Maintaining Data Consistency in Structured P2P Systems

Yi Hu, *Student Member, IEEE,* Laxmi N. Bhuyan, *Fellow, IEEE*, and Min Feng

Abstract—A fundamental challenge of supporting mutable data replication in a Peer-to-Peer (P2P) system is to efficiently maintain consistency. This paper presents a framework for balanced consistency maintenance (BCoM) in structured P2P systems with heterogeneous node capabilities and various workload patterns. Replica nodes of each object are organized into a tree structure for disseminating updates, and a sliding window update protocol is developed for consistency maintenance. We present an analytical model to optimize the window size according to the dynamic network conditions, workload patterns and resource limits. In this way, BCoM balances the consistency strictness, object availability for updates, and update propagation performance for various application requirements. On top of the dissemination tree, two enhancements are proposed: (1) a fast recovery scheme to strengthen the robustness against node and link failures, and (2) a node migration policy to remove and prevent bottlenecks allowing more efficient update delivery. Simulations are conducted using P2PSim to evaluate BCoM in comparison to SCOPE [1]. The experimental results demonstrate that BCoM outperforms SCOPE with lower discard rates. BCoM achieves a discard rate as low as 5% in most cases while SCOPE has almost 100% discard rate.

Index Terms—Peer-to-Peer, consistency, protocol design, simulations.

1 Introduction

Structured P2P systems have been effectively designed for wide area data applications [2] [3] [4] [5] [6] [7]. While most of them are designed for read-only or low-write sharing contents, a lot of promising P2P applications demand support for mutable contents. Such examples are modifiable storage systems (e.g. OceanStore [4], Publius [8]), mutable content sharing (e.g. P2P WiKi [9]), even interactive ones (e.g. P2P online games [10], P2P Social Networking [11], and P2P collaborative workspace [12]). The P2P approach improves data availability, fault tolerance, and scalability for static content sharing. But mutable content sharing raises issues of replication and consistency management. P2P dynamic network characteristics combined with diverse application requirements and heterogeneous resource constraints pose unique challenges for P2P consistency management [13].

P2P systems are typically large, where peers with heterogeneous resource capabilities experience varying network latencies. Also, the frequent joining and leaving of nodes make the P2P overlay failure prone. Neither sequential consistency [14] nor eventual consistency [15] individually works well in a P2P environment. It has been proved [16] that among three properties, atomic consistency, availability and partition-tolerance, only two can be satisfied at a time. Applying sequential consistency leads to prohibitively long synchronization delays due to the large number of peers and the unreliable overlay. Even "deadlock" may occur when a crashed replica node causes other replica nodes to wait forever. Hence, system scalability is restricted due to low data availability resulting from long synchronization delay. At the other extreme, eventual consistency allows replica nodes to concurrently update their local copies, only requiring that all replica copies become identical after a long enough failure-free and update-free interval. Since replica nodes are highly unreliable in P2P systems, the node issuing update may have gone offline by the time update conflicts are detected, leading to unresolvable conflicts. It is infeasible to rely on a long duration without any failure or further updates. As a result, eventual consistency fails to provide any end-to-end performance guarantee to P2P users.

This paper presents a Balanced Consistency Maintenance (BCoM) protocol for structured P2P systems to balance between consistency strictness, object availability for updates, and update dissemination latency. BCoM is designed for P2P implementations of social networking [11] (e.g. Facebook) and collaborative editing [9], [12] (e.g. WiKi or CVS). Users of these P2P applications frequently update common objects. They prefer objects highly available for updating although they can tolerate a certain extent of temporary inconsistency as long as they get the latest version within a time bound. Usually these updates are insertions where conflicts are infrequent.

BCoM protocol serializes all updates to eliminate the complicated conflict handling in P2P systems. It also allows certain obsolescence in each replica node to reduce the update discard rate of implementing sequential consistency. BCoM limits the extent of temporary inconsistency by developing a sliding window update

- *Y. Hu, L. N. Bhuyan, and M. Feng are with the Department of Computer Science and Engineering, University of California at Riverside, 900 University Ave., Riverside, CA, 92521.*
E-mail: yihu@cs.ucr.edu

Digital Object Indentifier 10.1109/TPDS.2012.81

protocol. The size of the sliding window regulates the number of allowable updates buffered by each replica node. Thus, BCoM provides a measure of consistency guarantee which is specified by an application rather than eventual consistency. BCoM develops an analytical model to set the window size as follows: given an inconsistency bound, the window size is set to minimize the update discard rate while ensuring the expected delay is no worse than the baseline by a small given threshold.

Existing bounded consistency techniques for P2P systems can be divided into two categories: probabilistic consistency [17] [18] and time-bounded consistency [19] [20], both of which have limitations. Probabilistic consistency only guarantees consistency on most nodes. It cannot guarantee that every node receives all updates. Thus, it is not applicable to the situation where all intermediate updates are valued by every user. BCoM overcomes this problem by ensuring consistency bounds on every node. Besides, existing probabilistic consistency protocols involve redundant update propagation, which is eliminated in BCoM. Time-bounded consistency sets a uniform temporal constraint on inconsistency for all nodes. In the situation where nodes have various update frequencies, it is impossible to set a temporal constrain that works for all nodes. To solve this problem, BCoM uses a sliding window to directly limit the number of updates that have not been received by all replica nodes.

An update window protocol has been designed for web-server systems [21] to limit the number of uncommitted updates at each replica node. But the authors of [21] do not address update conflicts and potential cascading impacts. Moreover, their window size optimization model requires information on each node. There are two obstacles making it impractical to apply this technique to P2P systems: (1) unlike the web-servers, P2P replica nodes are highly dynamic and unreliable which makes the update conflict problem worse, (2) the number of replicas in P2P systems is orders of magnitude larger than that in web-server systems. Hence, collecting information from each node is infeasible in P2P systems. BCoM develops a sliding window protocol that avoids these obstacles. In BCoM, updates are serialized to avoid conflicts and a distributed analytical model is developed to optimize the window size with simple system-wide information, such as the total layers of replica nodes and the bottleneck service time. This information can be collected periodically with low overhead. Therefore, the consistency maintenance provided by BCoM scales well in dynamic P2P systems.

In BCoM, replica nodes of each object are organized into a d-ary dissemination tree (dDT) to propagate updates. dDT is built on top of the overlay structure, an auxiliary structure for consistency maintenance of an object. We evaluate the efficiency of BCoM with comparison to SCOPE [1], which also builds an auxiliary tree structure on top of the overlay for sending updates. SCOPE proposed an ID partitioning algorithm to construct their update dissemination tree for maintaining sequential consistency in structured P2P networks. There are other tree based consistency management for structured P2P (e.g., [22]), but the tree construction methods fundamentally are inherited from SCOPE. Therefore, we choose to compare the performance of BCoM with SCOPE. In SCOPE, the update dissemination tree is built by recursively partitioning the identifier space and selecting a representative node as the tree node for each partition. The drawback is that a tree node may not be a replica node, thus not all tree nodes are interested in receiving or propagating updates about the object. Including such nodes in the dissemination tree introduces extra overhead for sending updates. The ID partitioning algorithm may also assign a node to be several tree nodes in SCOPE because of its ID. Such nodes may be easily overloaded when sending updates. BCoM avoids these two problems by constructing the update dissemination tree dDT only from replica nodes, with each replica node mapped to a tree node. BCoM also builds a dDT as balanced as possible to reduce the tree height. Smaller tree height reduces the number of hops for update propagation and thus the delay, which improves the object availability for updates.

For each object in BCoM, replica nodes join the dDT of this object through the root node, and all updates about the objects are sent to the root for serialization. However, the root will not be a bottleneck caused by a large number of replicas, as the root only sends updates to its children, who in turn send the updates to their children. The root only has a small constant number of children, and the node degree is independent of the total number of tree nodes (i.e. replicas). The update rates will neither overload the root because the root only serializes the updates it received. No communication overhead is imposed on the root and the computation overhead for serializing updates is negligible for any modern computer. A root node may be overloaded by being root for too many objects. Since the root of an object is selected through hashing the object ID to the node ID in a structured P2P overlay, load balance schemes may solve the problem, which is beyond the scope of this paper.

