needs to be stored with each file, which result in a very small per-file state space overhead; and (3) The technique can handle dynamic joins and leaves. Upon rejoining the network, the peer simply resets the TTRs of all cached files to TTR_{min}.

2.3. Hybrid Push and Adaptive Pull Technique

A push-based technique can provide good consistency guarantees for peers that are online and reachable from the owner. Pull, on the other hand, is better suited for dynamic networks but provides weaker guarantees—the consistency guarantees in pull are crucially dependent on the effectiveness of polling. Push can be combined with the adaptive pull approach in a hybrid technique that combines the best features of the two approaches.

The push part of the hybrid approach works exactly as the invalidation-based push technique. In addition, the hybrid technique requires peers to occasionally poll the owner to check whether the file was updated. Generally, it is difficult to achieve the ideal scenario where only peers who miss an invalidation message poll the owner, but we can modify the adaptive pull technique to make the polling less aggressive. Less aggressive polling reduces wasted polls from peers who are within reach of the owner. We make the following modifications to the adaptive pull technique:

In addition to adapting the TTR to the update rate, we take into account the number of active neighbors of a peer when computing the TTR. In general, a peer should poll more frequently when the network sees frequent joins and leaves, since frequent changes to the overlay topology increases the probability of missing an invalidate message. Similarly, the peer should poll less frequently when the network is stable. We use the number of active neighbors of a peer as an indicator of the network dynamics. Suppose that a peer has N_{conn} active connections to its neighbors and let $N_{avgconn}$ denote the average connectivity of a peer in the network (most P2P systems use $N_{avgconn}$ as a predefined parameter to ensure good connectivity—upon joining, a peer attempts to create logical links to these many other peers). In such a scenario, the TTR is chosen more aggressively when the number of neighbors drops below average and is made larger when a peer is well-connected and has more neighbors than the average peer.

Thus, after computing the TTR in Equation 2, the TTR is further tuned as

$$TTR = TTR + (1 + \frac{N_{conn} - N_{avgconn}}{N_{avgconn}}) \times \alpha \tag{4}$$

where α is a constant. The TTR is decreased if the peer has a small number of neighbors and increased otherwise. Like before, this TTR value is constrained by the maximum and minimum allowable TTR values TTR_{max} and TTR_{min}.

The TTR value can also be updated upon receiving a push-based invalidation message. The peer can mark the stored copy as stale and decrease the TTR based on Equation 2. This TTR is used for future polls if the file is subsequently refreshed by the user.

By combining push and poll, the approach is able to provide good consistency guarantees to reachable peers, while accommodating the needs of distant peers via pull. Further, our modified TTR computation technique can adapt the TTR value to network dynamics and poll more frequently in the presence of frequent joins and leaves. We have modified the Gnutella protocol to incorporate these consistency techniques; implementation details and implementation experiments may be found in [11].

3. Experimental Evaluation

In this section, we demonstrate the efficacy of our techniques using simulations. We first present our experimental methodology and then our experimental results.

3.1. Experimental Methodology

Simulation Environment: We have designed an event-based simulator to evaluate our cache consistency techniques. Our simulator simulates an overlay network of Gnutella peers. We borrow heavily from recent measurements studies [8, 9, 10] to initialize various simulation parameters such as link bandwidths, network diameter, node connectivity, session times, file popularity, etc. The default values of various parameters used in our simulations is listed in Table 1.

While measurements of query rates and file downloads are available from recent studies, realistic distributions of file update rates are not available since current P2P systems only share static files. Consequently, we use update distributions of web pages [1, 2] to be a reasonable indicator

Parameter	Description	Default Value
L_{sim}	Length of simulation	10 hours
F_{enable}	This flag turns on/off the failure mode	[FALSE, TRUE]
R_f	Percentage of maximum offline nodes	50%
I_f	Average time between successive disconnections	5 seconds
D_f	Average offline duration	2 hours
$I_{topochk}$	Average time between successive topology checks	5 minutes

Table 1. Parameters for the dynamic network environment

of updates in P2P environments. Specifically, we assume four types of files: highly mutable, very mutable, mutable, and immutable. Each category has a different mean update rates. The percentage of the files in each category and the mean update rate in each category is (0.5%, 15 sec), (2.5%, 7.5 min), (7%, 30 min), and (90%, 1 day).

