Efficient and Scalable Consistency Maintenance for Heterogeneous Peer-to-Peer Systems

Zhenyu Li, *Student Member*, *IEEE*, Gaogang Xie, *Member*, *IEEE*, and Zhongcheng Li, *Member*, *IEEE*

Abstract—Consistency maintenance mechanism is necessary for the emerging peer-to-peer (P2P) applications due to their frequent data updates. Centralized approaches suffer from single point of failures, while previous decentralized approaches consume too many network resources because of locality-ignorant structures. To address this issue, we propose a scalable and efficient consistency maintenance scheme for heterogeneous P2P systems. Our scheme takes the heterogeneity nature into account and forms the replica nodes of a key into a locality-aware hierarchical structure, in which the upper layer is Distributed Hash Table (DHT)-based and consists of powerful and stable replica nodes, while a replica node at the lower layer attaches to a physically close upper layer node. A $d\text{-}ary$ update message propagation tree (UMPT) is dynamically built upon the upper layer for propagating the updated contents. As a result, the tree structure does not need to be maintained all the time, saving a lot of cost. Through theoretical analyses and comprehensive simulations, we examine the efficiency and scalability of this design. The results show that, compared with previous designs, especially locality-ignorant ones, our approach is able to reduce the cost by about 25-67 percent.

Index Terms—Consistency maintenance, P2P systems, locality aware, hierarchical structure.

1 Introduction

In peer-to-peer (P2P) systems, shared resources are normally replicated on multiple nodes to improve system reliability and availability. In addition, query results are cached along the query path to reduce response time of subsequent queries. Hence, replication and caching are widely adopted ways to improve system performance in current P2P systems. Existing works [9], [15], [17], however, mainly focus on replica creation and do not pay much attention on the consistency maintenance. Although data objects in some P2P file sharing systems, such as Gnutella [2] and KaZaA [3], are generally static and consistent, other P2P systems, including OceanStore [6] and Publius [1], allow users to modify their own data, making replicas of modified data inconsistent. On the other hand, with the rapid evolution of P2P-based applications, P2P systems support frequent content updates, such as online auction, trust management [8], distributed network measurement systems, and distributed games. Inconsistency in these systems would deteriorate the system performance, or even attaint the systems. Thus, a cost-effective consistency maintenance mechanism with short convergence time is highly desired.

- *Z. Li is with the Institute of Computing Technology, Chinese Academy of Sciences, and the Graduate University of Chinese Academy of Sciences, Room 709, No. 6, South Road, Kexueyuan, Zhongguancun, P.O. Box 2704, Beijing 100080, P.R. China. E-mail: zyli@ict.ac.cn.*
- *G. Xie and Z. Li are with the Institute of Computing Technology, Chinese Academy of Sciences, Room 734, No. 6, Kexueyuan, Zhongguancun, P.O. Box 2704, Beijing 100080, P.R. China. E-mail: {xie, zcli}@ict.ac.cn.*

Manuscript received 4 Mar. 2007; revised 26 Nov. 2007; accepted 25 Feb. 2008; published online 7 Mar. 2008.
Recommended for acceptance by K. Hwang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2007-03-0066.
Digital Object Identifier no. 10.1109/TPDS.2008.46.

In P2P systems, we call the node having a replica of the object indexed by key k as the replica node of k. Replica nodes of key k constitute a replica group: $group_k$. When an object is modified legally on a replica node, the updated content should be delivered to all the members in $group_k$ as soon as possible. Since the centralized approaches are sufficient for small replica groups (i.e., the groups with size smaller than 10), this paper mainly focuses on consistency schemes for *large* replica groups such as replica groups of popular objects and large-scale distributed game systems. To achieve this goal, an overlay network is established for each replica group, which transforms the multicast problem to a broadcast one within the scope of the overlay.

Several solutions [11], [13], [25], [26] have been proposed for solving the consistency problem in P2P systems. Those designs, however, have one or several following limitations: 1) incurring a lot of duplicate update messages, 2) only offering a probabilistic consistency, and 3) propagating updated contents on locality-ignorant structures, so that consuming unnecessary bandwidth (e.g., bisection backbone bandwidth), and inefficient in terms of network resource consumption.

In this paper, we propose a scalable locality-aware consistency maintenance method for heterogeneous P2P systems. An auxiliary hierarchical overlay network is established for each replica group: the upper layer is Distributed Hash Table (DHT) based and consists of powerful and stable replica nodes of the key, and the ordinary replica nodes (ORNs) at the second layer attach to physically close upper layer nodes. When an updated replica emerges on a node, the corresponding upper layer node dynamically initializes a $d\text{-}ary$ source-specific tree, which is called update message propagation tree (UMPT),

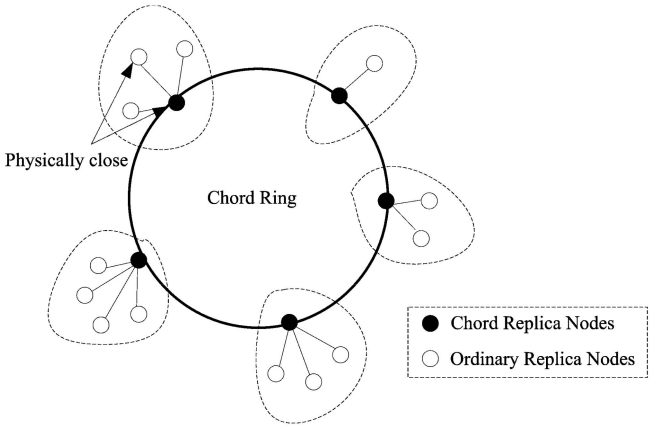


Fig. 1. Hierarchical structure. Powerful and stable replica nodes, called CRNs, are organized in a Chord ring. ORNs attach to physically close CRNs.

on top of the upper layer by continuously partitioning the DHT identifier space. The updated content is propagated along this tree. After all replica nodes in the group have received the updated content, the UMPT is destroyed. Therefore, the tree structure does not need to be maintained all the time.