BCoM presents two enhancements to further improve the performance of a dDT. One is the *ancestor cache* scheme, where each node maintains a cache of ancestors for fast recovery from parent node failures or leaving. This relieves the tree-structure “multiplication of loss” problem [23] (i.e. all the subtree nodes rooted at the crashed node will lose updates), which is especially critical in P2P systems. Maintaining the ancestor cache does not introduce extra overhead since the needed information conveniently piggybacks on update propagation. A small size cache significantly improves the robustness of dDT against node churn and failures. The other is the *node migration* scheme, where more capable nodes are migrated to upper layers and less capable nodes are migrated to lower layers. The reason is that if an upper layer node is slow in propagating updates, the consistency constraint blocks its ancestors from receiving new updates, and all its subtree nodes will not receive timely updates. The node migration scheme is to prevent and remove bottleneck nodes. Two forms of node migration are presented, one is to remove blocking and the other is to prevent blocking.

The contributions of our paper are the following:

- We propose a consistency maintenance framework (BCoM) in structured P2P systems. A sliding window update protocol and two enhancement schemes are presented. BCoM balances consistency strictness, object availability for updating, and update dissemination latency.
- We analyze the problem of setting the window size in response to dynamic network conditions, changing workload patterns, and different resource constraints through a queueing model. Our model serves diverse consistency requirements from various data sharing applications.
- We evaluate the performance of BCoM with comparison to SCOPE [1] using the P2PSim simulation tool. SCOPE is the most relevant work to BCoM, and it is widely studied in structured P2P systems for consistency management.

The rest of the paper is organized as follows: Section 2 describes BCoM techniques and deployment. Section 3 presents the analytical model for window size setting. The performance of BCoM is evaluated in Section 4, and case study results are presented in Section 5. The scholarly literature is reviewed in Section 6. The paper is concluded in Section 7.

Preliminary conference version of this paper was published in [24].

2 Description of BCoM

BCoM aims to: (1) provide bounded consistency for maintaining a large number of replicas of a mutable object; (2) balance the consistency strictness, object availability for updating, and update propagation performance based on dynamic network conditions, workload patterns, and resource constraints; (3) make the consistency maintenance robust against frequent node churn and failures. To fulfill these objectives, BCoM organizes all replica nodes of an object into a d-ary dissemination tree (dDT) on top of the P2P overlay for disseminating updates. It applies three core techniques: the sliding window update protocol, the ancestor cache scheme, and the tree node migration policy on a dDT for consistency maintenance. In this section, we first introduce the dDT structure, and then explain the three techniques in detail.

2.1 Dissemination Tree Structure

For each object, BCoM builds a tree with node degree d rooted at the node whose ID is the closest to the object ID in the overlay identifier space. We denote this d-ary dissemination tree of object i as dDT_i. Each node in dDT_i is a peer who holds a copy of object i. We name such a peer as a "replica node" of i, or simply as a replica node. An update can be issued by any replica node, but it should be submitted to the root. The root serializes updates to eliminate conflicts.

With node churn and failures in P2P systems, a dDT serves two cases of insertions: (1) a single node joining, and (2) a node with subtree rejoining. The goal of constructing a dDT is to minimize the tree height with low overhead in both cases.

We show an example of dDT_i construction with node degree d set to 2 in Figure 1. The replica nodes are ordered by their arrival times as node 0, node 1, node 2, etc. At the beginning, node 1 and node 2 joined. Both were assigned by node 0 (i.e., the root) as its children. Then, node 3 joined. Since node 0 cannot have more child, it passed node 3 to a child who has the smallest number of subtree nodes. Since both children (i.e., node 1 and node 2) had the same number of subtree nodes, node 0 randomly selected one to break the tie, say node 1, and increased the number of subtree nodes at node 1 by one. Node 1 assigned node 3 as its child because it had a space for a new child. When node 4 joined, node 0 did not have space for a new child and passed node 4 to the child with the smallest number of subtree nodes, node 2. Similarly, node 5 and node 6 joined. When node 6 crashed, all of its children detected the crash independently and contacted other ancestors to rejoin the tree. Every child of node 6 acts as a delegate of its subtree to save individual rejoining of the subtree nodes. Section 2.3 explains how to contact an ancestor for rejoining. The tree construction algorithm is given in Algorithm 1. We use $Sub_{no.}(x)$ to count the number of subtree nodes of node x, including itself.

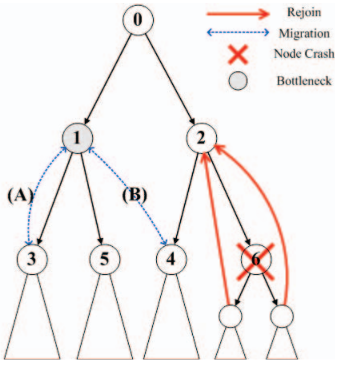


Fig. 1. Dissemination Tree Example

The dDT construction algorithm uses the number of subtree nodes as the metric for insertions, instead of the tree depth used in traditional balanced tree algorithms. This is because a rejoining node with a subtree may increase the tree depth by more than one, which is beyond the one by one tree height increase handled by traditional balanced tree algorithms. In addition, maintaining the total number of nodes in each subtree is simpler and more time efficient than maintaining the

Algorithm 1 dDT Construction (p, q)

```
Input: node $p$ receives node $q$'s join request
Output: parent of node $q$ in $dDT$
**if** $p$ does not have $d$ children **then**
    $Sub_{no.}(p) += Sub_{no.}(q)$
    **return** $p$
**else**
    find a child $f$ of $p$ s.t. $f$ has the smallest $Sub_{no.}$
    $Sub_{no.}(f) += Sub_{no.}(q)$
    **return** $dDT$ Construction $(f, q)$
```

depth of each subtree. Internal nodes need to wait until an insertion completes, then the updated tree depth can be collected layer by layer from leaf nodes back to the root. This makes the real time maintenance of the tree depth difficult and unnecessary when tree nodes are frequently joining and leaving. However, internal nodes can immediately update the total number of subtree nodes after forwarding a new node to a child. In BCoM, the tree depth is periodically collected to help set the sliding window size, where its result does not need to be updated in real time as discussed in Section 2.2.2. But using an outdated tree depth for insertions to dDT will lead to an unbalanced tree and degrade the performance.

2.2 Sliding Window Update Protocol

2.2.1 Basic Operations in Sliding Window Update

The sliding window update protocol regulates the consistency strictness in a dDT. "Sliding" refers to the incremental adjustment of the window size based on dynamic system conditions. If dDT_i of object i is assigned a sliding window size k_i, any replica node in dDT_i can buffer up to k_i unacknowledged updates before getting blocked from receiving new updates. In other words, each node in dDT_i is given a buffer of size k_i. At the beginning, the root receives the first update, sends it to all children and waits for their ACKs. There are two types of ACKs, R_ACK and NR_ACK. Both indicate that the update has been received. The difference is that R_ACK means the sender is ready to receive the next update; NR_ACK means the sender is not ready. While waiting, the root accepts and buffers the incoming updates as long as its buffer is not overflowed. When receiving an R_ACK from a child, the root sends the next update to this child if there is a buffered update that has not been sent to this child. When receiving an NR_ACK from a child, it marks the update to be received by this child and stops sending update to this child. After receiving ACKs from all children, the update is removed from the root's buffer.

There are two cases of buffer overflow: 1) when the root's buffer is full, the new updates are discarded until there is a space; 2) when an internal node's buffer is full, the node sends NR_ACK to its parent for the last received update. An R_ACK is sent to its parent when the internal node has a space in its buffer to resume receiving updates. A leaf node does not have any buffer. After receiving an update, it immediately sends an R_ACK to its parent.

Figure 2 shows an example of the sliding window update protocol with the window size set to 8. V stands for the version number of an update, as $V10-V13$ means that the node keeps the updates from the 10th version to the 13th version. Each internal node keeps the next version for its slowest child up to the latest version it received. Each leaf node only keeps the latest version it received.

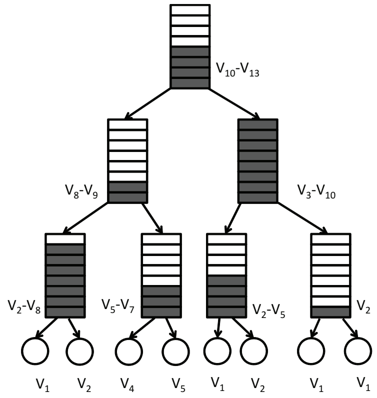


Fig. 2. An example of sliding window update protocol.