Performance Metrics: We use a metric called the false valid ratio to evaluate various consistency techniques. The *false valid ratio* is the fraction of query responses or downloaded files that return a stale file (i.e., are falsely reported as valid by their peers). The *query false valid ratio (QFVR)* is the fraction of query responses that list stale files. The *download false valid ratio (DFVR)* is the fraction of the downloaded files that are stale. The false valid ratio for queries and downloads should be as close to zero as possible.

Simulated Network Environment: Our simulations are performed in a dynamically changing P2P network. We assume peers leave and rejoin the network randomly, based on the three parameters R_f, I_f, and D_f, as defined earlier. This situation is close to the Gnutella network in use today. Since the memory and computation overhead required to run large-scale simulations are prohibitive. Unless specified otherwise, we assume a smaller network consisting of 500 peers and 5000 objects. While a 500 peer network is small for many P2P studies, it is adequate for studying consistency techniques.

In a dynamic network, peers will frequently leave and rejoin the network. When a peer leaves the Gnutella network, it tears down all connections to its neighbors. This causes each of its neighbor to lose one of their active connections. In the scenario where many peers leave the network, the network may become partitioned. To overcome this drawback, actual Gnutella implementations [12, 5, 13] let a peer form new links with other active peers if a neighbor leaves the network. To simulate this behavior, we implement a topology checking process that periodically checks the connectivity of each peer and constructs new logical links if a peer has fewer neighbors than a threshold.

3.2. Simulation Results

3.2.1. Impact of Interval Between Successive Topology Checks. In this section, we study the effects of topology checking process by varying the $I_{topochk}$ values from 5 seconds to 170 seconds. This parameter effectively determines the delay between a broken connection and the instant when a new connection is formed by a peer. Observe that this parameter only impacts push, since other experiment in [11] showed that the connectivity of the overlay does not impact pull. Hence, we focus on the push and the hybrid push-pull approach. Figure 1(a) and 2(a) show the query FVR and the download FVR for the two approaches. As shown, the longer the delay for replacing broken links with new neighbors, the worse the performance of push. In contrast, the hybrid approach is unaffected by the topology changes, since the approach can resort to pulls when invalidates do not reach a peer.

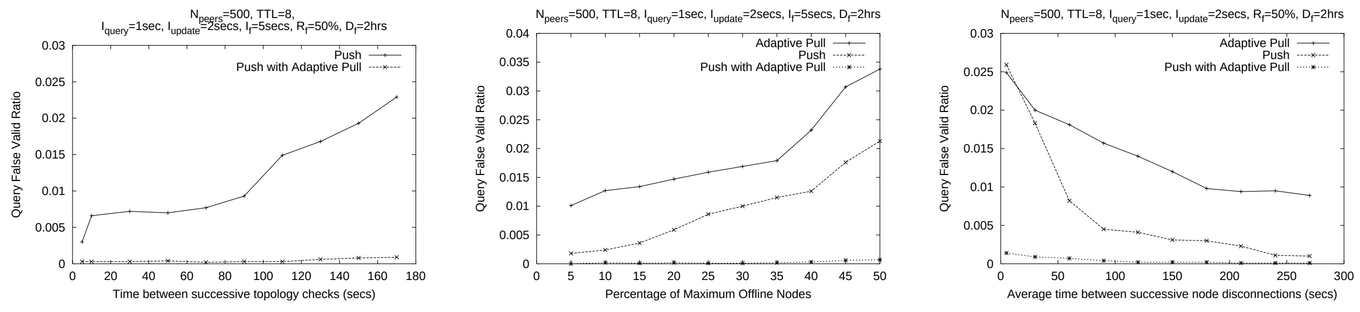
3.2.2. Impact of the Dynamics of the Gnutella Network. To understand the effects of the dynamics of the Gnutella network, we first vary the fraction of offline peers R_f from 5% to 50% and measure the impact on the QFVR and DFVR. As shown in Figure 1(b) and 2(b), push can provide better consistency guarantees than a pure pull approach, even in a dynamic network. The FVRs degrade as the fraction of offline nodes increases. However, the hybrid approach outperforms both push and pull, since it employs a combination of the two and can employ pull in scenarios where push is ineffective. The approach can provide good fidelity and is relatively unaffected even when the fraction of offline nodes is as high as 50%.

Next, we vary the time between successive disconnections I_f. Intuitively, as I_f increases, fewer nodes leave the network. The results are similar to the previous scenario (see Figure 1(c) and 2(c)). Push outperforms a pure-pull approach; both techniques yield better consistency guarantees in more stable networks. Like before the hybrid approach performs well and is relatively unaffected by the dynamics of the network.