The major contributions of this work are as follows:

- By taking network locality information and the heterogeneity of node capacity into consideration, we design an efficient consistency maintenance scheme for heterogeneous P2P systems with shorter convergence time and lightweight bandwidth consumption.
- Thorough theoretical analyses show that our scheme is time efficient and cost effective, as well as highly scalable.
- We conduct comprehensive simulations, and results show that compared with locality-ignorant scheme, our approach reduces the cost by about 25-67 percent.

The rest of this paper is organized as follows: Section 2 gives an overview of our scheme, followed by a detailed description of its design in Section 3. In Section 4, we analyze the performance of our scheme theoretically. In Section 5, we evaluate our scheme through extensive simulations. A survey of related work is presented in Section 6. We conclude this work in Section 7.

2 Overview

In this section, we present a brief overview of our solution, deferring the detailed description to the next section. In the following parts, we use Chord [9] as a representative DHT protocol for analysis and description, but it is straightforward for other DHT protocols.

Fig. 1 captures the hierarchical model in our scheme. Replica nodes of a key are organized in a two-layer structure. The upper layer is Chord based and consists of more powerful and stable nodes. We call the replica nodes at the upper layer as *Chord Replica Nodes* (CRNs). Each node at the second (lower) layer attaches to a physically close CRN, and nodes at this layer are denoted as ORNs. Obviously, a CRN and the ORNs attaching to it constitute a cluster, and the CRN acts as a rendezvous point of the cluster. In order to avoid overloading CRNs and relieve the impact of the single point of failure problem, an upper bound on the number of ORNs that a CRN can be attached should be imposed. We call this upper bound as *cluster capacity* and denote it as m_c. The default value of m_c is 16.

An update operation initiated by an ORN is first submitted to its corresponding CRN. When a CRN receives an update request message or initiates an update operation by itself, it dynamically builds a $d\text{-}ary$ UMPT on the upper layer by continuously partitioning the Chord identifier space. The updated content is delivered along the UMPT in a top-down fashion. In addition to transferring the updated content to the child CRNs on the update tree, a CRN also delivers the updated content to the ORNs attaching to it. Since CRNs are physically close to the ORNs attaching to them, this enables fast convergence and saves the bandwidth consumption, or our scheme is *locality aware*. After the update operation completes, the update tree is destroyed in a bottom-up fashion.

Before delving into the details, we define some notations. The total number of nodes in the system is denoted as N. The number of replica nodes of key k is denoted as n, and the number of CRNs of key k is denoted as n_c. Replica nodes have different capacities. Replica node x's capacity is denoted as c_x, which specifies the number of replica nodes in $group_k$ to which x is willing to send updated content concurrently. We assume nodes can estimate the capacities (e.g., storage, bandwidth, and CPU speed) by themselves. In practice, node expected uptime should also been taken into consideration as in Gnutella protocol [2].

3 Design

In this section, we describe the detail of our consistency maintenance scheme. We begin with locality information generation and hierarchy architecture construction, then describe how to build the UMPT and how to propagate updated content. We end this section with the maintenance and failure recovery mechanism.

3.1 Generating Locality Information

We take advantage of landmark clustering scheme [19] to generate locality information. m nodes are picked up randomly from the Internet as landmark nodes. A replica node measures the distance (i.e., delay) to these landmark nodes and obtains a landmark vector $< d_1, d_2, \ldots, d_m >$, where d_i is the distance from the replica node to the $i\text{th}$ landmark node. Replica nodes use their landmark vectors as coordinates in an $m\text{-dimension}$ Cartesian space, which is called landmark space. The intuition behind is that physically close nodes are likely to have similar or close landmark vectors, and thus close to each other in the landmark space. As pointed out in [19], landmark clustering is a coarse-grain approximation and not very effective in differentiating nodes within close distance. However, we just simply use it because coarse-grain information is adequate for our scheme, and the simulation results also show that approximate information works well.

```
x.join_group()
1: x measures the distance to landmark nodes, and computes its landmark number ld_x
2: crn = crn_0.find_successor(ld_x)  /* find_successor is provided by Chord[9]*/
3: vector v_CRN ← crn.GetCRNs(ld_x, c_x)
4: while v_CRN.empty() is false do
5:     crn_1 ← the node with landmark number closest to ld_x
6:     if crn_1.Can_Attach() then
7:         x attaches to crn_1 and selects other w CRNs randomly from v_CRN as backup CRNs
8:         return
9:     else
10:        v_CRN.erase(crn_1)
11:    end if
12: end while
13: if  v_CRN.empty() is true  then
14:    if  c_x ≥ d  then      /*join as an CRN*/
15:        x joins the upper layer as a CRN based on Chord protocol and publishes its information on the upper layer.
16:    else
17:        x asks crn_0 to locate a random CRN crn' that crn'.Can_Attach() == true
18:        x attaches to crn'
19:    endif
20:endif

crn.GetCRNs(landmark_number ld_x, capacity c_x)
1:    R_x = (R_0 × α) / c_x
2:    crn locates the CRNs whose landmark number lie in the range R = {r : |r − l| < R_x}, and pushes these CRNs back to the
      vector v_CRN
3:    return v_CRN

crn.Can_Attach()
1:    if  current cluster size <  m_c  and (d + current cluster size) < c_crn  then
2:        return true
3:    else
4:        return false
5:    endif
```

Fig. 2. Algorithm for new replica node x of key k joining the hierarchical structure associated with k.