2.2.2 Window Size Setting

The sliding window size k_i is critical for balancing the consistency strictness, object availability for updating, and update dissemination latency. A large k_i masks the long network latency and the temporary unavailability of replica nodes, thus lowers the update discard rate. But a large k_i enlarges the discrepancy between the local version of a replica node with the latest version at the root. Thus, a large window size k_i weakens the consistency and increases the queueing delay of update propagation in dDT_i. *On the extremes, infinite buffer size provides eventual consistency without discarding updates, and buffer size zero provides sequential consistency with the worst update discard rate.*

We present an analytical model in Section 3 to set the sliding window size k_i so that the discard rate is minimized under a delay constraint and a consistency constraint. The detail formula is given in Section 3. Here, we explain the procedure for setting the window size. The root sets the window size for all tree nodes and adjusts it periodically when needed. The root measures the input metrics for computing the window size every T seconds and adjusts the value of k_i only after the metrics stabilize and the old k_i violates certain constraints. In this way, unnecessary changes due to temporary disturbances are eliminated to keep dDT_i stable. If k_i needs to be adjusted, it is incrementally increased or decreased until the constraints are satisfied.

The input metrics of computing the window size k_i include the update arrival rate λ, the tree height L, and the bottleneck service time μ_L. The arrival rate is directly measured by the root. The tree height and bottleneck service time are collected periodically from leaf nodes to the root in a bottom-up approach. The values of the two metrics are aggregated at every internal node so that the maintenance message keeps the same size. The aggregation is performed as follows: each leaf node initializes the tree height to zero ($L = 0$) and the bottleneck service time μ_L to the update propagation delay between itself and its parent. Each node sends the maintenance message to its parent. Once an internal node receives the maintenance messages from all children, it updates L as the maximum value of its children's tree height plus 1 and μ_L as the maximum value among its and every child's service time. If its service time is longer than a child's, a non-blocking migration is executed to swap the parent with the child. The aggregation continues until the root is reached.

2.3 Ancestor Cache Maintenance

Each replica node maintains a cache of m_i ancestors starting from its parent leading to the root in dDT_i. The value of m_i is set based on the node churn rate (i.e., the number of nodes joining and leaving the system during a given period) so that the probability that all m_i nodes simultaneously fail is negligibly small. If a node does not have m_i ancestors, it caches all the ancestors from its parent to the root.

A node contacts its cached ancestors sequentially layer by layer upwards when its parent becomes unreachable. This can be detected by ACKs and maintenance messages. Sequentially contacting enables a node to find the closest ancestor. The root is finally contacted if all the other ancestors are unavailable. The root failure is handled by the overlay routing, as a node with the nearest ID will replace the crashed node to be the new root. Different replication schemes may be used to reduce the cost of root failure, which is specific to a structured P2P overlay and beyond the scope of this paper.

The contacted ancestor runs the tree construction Algorithm 1 to find a new position for a rejoining node with its subtree. BCoM does not replace a crashed node with a leaf node to maintain the original tree structure because migration brings down a bottleneck node to the leaf layer for performance improvement. The new parent node transfers the latest version of the object to the new child if necessary. Since each node only keeps k_i previous updates, content transmission is used to avoid the communication overhead for getting the missing updates from other nodes. The sliding window update protocol resumes for incoming updates.

The ancestor cache provides fast recovery from node failures with a small overhead. Assuming the probability of a replica node failure is p, an ancestor cache of size m_i has a successful recovery probability as $1 - p^{m_i}$. An ancestor cache is easily maintained by piggybacking an ancestor list on each update. Whenever a node receives an update it adds itself to the ancestor list before propagating the update to its children. Each node uses the newly received ancestor list to refresh its cache. There is no extra communication, and the storage overhead is also negligible for keeping the information of m_i ancestors.

2.4 Tree Node Migration

Any non-leaf node will be blocked from receiving new updates if one of its descendants has a buffer overflow in the sliding window update protocol. It is quite possible that a lower layer node performs faster than a bottleneck node. This motivates us to promote the faster node to a higher level and degrade the bottleneck node to a lower level. For example in Figure 1, assume node 1 is the bottleneck node causing the root 0 to be blocked. The faster node may be a descendant of the bottleneck node as shown in (A) or a descendant of a sibling of the bottleneck node as shown in (B). When blocking occurs, node 0 can swap the bottleneck node 1 with a faster descendant who has more recent updates, like node 4, to remove blocking. Before blocking occurs, node 1 can be swapped with its fastest child who has the same update version to prevent blocking. The performance improvement through node migration is confirmed by our analytical model in Section 3.

There are two forms of node migration, as described below.

- Blocking triggered migration: the blocked node searches for a faster descendant who has a more recent update than the bottleneck node, and swaps them to remove blocking.
- Non-blocking migration: when a node observes a child performing faster than itself, it swaps with this child. Such migration prevents blocking and speeds up the update propagation for the subtree rooted at the parent node.

The swapping of (B) in Figure 1 is an example of blocking triggered migration and (A) is an example of non-blocking migration. Both forms of migration swap one layer at a time and, hence, multiple migrations are needed for multi-layer swapping. The non-blocking migration helps promote faster nodes to upper layers, which makes the searching in blocking-triggered migration easier. Since the overlay DHT routing in structured P2P networks relies on cooperative nodes, we assume BCoM is run by these cooperative P2P nodes transparent to end users. Tree node migration uses only the local information and improves the overall system performance.

2.5 Basic Operations in BCoM

BCoM provides three basic operations:

- Subscribe: if a node p wants to read the object i and keep it updated, p sends a subscription request to the root of dDT_i through the overlay routing. After receiving the request, the root runs Algorithm 1 to locate a parent for p in dDT_i, who will transfer its most updated version of object i to p. p receives the subsequent updates by following the sliding window update protocol. The message overhead of a subscription is at most the tree height as inserting a new node searches along a path from the root to a leaf in dDT_i.

- Unsubscribe: if a node p is not interested in object i anymore, it promotes its fastest child as the new parent and transfers its parent and other children's information to the newly promoted node. p also notifies them of the newly promoted node to update their related maintenance information. The message overhead of an unsubscription is constant, since the number of involved nodes is no more than the tree node degree, and each node has a constant overhead to update its local maintenance information.
- Update: after subscribing, if a node p wants to update the object, it sends an update request directly to the root using the IP routing. The root's IP address is obtained through the subscription or the ancestor cache. If the root crashed, p submits the update to the new root through the overlay routing. Updates are serialized at the root by their arrival times. The specific policy for resolving conflicts is application dependent. The message overhead of an update is constant for the direct submission to the root.

3 Analytical Model for Sliding Window Setting

The frequent node churn in P2P systems forbids us to use any complicated optimization techniques that require several hours of computation at workstations (e.g., [25]) or every node information in the system (e.g., [21]). BCoM adjusts the sliding window size to the dynamic P2P systems relying on limited information.

This section analyzes the setting of the sliding window size k_i for object i, where the update propagation to all replica nodes is modeled by a queuing system. We first analyze the queueing behavior of the dissemination tree dDT_i when it begins to discard an update. We then calculate the update discard probability and the expected latency for a replica node to receive an update. Finally, we set k_i to minimize the update discard rate given a consistency bound while ensuring the expected delay is no worse than the baseline by a small given threshold.

3.1 Queueing Model

Assuming the total number of replica nodes is N, the node degree is d, and there are L ($L = O(\log_d N)$) layers of internal nodes with an update buffer of size k_i (i.e., each node in layer $0 \dots L-1$ has a sliding window k_i). The leaf nodes are in layer L and do not have any buffer. The update arrivals are modeled by a Poisson process with an average arrival rate λ_i (simply as λ), since each update is issued by a replica node independently and identically at random. The latency of receiving an update from the parent and an acknowledgment from the child is denoted as the service time for an update propagation. The service time for one layer to its adjacent layer below is the longest parent-child service time in these two layers. μ_l denotes the service time for update propagation from layer l to layer $(l+1)$. For example, μ_0 is the longest service time from the root to its child, μ_{L-1} is the longest service time from a layer $(L-1)$ node to its child (i.e., a leaf node). The update propagation delay is assumed to be exponentially distributed. The update propagation in dDT_i is modeled as a queuing process as shown in Figure 3 (a): updates arrive at the root with an average rate λ, then go to the layer 0 node's buffer of size k_i. The service time for propagating from layer 0 to layer 1 is μ_0. After that, the updates go to a layer 1 node's buffer of size k_i with service time μ_1 for propagating to a layer 2 node. The propagations end when updates are received by a leaf node in layer L.