Overall, our results demonstrate that a hybrid push-pull approach works well in highly dynamic P2P networks and can provide good fidelity at a cost that is comparable to a pure push approach.

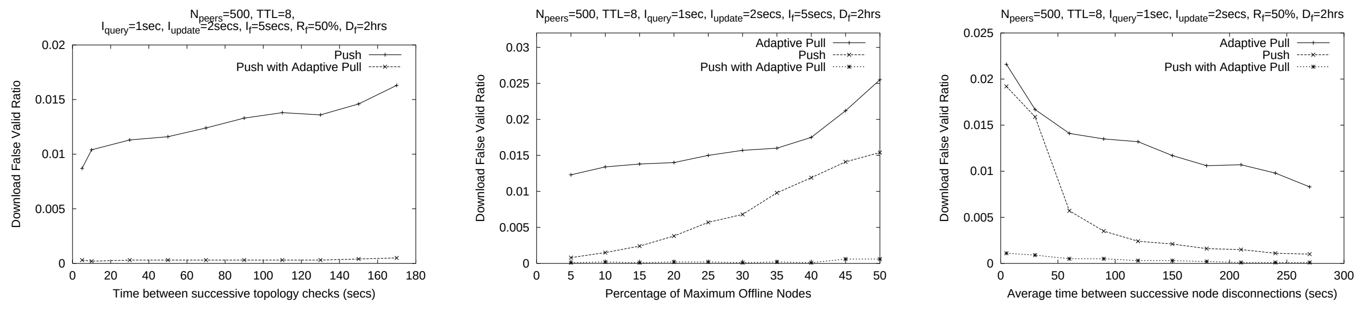
4. Concluding Remarks

While current of peer-to-peer systems share predominantly static files, we argued that future peer-to-peer networks will support sharing of files that are modified frequently by their users. We presented techniques to maintain temporal consistency of replicated files in a peer-to-peer network. We considered Gnutella and presented techniques



(a) Topology Checks (b) Maximum Offline Nodes (c) Node Disconnections

Figure 1. Query False Valid Ratio



(a) Topology Checks (b) Maximum Offline Nodes (c) Node Disconnections

Figure 2. Download False Valid Ratio

for maintaining consistency in Gnutella even when peers containing replicated files dynamically join and leave the network. We presented extensions to the Gnutella protocol to incorporate our consistency techniques and implemented them into a Gtk-Gnutella prototype. An experimental evaluation of our techniques showed that: (i) a hybrid approach based on push and pull achieves high fidelity in highly dynamic P2P networks and (ii) the run-time overheads of our techniques are small, making them a practical choice for P2P networks.

References

[1] V. Duvvuri, P. Shenoy, and R. Tewari. Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web. In *Proceedings of the IEEE Infocom'00, Tel Aviv, Israel*, March, 2000.

[2] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Hierarchical Cache Consistency in a WAN. In *Proceedings of the USENIX Symposium on Internet Technologies (USEITS'99), Boulder, CO*, October 1999.

[3] J Chu, K Labonte, and B Neil Levine, Availability and Locality Measurements of Peer-to-Peer File Systems, *Proc. SPIE ITCom: Scalability and Traffic Control in IP Networks II Conference*, July 2002.

[4] The Gnutella Protocol Specification v0.4, Clips2 Distributed Search Solutions, http://dss.clip2.com.

[5] The KaZaA website, http://www.kazaa.com/

[6] R. Srinivasan, C. Liang, and K. Ramamritham, Maintaining Temporal Coherency of Virtual Data Warehouses, *The 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 2-4, 1998

[7] B. Urgaonkar, A. Ninan, M. Raunak, P. Shenoy and K. Ramamritham, Maintaining Mutual Consistency for Cached Web Objects, In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, Phoenix, AZ, April 2001

[8] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1), 2002.

[9] S. Saroiu, P. Gummadi, and S. Gribble, A measurement study of peer-to-peer file sharing systems, In *Proceedings of Multimedia Computing and Networking 2002*, January, 2002.

[10] K. Sripanidkulchai, The popularity of Gnutella queries and its implications on scalability, Technical Report, February 2001.

[11] J. Lan, X. Liu, P. Shenoy and K. Ramamritham, Cache Consistency Techniques for Peer-to-Peer File Sharing Networks, Technical Report, University of Massachusetts, June 2002.

[12] Limewire web site, http://www.limewire.com/

[13] GTK-gnutella web site, http://gtk-gnutella.sourceforge.net/