As described later, we only need a one-dimension key to represent a node's position in the landmark space. This key is denoted as *landmark number*. Thus, another challenge is related to map m-dimension landmark vectors to one-dimension landmark numbers while preserving the network locality. Space-filling curves [20] are good choices for that they map an m-dimension point to a one-dimension point without loss of proximity. One example of space-filling curves is Hilbert curve. We partition the landmark space into 2^{mr} smaller grids with equal size, where r is the order of Hilbert curve and controls the number of grids used to partition the landmark space. Then, we fit a Hilbert curve on the landmark space to number each grid. Replica node whose landmark number falls into grid ld has the landmark number as ld. Due to the proximity preserving property of Hilbert curve, closeness in landmark number indicates physical closeness. We denote the landmark number of replica node x as ld_x. In current design, the default values of m and r are 15 and 2, respectively.

3.2 Constructing Hierarchical Architecture

Each CRN y publishes its information on the upper layer, based on Chord protocol, by storing its landmark number ld_y and address on the successor of identifier ld_y. Therefore, the information of physically close nodes, whose landmark numbers are similar or close, will be stored closely on the Chord ring.

When a replica node of key k needs to keep its replica up to date, it joins the structure related to k. This operation is called *subscribe* in SCOPE [11]. We assume replica nodes learn the information of an existing CRN (denoted as crn_0) in the upper layer by some external mechanism as in [9]. When joining an existing group, a new replica node of key k first tries to find a close CRN in $group_k$ to attach (i.e., becoming an ORN). In detail, each new replica node x runs the routine *join_group()* as described in Fig. 2.

Node x first gets its landmark number, and then locates nearby nodes by searching a range on the upper layer with its landmark number as center and R_x as radius, and

$$R_x = (R_0 \times \alpha)/c_x, \tag{1}$$

where R_0 is the basic search radius, c_x is the capacity of the joining replica node, and α is a design parameter. After that, x tries to attach to a nearby CRN. The intuition behind is that the bigger a node's capacity is, the higher probability it is a CRN. Since each CRN in the Chord ring maintains a continuous identifier space and the information of CRNs

```
X.region_partition(region_x)
1: if  X.id + 1 > region_x.end  then  /*There is only one node (say X) in this region.*/
2:    return;
3: end if
/* There is no node between [regon_x.start, X.id). Thus, node X is pruned from further partition.*/
3: region←(X.id + 1, region_x.end);
4: Split region into d partitions with equal size
5: for i=1 to d  do
6:    region[i] ←the i-th partition;
7:    RPN_region[i] = X.get_rpn(region[i]);
8:    if  RPN_region[i] != NULL  then
9:       X.children = X.children ∪ RPN_region[i];
10:      RPN_region[i].region_partition(region[i]);
11:   end if
12: end for

X.get_rpn(region)
1:  id ←first ID of this region;
2:  node←X.find_successor(id);
3:  if  node.id ∉ region  then
4:     return NULL;
5:  else
6:     return node;
7:  end if
```

Fig. 3. Algorithm for building update tree on the structure.

with close landmark numbers is stored closely in Chord ring, locating nearby CRNs should be fast.

If there is no nearby CRN that can satisfy the attachment request of x, x's subsequent operation depends on its capacity c_x: if c_x exceeds d, x joins as a CRN and publishes its information on the upper layer; otherwise, x asks the well-known CRN crn_0 to locate a random CRN crn' which still can accept an ORN, and then attaches to crn', where d is average out degree of CRN in the UMPT. Thus, d acts as a capacity threshold to estimate whether a replica node is CRN capable. In practice, in addition to node capacity (e.g., RAM, CPU speed, and bandwidth), other metrics, such as expected uptime, firewalled or not, should also been taken into consideration as in Gnutella protocol [2].

Following the above joining mechanism, a more powerful node still has a probability to become an ORN of a close CRN with smaller capacity. This may occur when a less powerful replica node joins first, and then a more powerful and physically close replica node joins. To promote powerful and stable replica nodes to the upper layer, each ORN periodically evaluates itself to decide whether it is CRN capable or not. If an ORN is much more powerful than its CRN (e.g., two times more powerful), it rejoins as a CRN.

Intuitively, a bigger cluster capacity (m_c) will help the nearby replica nodes being grouped into one cluster with higher probability. However, a bigger cluster capacity may also increase the number of ORNs that attach to a CRN. This may not only overload the CRNs but also increase the recovery cost of a CRN failure. Thus, we need a compromise here. If the cluster capacity m_c is 0, all the replica nodes are formed in a flat Chord ring. Therefore, no locality information is taken into account. We call the consistency maintenance scheme with m_c equal to 0 as *locality-ignorant* one. Correspondingly, we call the consistency maintenance scheme with m_c equal to 16 (the default value of m_c in our experiments) as *locality-aware* one. We evaluate the impact of m_c in the simulation experiments (Section 5.2.1).

Another thing worth pointing out is that when a new replica node joins as an ORN, it stores other w close CRNs as backup CRNs. This is used to increase the fault tolerance. The default value of w is 4. We defer the detail of maintenance in later sections.

3.3 Building Update Message Propagation Tree

An ORN submits the update operation to its corresponding CRN. When a CRN receives an update request message or initiates an update operation by itself, it dynamically builds a *d-ary* UMPT on the upper layer by continuously partitioning the Chord identifier space. Initially, the CRN, crn_i, holds the whole identifier space. This identifier space is partitioned into d parts with equal size. We choose the first CRN of each part as the *representative node* to hold the identifier space of this part and set these d representative CRNs as the children of crn_i. Each part is further partitioned into d parts with equal size, and so on, until there is only one CRN in this identifier part. The pseudocode is listed in Fig. 3. The function *find_successor(id)*, provided by the Chord protocol, is used to find the successor node with the *id*. The function of *get_rpn(region)* is to get the representative node of *region*.