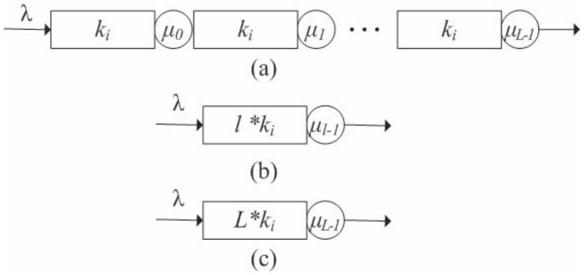


Fig. 3. Queuing Model of Update Propagation.

An update may only be discarded by the root when its buffer is overflowed. This happens when the root is waiting for an R_ACK from its slowest child in layer 1, who is waiting for an R_ACK from its slowest child in layer 2. The waiting cascades until a bottleneck node of dDT_i is reached, say in the layer l, $0 \le l \le L$. The nodes in layers $l+1 \dots L$ (if $l < L$) do not receive any update even when their buffers are not full. All the nodes in the path from the root to the bottleneck node have buffer overflow. The nodes along the path are denoted as $p_0, p_1 \dots p_l$, where p_0 is the root and p_l is the bottleneck node. After the bottleneck node p_l has a space in its buffer and sends an R_ACK to its parent, the R_ACK is then propagated to the root p_0 such that the root can purge the update from its buffer and accept a new one. The update propagation from $p_0 \rightarrow p_1, p_1 \rightarrow p_2, \dots p_{l-1} \rightarrow p_l$ is in parallel and the service time μ_{l-1} between p_l and p_{l-1} should be the longest along this path (i.e., $\mu_{l-1} > \mu_j, 0 \le j < l-1$). Therefore, the queuing model of the update discard is transformed to a queue with an effective buffer of size $l * k_i$ for dDT_i, and the service time is μ_{l-1}, as shown in Figure 3 (b).

This queuing model explains that given a k_i, the effective buffer size $l * k_i$ is determined by l, which is the layer of the bottleneck node. The larger the effective buffer size, the lower the discard probability. When the bottleneck node is a leaf node ($l = L$), buffer resources of dDT_i are fully used with an effective buffer size $L * k_i$. This inspires the Tree Node Migration techniques presented in Section 2.4, which moves down bottleneck nodes to the leaf layer. The discard probability of an update is computed based on the queuing model of dDT_i after being optimized by tree node migrations as shown in Figure 3 (c). The queue becomes an $M/M/1/$ queue with a buffer size $L * k_i$, an arrival rate λ and a service time μ_{L-1}.

3.2 Availability and Latency Computation

Define the update request intensity as ρ.

$$\rho = \frac{\lambda}{\mu_{L-1}} \tag{1}$$

Define the probability of n updates in the queue as π_n. Based on the queueing theory for $M/M/1$ finite queue [26], π_n is represented as Eq.2.

$$\pi_n = \rho^n \pi_0 \tag{2}$$

The discard probability is π_{L*k_i}, which indicates the buffer overflow. From $\Sigma_{n=0}^{L*k_i} \pi_n = 1$, we get $\pi_0 = \frac{1-\rho}{1-\rho^{L*k_i}}$. And the discard probability is computed in Eq.3.

$$\pi_{L*k_i} = \frac{1-\rho}{1-\rho^{L*k_i}} \rho^{L*k_i} \tag{3}$$

The expected number of packets in the queue $E[N_{L*k_i}]$ is calculated in Eq.4.

$$E[N_{L*k_i}] = \sum_{0 \le n \le L*k_i} n * \pi_n \tag{4}$$

Plug in the Eq.2 for π_n, the final form of $E[N_{L*k_i}]$ is given in Eq.5.

$$E[N_{L*k_i}] = \frac{(L*k_i+1)\rho^{L*k_i+1}}{(\rho^{L*k_i+1}-1)} + \frac{\rho}{(1-\rho)} \tag{5}$$

The expected delay $E[T_{L*k_i}]$ is calculated by Little's law in Eq.6, where $E[N_{L*k_i}]$ is the expected number of packets in the queue and $\lambda(1-\pi_{L*k_i})$ is the arrival rate of the accepted updates.

$$E[T_{L*k_i}] = \frac{E[N_{L*k_i}]}{\lambda(1-\pi_{L*k_i})} \tag{6}$$

3.3 Window Size Setting

The effectiveness of a consistency protocol is measured by three attributes: consistency strictness, object availability, and latency for receiving an update. The three are in subtle tension towards each other. Given the update arrival rate and the service time, increasing the window size k_i lowers the discard probability, while prolongs the expected latency and weakens the consistency strictness. π_{L*k_i} is the discard probability. The expected latency $E[T_{L*k_i}]$ indicates the average delay for an update to be received by a replica node. The consistency strictness is measured by the number of updates a replica node has not yet received, which is at most $L * k_i$ in dDT_i.

BCoM sets the window size to minimize the update discard rate under the constraints that the number of not-yet-received updates is bounded to K_m and the latency for receiving an update is no worse than the sequential consistency for a small bound T_s as calculated in Eq.7. $E[T_{L*k}]$ is the expected latency with a window size k and $E[T_L]$ is the expected latency when applying sequential consistency to dDT_i, which serves as the baseline for bounding the latency performance. The latency threshold T_s and the consistency strictness threshold K_m are set according to application requirements. In our simulation, empirically setting T_s to 1.3 achieves good results as shown in Figure 9 and Figure 11, the discard probability is improved from almost 100% to 5% at the cost of latency increases less than one third most of the time. K_m is set to 60 for a network of 1000 nodes.

$$k_i = arg\min \ \pi_{L*k} \ s.t. \ \frac{E[T_{L*k}]}{E[T_L]} \le T_s, L*k \le K_m \tag{7}$$

4 Performance Evaluation

In this section, we extend the P2PSim tool [27] to simulate BCoM with heterogeneous node capacities and transmission latencies. While BCoM can be applied to any type of structured P2P systems, we choose Tapestry[28] as a representative network for simulations. We evaluate the efficiency of BCoM with comparison to SCOPE[1], which is the most relevant work and a widely studied consistency technique in structured P2P systems.

Simulation Setting: We simulate a network of 1000 nodes because anything larger cannot be executed stably in P2PSim. The number of objects ranges from 10^2 to 10^4. The object popularity follows a Zipf's distribution, and the update arrivals are generated by a Poisson process with different average arrival rates. By default, each node issues 200 updates during a simulation cycle, which is $7.2 * 10^6$ time slots. We simulate the situation where frequent updates may overload the servers, which motivates the use of P2P systems. Given that transmitting one update uses only 10 to 100 time slots, the number of time slots covered in a simulation cycle (i.e., $7.2*10^6$) is large enough to generate sustainable results. The data points in our figures are the average values of 20 trials.

The heterogeneity of node capacities follows a Pareto distribution [22]. We set the shape parameter $a = 1$ and scale parameter $b = 900$ to get 900 different node capacities. Network topology is simulated by two transit-stub topologies generated by GT_ITM [29] to model dense and sparse networks: (1) ts1k-small (dense) - 2 transit domains each with 4 transit nodes, 4 stub domains attached to each transit node, and 31 nodes in each stub domain. (2) ts1k-large (sparse) - 30 transit domains each with 4 transit nodes, 4 stub domains attached to each transit node, and 2 nodes in each stub domain.

The node degree is set to 5 based on the average Gnutella node degree, which is 3 to 5. To have a fair comparison, we also set the vector degree of each SCOPE node to 5. The *update discard rate* (the ratio of the number of discarded updates to the total number of updates), and the *update dissemination latency* (the average delay for a replica node to receive an update) are used to measure the performance.

Efficiency of the Window Size: This simulation examines the efficiency of applying sliding window update protocol. The curves in Figure 4 and Figure 5 show that by increasing the window size from 1 to 20, the discard rate is dropped from 80% to around 5%, and the latency is increased only by 20%. The results confirm that BCoM significantly improves the object availability with slightly increased latency compared to applying the sequential consistency.

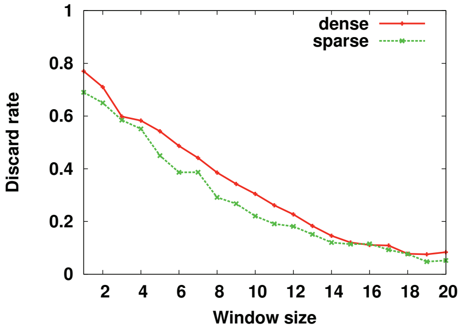


Fig. 4. The impact of window size on discard rate.

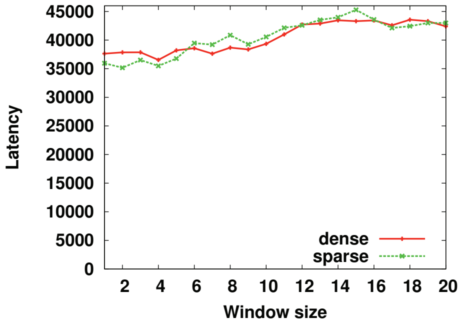


Fig. 5. The impact of window size on latency.

Impacts of the Window Size on the Extent of Inconsistency: This simulation examines the extent of inconsistency among all replicas of an object by varying the window size. Figure 6 shows the results, where the inconsistency extent is defined as the average number of updates a replica falls behind the latest version at the root. Since the upper bound of inconsistency in BCoM is the window size multiplied by the tree height of a dDT, we use the average to show the real situation instead of the theoretical upper bound. As expected, the inconsistency extent grows larger as the window size increases, however, it grows much slower than the upper bound. When the window size is 20 and the tree height is around 4 to 6, the upper bound is around 80 to 120. But the average inconsistency extent is only 11, much smaller than the upper bound. Given the total number of replica nodes is 1000, such inconsistency is quite acceptable when the update discard rate is dropped to only 5% as shown in Figure 4.

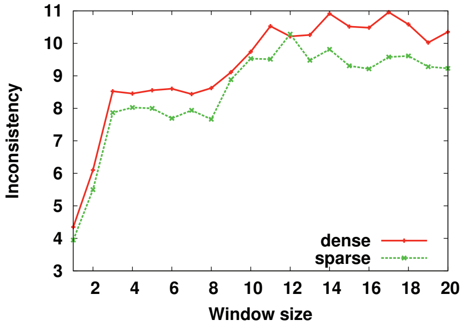


Fig. 6. The impact of window size on inconsistency degree.

Accuracy of the Analytical Model: This simulation examines the accuracy of the analytical model for window size setting presented in Section 3. We compare the latency estimated by Eq.6 with the simulation latency results as shown in Figure 5. We also show the error rate of latency estimation in Figure 7, which is the ratio of the difference between the estimation and the simulation results over the simulation results. The discrepancy is mainly caused by the node churn because node leavings and rejoinings introduce extra delay and change parts of the tree structure. However, the error rates are less than 25% with different window sizes. Such small error rates indicate that our analytical model has captured the queueing behavior of the update dissemination in BCoM, which is the dominant factor to the system performance.

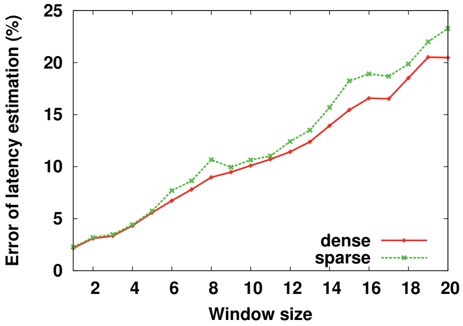


Fig. 7. The impact of window size on latency estimation.

Storage Overhead: This simulation shows the storage overhead for buffering updates. The upper bound of storage overhead at a replica is the update packet size multiplied by the window size (i.e., the maximum buffer size). Figure 8 presents the average buffer occupancy of all replicas to show the storage overhead in the real situation instead of the theoretical upper bound. The average buffer occupancy decreases as the window size increases, when the window size is 20 the average buffer occupancy is only 10%. Thus, most replicas have small storage overhead. This is because the window size is determined by the bottleneck service time (i.e., the slowest replica's service time) to reduce the update discard rate, other replicas only have a small number of udpates buffered even when the window size is large.

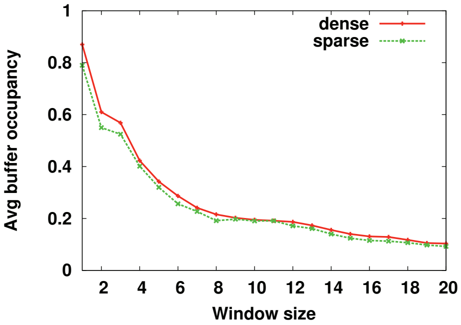


Fig. 8. The impact of window size on storage overhead.

Scalability of BCoM: This simulation verifies the scalability of BCoM with comparison to SCOPE[1] by varying the number of replica nodes and the update rate of each object. The results in Figure 9 and Figure 10 show that the discard rate of BCoM is maintained to less than 10% as the number of replicas per object increases from 10 to 1000 and the number of updates issued per node increases from 1 to 200. On the other hand, applying the sequential consistency makes the discard rate of SCOPE almost 100%, except for a very small number of replica nodes (i.e., 10 replicas per object) or an extremely low update rate (i.e., 1 update per node). An update cannot be accepted by SCOPE until the previous update is received by every replica node. The prohibitively long synchronization for the sequential consistency makes SCOPE discard most updates.

We intentionally relax the consistency requirement for SCOPE when calculating their discard rate, latency and overhead results by not requiring their new joining/rejoining nodes to be synchronized. This relaxation gives better results to SCOPE for all three metrics. Without this relaxation, SCOPE's discard rate is even worse, which is not useful for comparison. Thus, the results of BCoM include content transfer delay for all joining/rejoining nodes while the results of SCOPE exclude the synchronization delay for all joining/rejoining nodes. This is an important reason why BCoM has slightly longer dissemination latency than that of SCOPE when the number of replica nodes is large as shown in Figure 11 or the updates are frequent as shown in Figure 12.

Another critical reason why SCOPE has slightly better latency and overhead is that it discards a large portion of updates. The measurements of latency and overhead only count accepted updates. With such high discard rate, SCOPE takes advantage of accepting much fewer updates than BCoM when measuring latency and overhead.

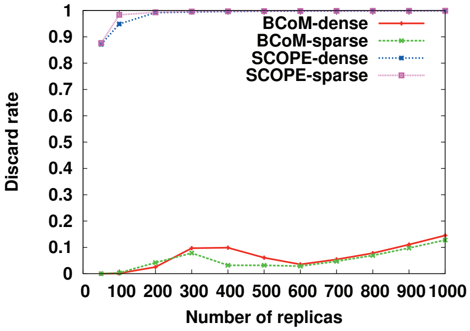


Fig. 9. The impact of replica number on discard rate.

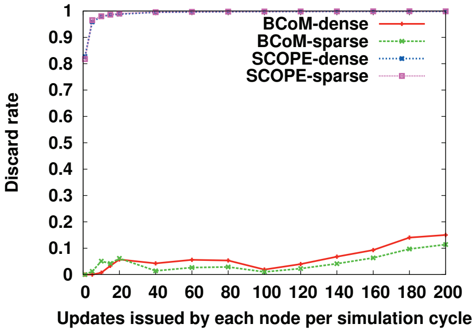
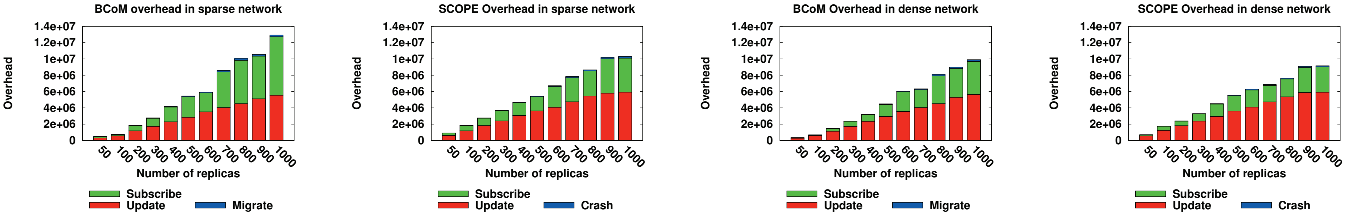


Fig. 10. The impact of update pattern on discard rate.

Overhead of BCoM: This simulation compares the overhead of BCoM with that of SCOPE as shown in Figure 13. The consistency maintenance overhead of each object consists of three parts: subscription overhead, update overhead, and crash/migration overhead. We use the label ”migrate” to indicate the migration and crash recovery overhead in BCoM. BCoM keeps the overhead at the same level as that in SCOPE because the ancestor cache maintenance and the node migration mostly piggyback on update dissemination.