Fig. 4 shows a Chord ring consisting of 10 CRNs and a corresponding *2-ary* UMPT. Initially, crn_0 holds the whole identifier space [0, 15]. The whole identifier space is partitioned into two parts: one is [1, 8] and the other is [9, 15]. crn_1 and crn_{10} are the representative nodes of region [1, 8] and region [9, 15], respectively, for that both crn_1 and crn_{10} are the first replica nodes in their corresponding regions. crn_1 and crn_{10} continue partitioning their regions as crn_0 does. Since there is no node in region [14, 15], the right child of crn_{10} is null.

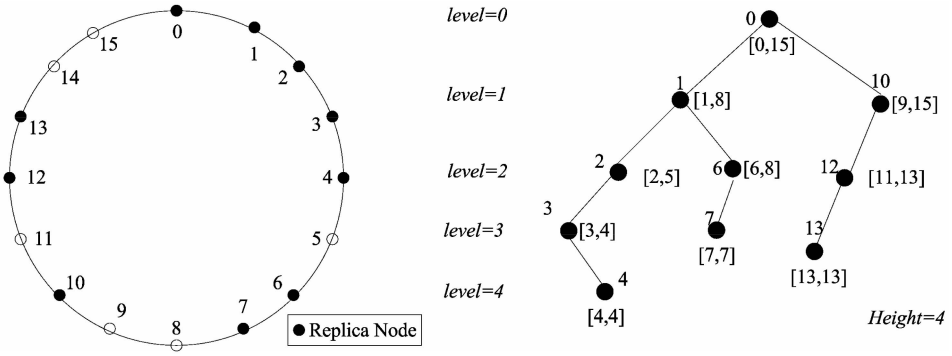


Fig. 4. Chord ring and a 2-ary update tree built based on this ring.

3.4 Propagating Updated Content

After building the UMPT, updated content is propagated along the tree in a top-down fashion. In addition to transferring the updated content to the child replica nodes on the update tree, a CRN also delivers the updated content to the ORNs that attach to it. When receiving the latest updated content, a replica node checks and verifies the data, updates its content, and forwards the content if necessary. Due to the tree structure, each replica node will receive only one update message. Moreover, since CRNs are physically close to the ORNs that attach to it, consistency maintenance converges fast, and bandwidth consumption is greatly saved. After the update operation completes, the UMPT is destroyed in a bottom-up fashion. The detail of UMPT destroying and the failure recovery mechanism during the update period are detailed later.

The reason why we build the update tree dynamically and destroy it after the update operation is that, to fully utilize the system resources, we should use multiple update trees to propagate updated contents. Otherwise, if we use only one update tree, most nodes are leaf nodes that would make no contribute for content delivering. In our opinion, the maintenance cost of multiple update trees is larger than the cost of building $d\text{-}ary$ UMPTs when necessary in dynamic P2P systems, considering that only $O(n_c \times \log d)$ Chord query messages are required to build an UMPT (proved in Section 4.2), where n_c is the number of CRNs of key k.

3.5 Maintenance of Hierarchical Structure

If a replica node rejoins after a period of departure, it may miss updated contents during this period. As in [13], we use a pull scheme: after rejoining the hierarchical structure, CRNs ask their successors for the latest content and ORNs ask their CRNs for the latest content.

When a replica node of key k does not need to keep the replica of k up to date any more, it leaves current structure of key k. This operation is called *unsubscribe* in SCOPE [11]. Leaving mechanism for an ORN is simple: sends a *Leave Message* to the corresponding CRN, and then leaves. When receiving the *Leave Message* from an ORN that attach to it, the CRN deletes the stored information for this ORN. If a CRN leaves, it selects the most reliable powerful replica node from its ORNs to replace itself. The selected ORN rejoins as a CRN and takes over other ORNs that attach to the leaving CRN. If there is no ORN powerful enough to replace the leaving CRN, the cluster is split into several small groups and powerful nodes are chosen from each group to act as CRNs for these groups. Then, the CRN leaves the upper layer based on the Chord protocol and unsubscribes its information from the Chord ring.

Failure of an ORN can be detected by its corresponding CRN through periodical message exchanging, and this has little impact on the structure. When a CRN fails, we resort to Chord protocol to recover the upper layer. Failures of CRNs can also be detected by their ORNs through periodically message exchanging. The ORNs previously attaching to the faulty CRN first try to attach to one of their backup CRNs. If all the backup CRNs of the ORN fail, it rejoins the hierarchical structure. Note that if the failures of CRNs are independent, this scenario is rare. It is worth pointing out that attaching to a backup CRN may impair the proximity effect, as the new CRN may not be the closest one in the upper layer. However, the simulation results show that this has little impact, and our scheme is as effective as usual.

An ORN checks the availability of its backup CRNs periodically. If one half of its backup CRNs is not available any more, an ORN sends a *Backup CRNs Query Message* through its CRN to locate nearby CRNs to fill up its backup CRNs list. The query message is similar to the join message in terms of locating nearby CRNs.

A major concern is the maintenance cost. However, we believe that this is reasonable. First, we do not establish an auxiliary overlay for small replica groups (e.g., groups with less than 10 replica nodes) for that centralized scheme is sufficient. We assume the distribution of replica group sizes is similar to the distribution of objects' popularities, which is a power-law distribution [11]. Then, most of the groups are with small sizes and do not have overlays. For example, for a 10^4-node system with 10^5 keys, if the number of replica nodes of ith most popular key is equal to $1/i$ of the total number of nodes, then only 10^3 such auxiliary overlays are used. Second, the structure is hierarchical and only about 1/10-1/4 replica nodes of a key reside in the DHT-based upper layer (see simulation results in Section 5.2.1), which is the major contribution of the maintenance cost. Third, in large-scale cooperative P2P computing, such as distributed games, all nodes are interested in one event, so each application has only one overlay.