Fault Tolerance of BCoM: This simulation examines BCoM's robustness against node failures by varying the node mean life time. The node life time is the ratio of the average number of slots a node stays online at one time to the total number of slots in a simulation cycle. The smaller the life time is, the more frequently the nodes join and leave. The results of SCOPE are not presented because their discard rate is nearly 100% in the presence of nodes joining and leaving. Figure 14, Figure 15, and Figure 16 show the impacts of various churn rates on the dDT tree height, the update discard rate, and the update dissemination latency. The results demonstrate that BCoM is robust against the node churn. Figure 14 shows that the tree height is ranging from 4 to 6, which means our dDT is nearly complete as a 5-ary tree of 1000 nodes has tree height more than 4. The discrepancy caused by node churn on the tree height is less than 2, which helps BCoM maintain consistent discard rate and latency. Buffering some earlier updates in each intermediate node masks the delay for nodes to rejoin/join the tree and keeps the discard rate low under



(a) (b) (c) (d)

Fig. 13. Overhead comparison between BCoM and SCOPE

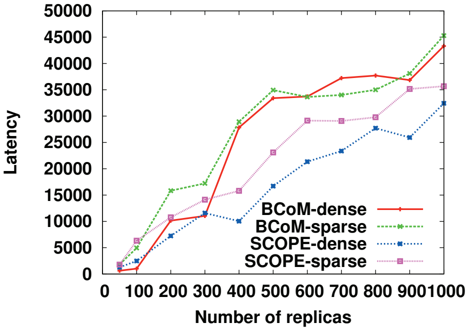


Fig. 11. The impact of replica number on latency.

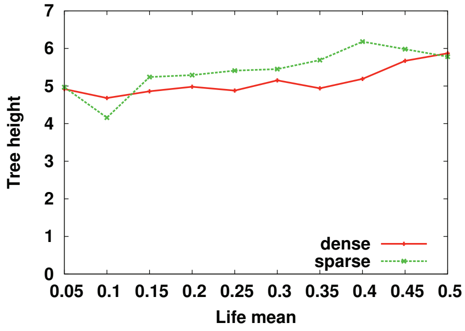


Fig. 14. The impact of churn rate on tree height

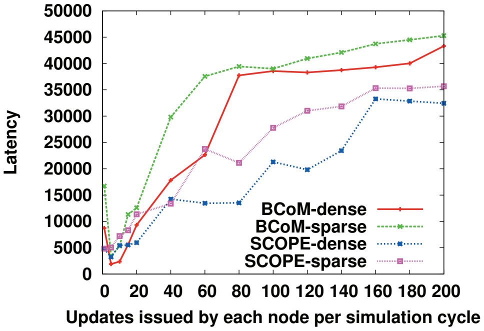


Fig. 12. The impact of update pattern on latency.

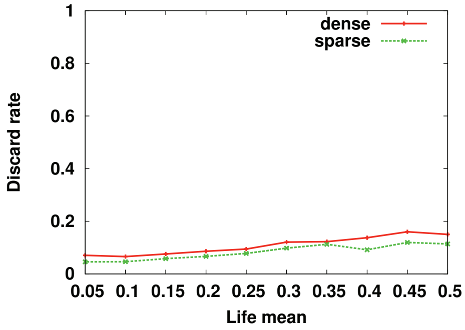


Fig. 15. The impact of churn rate on discard rate

the node churn as shown in Figure 15. The efficient dDT construction and the use of ancestor cache reduce the delay for node joining/rejoining and prevent the degradation of update dissemination as shown in Figure 16. Note that the latency goes down sharply when node life time is small. The reason is that a node clears its buffer when it goes offline. Therefore, with an extremely short life time, a node's buffer is always near empty so that the queueing delay in this case is also extremely short.

5 Case Study

In this section, we evaluate the performance of BCoM using a large-scale social networking application FriendFeed [30]. FriendFeed is a real-time feed aggregator that consolidates the updates from social media and social networking websites. It is created in 2007 and acquired by Facebook in 2009. Existing solutions to implement social networking is using dedicated servers, which are notoriously difficult to scale. Social network applications are incessantly evolving as more new users join with more frequent social interactions. Servers' capacities should continuously keep up for the growing demand. Twitter engineers have famously described re-architecting Twitters servers multiple times to keep up with rapid increases in throughput as the system became more popular [31]. We believe that the P2P approach is the direction for the future social network applications with better scalability. Moreover, the P2P implementation is economic-friendly, which is particularly appealing to start-ups as we envision new social network applications are emerging.

5.1 Trace Data and Experimental Setup

Friendfeed users share posts on his/her blog with a list of subscribers, who can comment directly under the original blog post. BCoM is applied to FriendFeed for maintaining consistency between a FriendFeed user and all his/her subscribers. Each user's blog is modeled as an

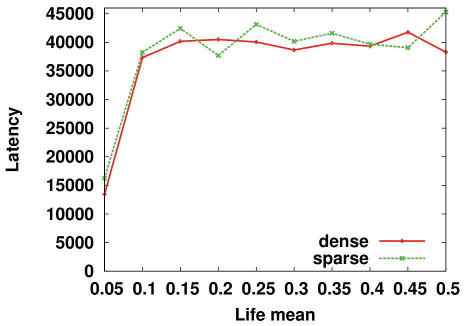


Fig. 16. The impact of churn rate on latency

object in BCoM. Either a post or a comment is an update about the object, and a subscriber to a FriendFeed user is a replica node of the object (i.e., the user's blog).

Workload Model. We use the real trace data in [30] to generate the workload of subscriptions and updates. The trace includes 671840 FriendFeed users, and is collected from Aug 1, 2010 to Sep 30, 2010. Table 1 gives the summary of the trace data.

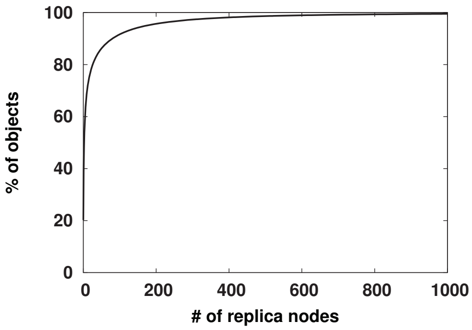


Fig. 17. CDF of the number of replica nodes per object

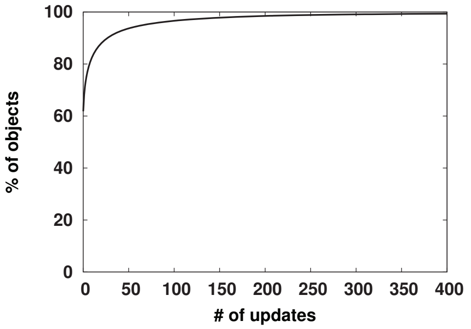


Fig. 18. CDF of the number of updates per object

We examine the main features of the trace data in Figures 17, 18, 19. Figure 17 shows the cumulative distribution function (CDF) of the number of replica nodes per object. The object popularity is highly skewed. More than 90% of objects have replica nodes less than 100, yet around 1% of objects have replica nodes more than 800. The maximum number of replica nodes per object reaches 113923. Figure 18 shows the CDF of the number of updates per object. The update distribution is also highly skewed. More than 90% of the objects have updates less than 50, while, around 2% of objects have updates more than 200. Figure 19 shows the total number of updates generated in the system, where we can see the udpate generation is unpredictable with frequent bursts. These trace analysis results show that the workload of update delivery and consistency maintenance is highly skewed among objects and constantly changing over time. One of the goals of BCoM is to provide balanced consistency maintenance for such workload patterns through our sliding window update protocol.