3.6 Maintenance When Propagating Updated Content

Recall that the UMPT is built dynamically when there is an updated content needed to be disseminated, and is destroyed when the update operation completes. Although the convergence time of our scheme is really short, it still has a probability that a replica node leaves and fails during

this short period. Because ORNs are not in the update tree, leaving and failing of ORNs have no impact on propagating updated content. When a CRN in an update tree leaves current structure, in addition to leaving the group according to the leaving scheme, it asks its parent to rebuild the subtree rooted by it and leaves according to the leaving mechanism introduced earlier. Given that CRNs have a good quality of availability and the convergence time of our scheme is short, the leaving scheme is reasonable.

To improve the fault resilience, we use an acknowledgment scheme. A CRN acknowledges its parent CRN as soon as it has received acknowledgments from all child CRNs. When receiving an acknowledgment from a child CRN, the parent CRN deletes the information (e.g., ID, address, and region information) related to this child CRN. This is a recursive process from bottom to top, and the leaf nodes acknowledge their parents as soon as sending the updated content to their ORNs. To ensure a strong consistency, each CRN in the update tree sets a timeout when it forwards the updated content to their child CRNs. If a CRN does not receive all the acknowledgments when the timeout expires, it rebuilds the subtree rooted by the faulty child node and retransmits the updated content along the subtree. To isolate node faults, the timeout intervals of CRNs decrease exponentially with the growth of the levels at which the CRNs reside:

$$timeout_{crn} = T_0 \times e^{-level_{crn}}, \quad 0 \le level_{crn} \le h, \tag{2}$$

where $level_{crn}$ is the level at which the CRN resides, h is the height of the *d-ary* UMPT, and T_0 is a design parameter. In this way, a CRN failure is restricted to the subtree rooted by the parent of faulty CRN with high probability.

4 Analysis

4.1 Analysis of Replica Nodes Join and Leaving

When a new replica node of key k joins, to locate the close CRNs on the upper level, $O(\log n_c)$ Chord query messages are required, where n_c is the number of CRNs of key k. If the new node joins as a CRN, another $O(\log^2 n_c)$ Chord joining messages are required. Thus, when a new replica node joins, on average

$$\#Msg_{join} = \log n_c + p_c \times \log^2 n_c, \tag{3}$$

messages are required, where $p_c \in [0,1]$ is the probability that a new replica node joins as an CRN.

The leaving operation of an ORN only uses one message (i.e., notifying its CRN). Suppose that a CRN leaving causes its cluster split into s smaller group. A CRN leaving uses $O(\log^2 n_c)$ Chord leaving messages and causes s nodes to rejoin as CRNs. The average number of messages used by a replica node leaving is

$$\begin{aligned}\#Msg_{leaving} &= \frac{n-n_c}{n} \times 1 + \frac{n_c}{n} \times (\log^2 n_c + s \times \log^2 n_c) \\ &= 1 + \frac{n_c}{n} \times \left[(s+1)\log^2 n_c - 1\right],\end{aligned} \tag{4}$$

where n is the number of replica node of key k.

As mentioned before, replica node join and leaving operations are called subscribe and unsubscribe operations in SCOPE [11], respectively. In SCOPE, for a structured P2P system with N nodes, a subscribe/unsubscribe operation can be finished in $O(\log^2 N)$ hops, which means at least $O(\log^2 N)$ messages would be generated in one operation. Since N (the total number of nodes in the system) is always bigger than n (the number of replica node of a key), our scheme is at least comparable to SCOPE in terms of the cost of join and leaving operations.

4.2 Analysis of UMPT

To propagate the updated content to all online replica nodes, an UMPT should be built first, and then the updated content is disseminated along with the UMPT. Thus, the update time needed by an update operation consists of two parts: the time for building UMPT and the time for propagating updated content.

Theorem 1. *The average height of a d-ary UMPT is $O(\log_d n_c)$, and a CRN resides only in one level on the update tree, where n_c is the number of CRNs of key k.*

Proof. Suppose the identifier length is v, that is to say the whole identifier space is 2^v. Each partition generates d smaller equally sized regions, each with size of $1/d$ of the previous partition region. After $\log_d n_c$ time partition, the generated region is reduced to $2^v/d^{\log_d n_c} = 2^v/n_c$. In Chord, nodes are distributed on the ring randomly. Thus, the average number of nodes in the region with size of $2^v/n_c$ is 1. This is the termination condition of our partition method. So, the average height of Chord tree is $\log_d n_c$.

Note that current node ID is excluded from the region for further partition. Thus, a CRN resides at only one level in the update tree. □

A CRN delivers the updated data to its child CRNs and its ORNs concurrently. Therefore, after building the update tree, updated content can be propagated to all replica nodes in $O(\log_d n_c)$ time.

Lemma 1. *If an M-node Chord ring is partitioned into r regions with equal size, then the successor of the first ID in the partitioned region can find the successor of a key in this region in $O(\log(M/r))$ logical hops, on average.*

Proof. Suppose the identifier length is v, the whole identifier space is 2^v. We denote node x_i as the successor of the first ID of the ith region r_i. Now, we analyze the logical hops required to find the successor of key q, where q is in region r_i and $q > x_i$. According to the Chord protocol, each hop halves the distance from the query node to the successor of q. Thus, after $\log(M/r)$ hops, the distance between the query node to the successor of k is, at most, $2^v/r/2^{\log(M/r)} = 2^v/M$. Due to the random distribution of nodes in the Chord ring, the average number of nodes in the region with size of $2^v/M$ is 1. Thus, x_i has found the successor of key q. □

Theorem 2. *A d-ary update tree can be built in $O(\log^2 n_c)$ logical hops, on average, where n_c is the number of CRNs of key k.*

Proof. From Theorem 1, the height of update tree is $h = O(\log_d n_c)$. In level $i (0 \le i \le h)$, the Chord ring is partitioned into d^i regions with equal size. From Lemma 1, a CRN in level i can find a child CRN in $\log(n_c/d^i)$ hops. Since the regions are partitioned

concurrently on the CRNs, the average number of logical hops can be expressed as follows:

$$\begin{aligned}\#hop(build) &= d \times \log n_c + d \times \log \frac{n_c}{d} + \cdots + d \times \log \frac{n_c}{d^h} \\ &\approx d \times \left(h \log n_c - \frac{h^2}{2} \times \log d \right) \\ &= d \log^2 n_c / 2 \log d.\end{aligned}$$

Given that d is constant and relatively small, it yields $\#hop(build) \approx O(\log^2 n_c)$. □

Corollary 1. *An update operation completes in $O(\log^2 n_c + \log n_c)$ time.*

Since n_c is much smaller than the total number of replica nodes of key k (about 1/10-1/4 of total replica nodes in our simulation), our scheme is time efficient and have a good quality of scalability as well.

Now, we analyze the Chord query messages used by the construction of a *d-ary* update tree.

Theorem 3. *With n_c CRNs, to build a d-ary update tree, $O(n_c \times \log d)$ Chord query messages will be used.*

Proof. In level $i (0 \le i \le h)$, the Chord ring is partitioned into d^i regions with equal size. From Lemma 1, to find all d child nodes, a node in level i will use $O(d \times \log(n_c/d^i))$ Chord query messages. Thus, on average, the total number of query messages $(\#Msg(build))$ is

$$\begin{aligned}\#Msg(build) &= d \times \log n_c + d^2 \times \log \frac{n_c}{d} + \ldots + d^h \times \log \frac{n_c}{d^{h-1}} \\ &= \log n_c \times \sum_{i=1}^{h} d^i - \log d \times \sum_{i=1}^{h-1} i d^{i+1} \\ &= \log n_c \times \frac{d(d^h - 1)}{d - 1} - \log d \\ &\quad \times \left(\frac{(h-1)d^{h+1}}{d-1} - \frac{d^2(d^{h-1} - 1)}{(d-1)^2} \right) \\ &\approx n_c \times \frac{d}{d-1} \times \log d \\ &\approx O(n_c \times \log d).\end{aligned}$$

□

From Theorem 3, the average number of query messages per CRN to build a *d-ary* UMPT is $O(\log d)$. Therefore, a smaller d is preferred. However, from Corollary 1, decreasing d will increase the convergence time of the update operation. Therefore, a tradeoff is required on the selection of d. We evaluate the impact of d in the simulation experiments (Section 5.2.2).

4.3 Analysis of Failure Recovery

Theorem 4. *If failures of replica nodes are independent and random, then a failure of a replica node would cause $O(\log n_c)$ duplicate update messages, at most, on average.*

Proof. A failure of an ORN does not cause duplicate updated message. If a CRN fails, we should rebuild the subtree and retransmit the updated content. In the worst case, the descendant CRNs of the faulty CRN and their ORNs have received updated content before, thus all the retransmitted messages are redundant. Suppose that there are $(n - n_c)/n_c$ ORNs that attach to a CRN on average, and the faulty CRN resides in the ith level in UMPT with probability of $p_i = d^i/n_c$, then, on average, the total number of duplicate update messages caused by a failure of a replica node is $\#Msg_{rdt}$ at most:

$$\begin{aligned}\#Msg_{rdt} &= \frac{n_c}{n} \times \sum_{i=1}^{h-1} \left[p_i \times \left(1 + \frac{n - n_c}{n_c} \right) \times \sum_{j=1}^{h-i} d^j \right] \\ &= \sum_{i=1}^{h-1} \left[\frac{d^i}{n_c} \times \frac{d[d^{h-i} - 1]}{d - 1} \right] \\ &= \frac{d}{d-1} \times \frac{1}{n_c} \times \sum_{i=1}^{h-1} [d^h - d^i] \\ &< \frac{d}{d-1} \times \frac{1}{n_c} \times \sum_{i=1}^{h-1} d^h \\ &= \frac{d}{d-1} \times (\log_d n_c - 1) \\ &\approx O(\log_d n_c - 1).\end{aligned}$$

□

5 Performance Evaluation

5.1 Experiment Setup

We develop an event-driven simulator, which runs on Linux system, to evaluate our consistency maintenance scheme and compare it with SCOPE [11] and gossip-based hybrid push/pull scheme [13].

In the simulations, the upper layer is Chord (30-bit identifier space) based. The basic search radius for locating nearby CRNs (R_0) is set to 20, and the design parameter α is set to the expectation value of replica nodes capacity.

To simulate the heterogeneity of node capacity, we used two capacity profiles [31]:

- *Pareto distribution.* Node capacity is generated using a Pareto distribution with the shape parameter $a = 2$ and the scale parameter $b = 16$. The samples outside the range [16, 500] are discarded.
- *Gnutella-like.* The replica node capacity is set according to the measured bandwidth distributions of [22] and [24]. Capacity is separated into five levels by an order of magnitude. We assign capacity of 1, 10, 10^2, 10^3, and 10^4 to replica nodes with probability of 20 percent, 45 percent, 30 percent, 4.9 percent, and 0.1 percent, respectively.

To evaluate the efficacy of our locality-aware scheme, two different transit-stub topologies are generated by GT-ITM [21]. Both topologies have about 2,500 nodes.

- *ts2.5k-small.* 4 transit domains each with 4 transit nodes, 5 stub domains attached to each transit node, and 30 nodes in each stub domain on average.