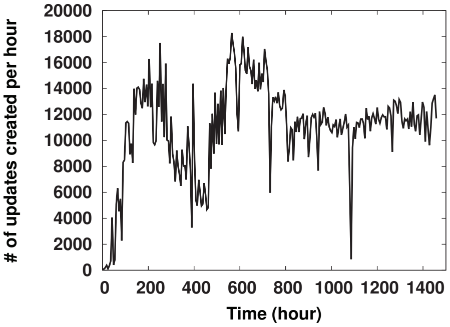


Fig. 19. The total number of updates in the system per hour

Network Model. In our experiments, each peer is an individual machine in a simulated network, representing a general Internet user experience. To measure the update dissemination latency, we adopt the widely used statistics of the user bandwidth capacity collected at U.S. Broadband report [32]. The upload capacity of peers is shown in Figure 20, which is in a range from 256 Kb/s to 10 Mb/s. To simulate wide geographic areas where peers come from, the inter-peer round-trip time (RTT) is simulated by drawing n ($n = 671840$ in our experiments) nodes from the real data set [33] which represents a wide area interactive application. The mean, median, and standard deviation of inter-player RTT of this data set are 81 ms, 64 ms, and 63 ms. Vivaldi 3D coordination system [34] is used to extrapolate the RTT values among pairs of nodes who did not probe each other in the data set. We use a two-state Gilbert model [35], which models the packet loss property of Internet paths, setting loss rate to 1% and mean loss burst time to 100 ms. The churn probability is set to 20% according to churn studies in [36]. The update dissemination tree structure in BCoM is configured the same as in our simulation in Section 4.

Performance Metrics. We use the *update discard rate*, *update dissemination latency*, and *buffer occupancy* to measure the performance of BCoM. These metrics are defined in our simulation Section 4.

Performance Results. Figure 21 shows the update discard rate in BCoM. Most of the time BCoM has zero discard rate except two short periods around 200^{th} hour and 600^{th} hour, where BCoM has non-zero discard rates due to update bursts as shown in Figure 19. Even

TABLE 1
Summary of FriendFeed Traces

Total nodes	671,840
Time duration	2 months
Total updates	16,200,549 (12,450,658 posts + 3,749,891 comments)
Average number of replica nodes per object	41.40
Average number of updates created each second in the whole system	3.07
Average update packet size	354.91 bytes

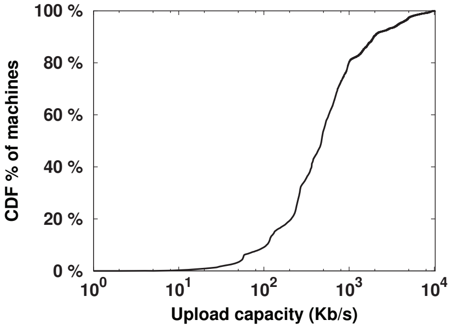


Fig. 20. CDF of peers upload capacity

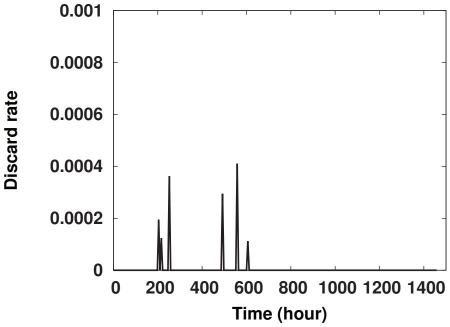


Fig. 21. The update discard rate

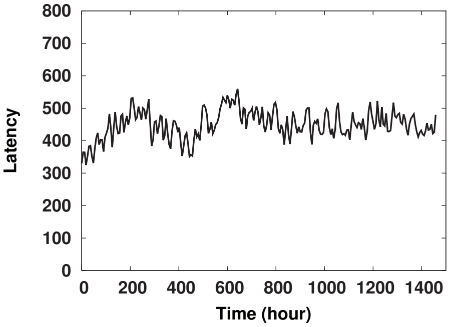


Fig. 22. The average update dissemination latency (in millisecond)

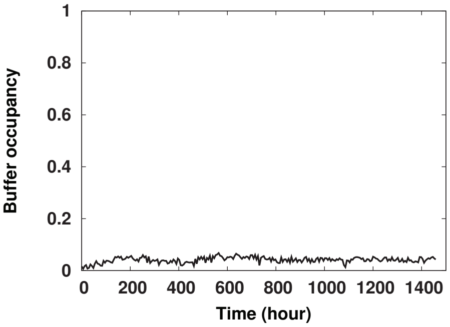


Fig. 23. The average buffer occupancy

though the maximum update discard rate is less than 0.04%, which is negligible in such a large network of more than $6.7 * 10^5$ nodes. Figure 22 shows the average update dissemination latency in BCoM, which is kept between 400 and 500 milliseconds. Such fast and stable update delivery confirms the efficiency of the sliding window update protocol with the two enhancements in BCoM. Figure 23 shows the average buffer occupancy in BCoM. The buffer size (i.e., window size) is adjusted around 1 to 20 for different objects and dynamic update workloads as shown in figures 17, 18, 19. The buffer size is set for accomodating the slowest node in an update dissemination tree, while most of the nodes have buffer occupancy less than 10% so that the buffer overhead introduced is quite small. All these results demonstrate the applicability and efficiency of BCoM in large-scale social network applications.

6 Related Work

Consistency Maintenance in P2P systems: In structured P2P systems, strong consistency is provided by organizing replica nodes to an auxiliary structure on top of the overlay for update propagation. Examples include the tree structure in SCOPE [1], the two-tiered structure in OceanStore [4], and a hybrid of tree and two-tiered structure in [22]. The tree construction algorithms in SCOPE [1] and in [22] build a tree by recursively partitioning the identifier space and selecting a representative node as a tree node for each partition. Only leaf nodes store object copies, all the intermediate nodes only store information of the tree structure in their sub-space. Nodes who may not be interested in the object are in the object's update dissemination tree, which adds unnecessary overhead of maintaining the tree from node failures. To the contrary, BCoM constructs the dissemination tree dDT by only involving replica nodes who are interested in the object, which greatly reduce the overhead of maintenance and update propagation. BCoM also efficiently builds the dissemination tree dDT to make it balanced and robust under the node churn.

In unstructured P2P systems, mainly two types of bounded consistency are provided: (1) probabilistic bounded consistency: rumor spreading [17] and replica chain [18] are used to ensure a certain probability of an update being received. The probability is tuned by adjusting the redundancy degree in propagating an update to balance the communication overhead with the consistency strictness. (2) time-bounded consistency: TTL guided push and/or pull methods are used (e.g., [19] [20]) to indicate a valid period for a replica copy. When the period expires, the replica node checks the validity of the replica copy with the source to serve the following read requests. The problems of these techniques are (1) node-level consistency is not ensured by probabilistic bounded consistency, and (2) time-bounded consistency sets a uniform TTL timer for all nodes. In the situation where nodes have various update frequencies, it is impossible to set a TTL timer that works for all nodes. BCoM avoids both drawbacks by using a sliding window update protocol, which directly limits the inconsistency by the number of buffered updates and ensures the bounded consistency for each replica node.

Overlay Content Distribution: Update dissemination in P2P systems has four requirements: (1) a bounded delay for update delivery, (2) robustness to the frequent node churn and different workload patterns, (3) awareness of heterogeneous peer capacities, and (4) scalability with a large number of peers. The LagOver [37] constructs an update dissemination tree by considering each user's capacity and latency requirements to address (1) and (3), both of which are also handled by the tree node migration in BCoM. The major difference is that LagOver improves the performance to meet the individual replica node's requirement, while the tree node migration in BCoM improves the overall system performance. Moreover, LagOver requires information on each user's latency requirement and capacity, which are infeasible to be implemented in P2P systems. To the contrary, BCoM's node migration only involves local information, which is also performed on demand to support (2) without asking a replica node to specify requirements in advance.

The "side link" is used in content dissemination tree in [23] to address (2), where each node keeps multiple side links from other subtrees to minimize the impact of loss multiplication in a tree structure. The two end nodes of a side link do not share any ancestor except the root. The ancestor cache in BCoM achieves the same goal by caching only ancestors and contacting the ancestors sequentially one layer upwards from the failed nodes. Our ancestor cache has extra benefits by avoiding communication overhead to maintain end nodes on other subtrees. All the ancestors' information can be piggyback on the update propagation. In BCoM a node sequentially contacts the cached ancestors to avoid conflict relocation decisions while in [23] a node uses multiple side links in parallel to retrieve the lost packets.