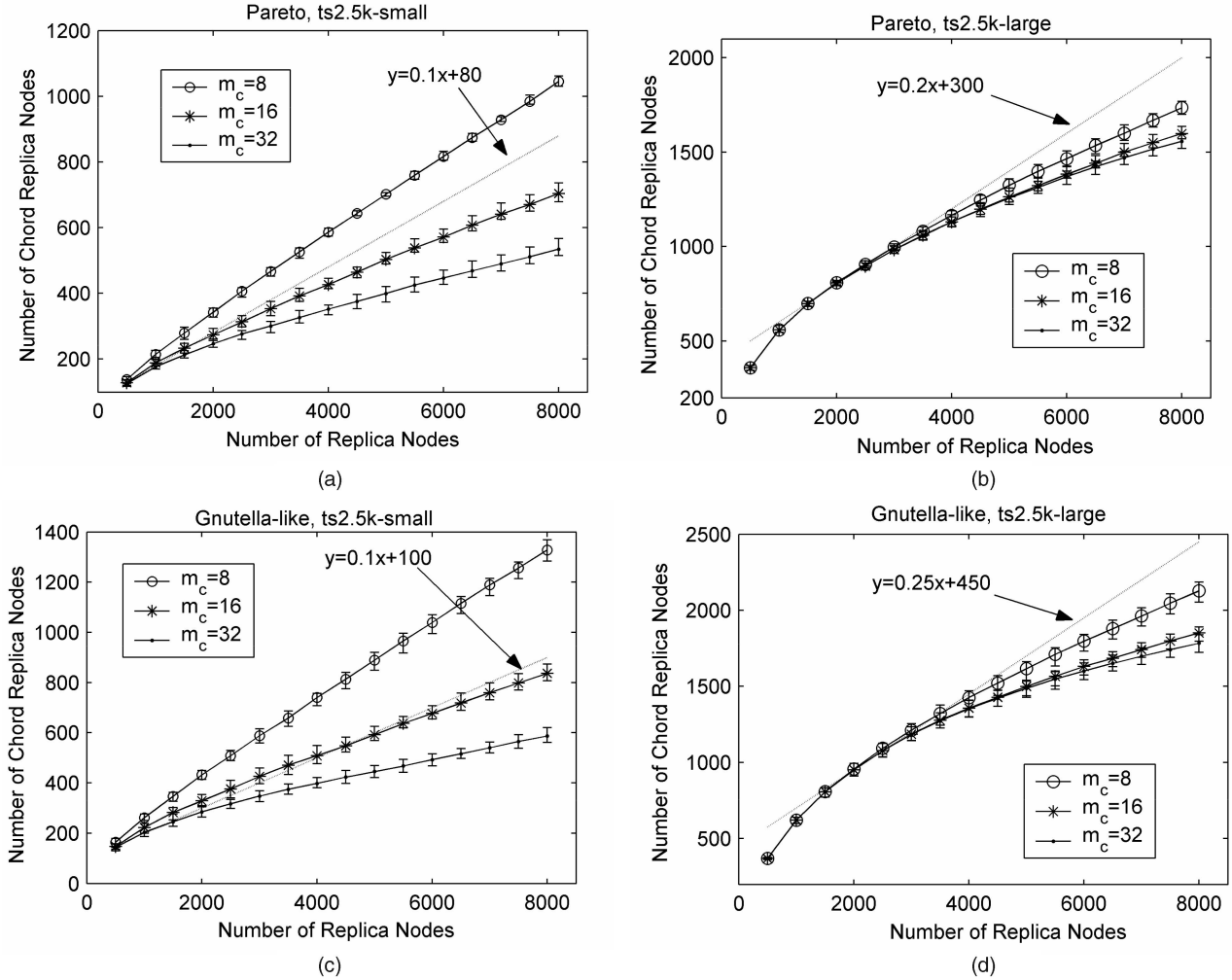


Fig. 5. Number of CRNs while varying the cluster capacity. (a) Pareto capacity profile and ts2.5k-small topology. (b) Pareto capacity profile and ts2.5k-large topology. (c) Gnutella-like capacity profile and ts2.5k-small topology. (d) Gnutella-like capacity profile and ts2.5k-large topology.

- *ts2.5k-large.* 70 transit domains each with 4 transit nodes, 4 stub domains attached to each transit node, and 2 nodes in each stub domain on average.

Note that the generated topologies are Internet topologies and the nodes act as routers in Internet. The nodes in transit domains and stub domains are called transit routers and stub routers, respectively. *Replica nodes randomly attach to the stub routers.* Intuitively, "ts2.5k-large" has a larger backbone and sparser edge network (stub) than "ts2.5k-small" and it represents a situation in which the replica nodes spread over the entire Internet. We assign different distance to the edge according to the edge type: the distance of the edge between a replica node to the router it attaches is 1 hop of unit of latency; the distance of intradomain edge is 2 hops of units of latency; the distance of the edge between transit and stub domain is 10 hops of units of latency; and the distance of intertransit edge is 50 hops of units of latency.

To achieve a fair comparison, the node *fanout* in hybrid push/pull scheme is set as the value of d. The probability that a replica node forwards the updated content $PF(t)$ is set as in [13], that is $PF(t) = 0.9^t$, where t is a counter which counts the number of push rounds that have already been executed. Updated content stops rumoring when 95 percent replica nodes have received the content. In SCOPE, the number of partitions at each level is set as the value of d. We focus on the replica consistency of one key k, and the number of replica nodes of key k is one half of the total number of the system, which means $n = N/2$.

The last thing worth pointing out is that the data points in our plots represent the average and 95 percent confidence interval values of 10 trials, unless noted otherwise.