Tunable Consistency Models: Previous works [38][39] have proposed continuous models for consistency maintenance, which have been extended by a composable consistency model in [13] for P2P applications. The core technique for maintaining consistency used in [13] is a hybrid of push and pull methods, which are also used to provide application tailored cache consistency in [40] [19]. Although each node can specify its consistency requirement, the model in [13] makes each node perform the strongest consistency maintenance from all its descendant nodes in the overlay replica hierarchy. Thus, the overhead of maintaining consistency at a node is not reduced even it only requires a weak consistency as long as one of its descendant nodes requires a strong consistency. An analytical model for adaptive update window protocol is presented in [21], where the window specifies the number of uncommitted updates in each replica node's buffer. The information of each node's update rate and propagation latency are required to optimize the window size in [21]. Such optimization is unrealistic for P2P systems due to their required global information. To the contrary, every BCoM node has a fair amount of consistency maintenance overhead because of the uniform buffer size and node degree in dDT. Moreover, BCoM provides incentives for nodes to contribute more bandwidth to update dissemination, as we promote faster nodes closer to the root and they will receive updates sooner. The window size optimization model in BCoM only requires limited information that can be obtained in a fully distributed way.

7 Conclusion

This paper presents a balanced consistency maintenance framework (BCoM) for balancing the object availability, update propagation latency and consistency strictness in structured P2P systems. A sliding window update protocol is applied with two enhancement schemes. The window size setting is analyzed through a queueing model, which works well for dynamic network conditions, different workload patterns and heterogeneous node capabilities. Various application consistency requirements are also smoothly served. The simulation results from P2PSim demonstrate that BCoM outperforms SCOPE [1] by greatly improving the update discard rate from almost 100% to 5%.

Acknowledgment

This research was partly supported by NSF grants CNS 0509440, CCF 0811834 and CNS 0832108.

References

[1] X. Chen, S. Ren, H. Wang, and X. Zhang, "Scope: scalable consistency maintenance in structured P2P systems," in *IEEE INFOCOM*, 2005.

[2] A. Rowstron and P. Druschel, "Storage management and caching in past a large-scale persistent peer-to-peer storage utility," in *ACM SOSP*, 2001.

[3] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide area cooperative storage with CFS," in *USENIX Security Symp.*, 2000.

[4] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, and D. Geels, "Oceanstore: an architecture for global-scale persistent storage," in *ACM ASPLOS-IX*, 2000.

[5] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "Scribe: A large scale and decentralized application level multicast infrastructure," *IEEE J-SAC*, vol. 20, no. 8, pp. 1489–1499, 2002.

[6] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: a decentralized peer-to-peer web cache," in *ACM PODC*, 2002.

[7] V. Ramasubramanian and E. G. Sirer, "Beehive: exploiting power law query distribution for o(1) lookup performance in peer-to-peer overlays," in *NSDI*, 2004.

[8] M. Waldman, A. D. Rubin, and L. F. Cranor, "Publius: A robust, tamper-evident, censorship-resistant web-publishing systems," in *USENIX Security Symp.*, 2000.

[9] G. Urdaneta, G. Pierre, and M. V. Steen, "A decentralized wiki engine for collaborative wikipedia hosting," in *WEBIST*, 2007.

[10] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: Enabling large-scale, high-speed, peer-to-peer games," in *ACM SIGCOMM*, 2008.

[11] S. Buchegger, D. Schioberg, L. H. Vu, and A. Datta, "Peerson: P2p social networking ł early experiences and insights," in *EuroSys*, 2009.

[12] G. Oster, P. Urso, P. Molli, and A. Imine, "Data consistency for P2P collaborative editing," in *CSCW*, 2006.

[13] S. Susarla and J. Carter, "Flexible consistency for wide area peer replication," in *IEEE ICDCS*, 2005.

[14] M. P. Herlihy and J. M. Wing, "Linearlizability: A correctness condition for concurrent objects," *ACM Trans Programm. Lang. Syst.*, vol. 12, no. 3, pp. 463–492, 1990.

[15] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, "Managing update conflicts in bayou, a weakly connected replicated storage system," in *ACM SOSP*, 1995.

[16] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," in *ACM PODC*, 2002.

[17] A. Datta, M. Hauswirth, and K. Aberer, "Updates in highly unreliable, replicated peer-to-peer systems," in *IEEE ICDCS*, 2003.

[18] Z. Wang, S. K. Das, M. Kumar, and H. Shen, "An efficient update propagation algorithm for P2P systems," *Computer Communications*, vol. 30, no. 5, pp. 1106–1115, 2007.

[19] X. Liu, J. Lan, P. Shenoy, and K. Ramaritham, "Consistency maintenance in dynamic peer-to-peer overlay networks," *Computer Networks*, vol. 50, no. 6, pp. 859–876, 2006.

[20] X. Tang, J. Xu, and W. C. Lee, "Analysis of TTL-based consistency in unstructured peer-to-peer networks," *IEEE TPDS*, vol. 19, no. 12, pp. 1683–1694, 2008.

[21] C. Zhang and Z. Zhang, "Trading replication consistency for performance and availability: an adaptive approach," in *IEEE ICDCS*, 2003.

[22] Z. Li, G. Xie, and Z. Li, "Efficient and scalable consistency maintenance for heterogeneous peer-to-peer systems," *IEEE TPDS*, vol. 19, no. 12, pp. 1695–1708, 2008.

[23] F. Wang, J. Liu, and Y. Xiong, "Stable peers: existence, importance, and application in peer-to-peer live video streaming," in *ACM MobiCom*, 2004.

[24] Y. Hu, M. Feng, and L. N. Bhuyan, "A balanced consistency maintenance protocol for structured P2P systems," in *IEEE INFOCOM mini conference*, 2010.

[25] H. Yu and A. Vahdat, "The costs and limits of availability for replicated services," *ACM TOCS*, vol. 24, no. 1, pp. 70–113, 2006.

[26] D. P. Bertsekas and R. G. Gallager, *Data Networks*. NJ: Prentice-Hall: Englewood Cliffs, 1986.

[27] "P2PSim," http://pdos.csail.mit.edu/p2psim/. [Online]. Available: http://pdos.csail.mit.edu/p2psim/

[28] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment," *IEEE J-SAC*, vol. 22, no. 1, pp. 41–53, 2004.

[29] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *IEEE INFOCOM*, 1996.

[30] F. Celli, F. M. L. D. Lascio, M. Magnani, B. Pacelli, and L. Rossi, "Social network data and practices: the case of friendfeed," in *International Conference on Social Computing, Behavioral Modeling and Prediction*, 2010.

[31] "E. weaver. improving running components at twitter." http://blog.evanweaver.com/2009/03/13/qcon-presentation/. [Online]. Available: http://blog.evanweaver.com/2009/03/13/qcon-presentation/

[32] "Broadband report," 2008, http://www.dslreports.com. [Online]. Available: http://www.dslreports.com

[33] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye, "Measurement and estimation of network qos among peer xbox 360 game players," in *PAM*, 2008.

[34] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network cooridante system," in *ACM SIGCOMM*, 2004.

[35] E. N. Gilbert, "Capacity of a burst-noise channel," *The Bell System Technical Journal*, vol. 39, 1960.

[36] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs," in *ACM SIGMETRICS*, 2000.

[37] A. Datta, I. Soica, and M. Franklin, "Lagover: latency gradated overlays," in *IEEE ICDCS*, 2007.

[38] N. Krishnakumar and A. Bernstein, "Bounded ignorance: A technique for increasing concurrency in a replicated system," *ACM TODC*, vol. 19, no. 4, 1994.

[39] H. Yu and A. Vahdat, "Design and evaluation of a continuous consistency model for replicated services," in *OSDI*, 2000.

[40] M. Zhao and R. J. Figueiredo, "Application-tailored cache consistency for wide-area file systems," in *IEEE ICDCS*, 2006.

PLACE PHOTO HERE

Yi Hu Yi Hu received her B.Eng. degree in Computer Science from City University of Hong Kong in 2006. Currently, she is a PhD student, advised by Dr. Laxmi N. Bhuyan, in Computer Science & Engineering Department at University of California, Riverside.

Min Feng Feng Min received his B.Eng. degree in Computer Science and Technology from Tongji University, Shanghai in 2005, and his M.Phil. degree in Computer Science from City University of Hong Kong in 2007. Currently, he is a PhD student, advised by Dr. Rajiv Gupta, in Computer Science & Engineering Department at University of California, Riverside.

Laxmi N. Bhuyan Laxmi Narayan Bhuyan is Distinguished Professor and Chairman of Computer Science and Engineering Department at the University of California, Riverside (UCR). Prior to joining UCR in January 2001, he was a professor of Computer Science at Texas A &M University (1989-2000) and Program Director of the Computer System Architecture Program at the National Science Foundation (1998-2000). He has also worked as a consultant to Intel and HP Labs.