5.2 Experiments Results

5.2.1 Number of Chord Replica Nodes

Intuitively, increasing cluster capacity (m_c) will increase the number of ORNs attached to a CRN and, hence, decrease the number of CRNs. When m_c is equal to 0, the number of CRNs is equal to the number of total replica nodes. Fig. 5 illustrates the number of CRNs while varying the cluster capacity with different node capacity profiles and different topologies. From the figures, we find three interesting results: 1) When increasing m_c from 8 to 32, the decrease of the number of CRNs (n_c) in ts2.5k-small topology is more obvious than that in ts2.5k-large one. This can be explained by the fact that in ts2.5k-large, the replica nodes spread sparsely in the entire Internet. Thus, there are fewer replica nodes that can be grouped in a cluster in ts2.5k-large topology. 2) The capacity profile has little effect

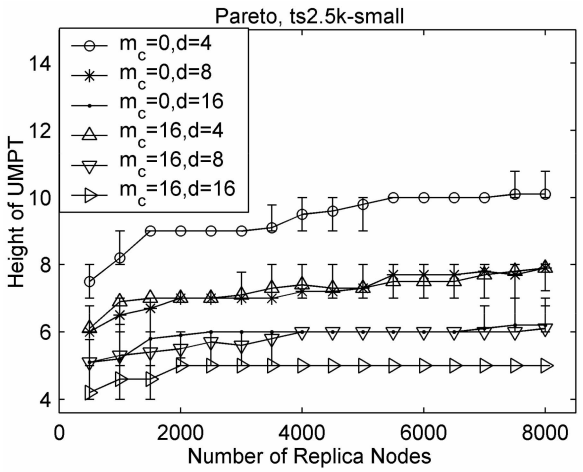


Fig. 6. Height of UMPT with Pareto capacity profile and ts2.5k-small topology.

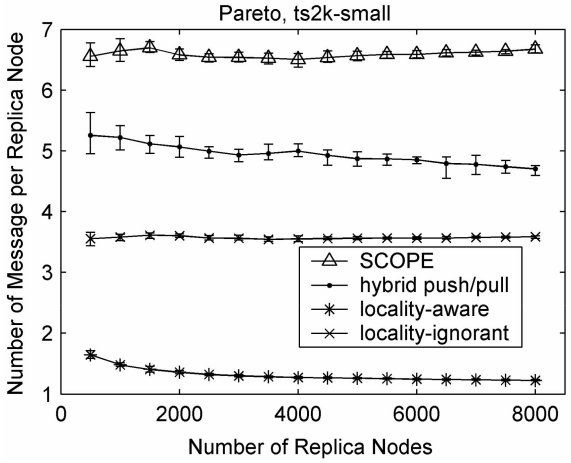


Fig. 7. Number of message per replica node with Pareto capacity profile and ts2.5k-small topology.

on n_c. 3) n_c is about one order of magnitude less than the total number of replica nodes in ts2.5k-small topology and about 1/5-1/4 of the total number of replica nodes in ts2.5k-large topology.

As mentioned in Section 3.2, however, a bigger cluster capacity may also increase the number of ORNs that attach to a CRN, which not only overloads the CRNs, but also increases the recovery cost of a CRN failure. In the current design, we set the default value of m_c to 16 and leave the issue of how to choose an optimal range of m_c to our future work.

5.2.2 Height of UMPT

The height of UMPT has direct influence on the update time and the cost of an update operation. Fig. 6 shows the height of UMPT on the number of replica nodes with Pareto capacity profile in ts2.5k-small topology under different experiment parameters. The results with other capacity profiles and topologies are similar. When increasing cluster capacity m_c from 0 to 16, the number of CRNs decreases, which in turn causes the height of UMPT decrease. Additionally, the height of UMPT almost decreases logarithmically with the growth of UMPT degree. However, from Theorem 3, a larger d would increase the cost of UMPT construction. In current design, the default value of d is set to 8 as the height of UMPT is already very small when d is 8.

5.2.3 Number of Messages for an Update Operation

Fig. 7 illustrates the average number of messages per replica node used for an update operation with Pareto capacity profile and ts2.5k-small topology. For hybrid push/pull scheme, only the update messages are counted in, while for others, the DHT query messages are also included.

SCOPE uses 6.5-6.8 messages per replica node, in which about 4.8 messages are DHT query messages for traversing Replica-Partition-Trees (RPT). This is mainly because, except the replica node of key k, other nodes in the system are also involved in the update operation. The hybrid push/pull scheme uses 4.5-5.5 update messages per replica node for that a replica node may receive update messages from several other replica nodes. In locality-ignorant scheme $(m_c = 0)$, the number of messages used is about 3.6 on average, two times more than the number of messages used in locality-aware scheme $(m_c = 16)$, which uses only about 1.2-1.6 on average. This is explained by the fact that much more Chord query messages are used for building UMPT in the locality-ignorant one. Thus, the locality-aware scheme is more efficient than others in terms of the number of messages used for an update operation.

5.2.4 Cost for an Update Operation

The cost of an update operation *Cost(update)* is defined as follows:

$$Cost(update) = \sum_{i=1}^{u} size\ of(message) \times dst, \qquad (5)$$

where u is the number of messages for an update operation, and *dst* denotes *message* delivered at the distance of *dst* hops of units of latency. We set the size of update message as 1,000 bytes and the size of query message as 27 bytes (i.e., 20 bytes for querying ID, 6 bytes for address information of source node and 1 byte for marking).

Fig. 8 illustrates the cost per replica node for an update operation with Pareto capacity profile and different topologies. Compared with hybrid push/pull scheme, the locality-aware scheme reduces the cost by about one order of magnitude, in both "ts2.5k-small" and "ts2.5k-large". The locality-aware scheme is also more effective than the locality-ignorant scheme and SCOPE in terms of the cost. However, the cost reductions in "ts2.5k-large" are not as significant as in "ts2.5k-small". This is explained by the fact that replica nodes spread over the entire network in "ts2.5k-large" topology, and hence, the number of replica nodes belonging to the same domain is relatively small.

Fig. 9 shows cumulative distribution of transferred messages with Pareto capacity profile and two different topologies. Intuitively, more messages are transferred within shorter distances indicates less cost and faster convergence. In Fig. 9a, we can see that the locality-aware scheme transfers about 28 percent of total update messages within 6 hops and about 91.2 percent within 30 hops, while the locality-ignorant scheme only transfers about 16.4 percent of total

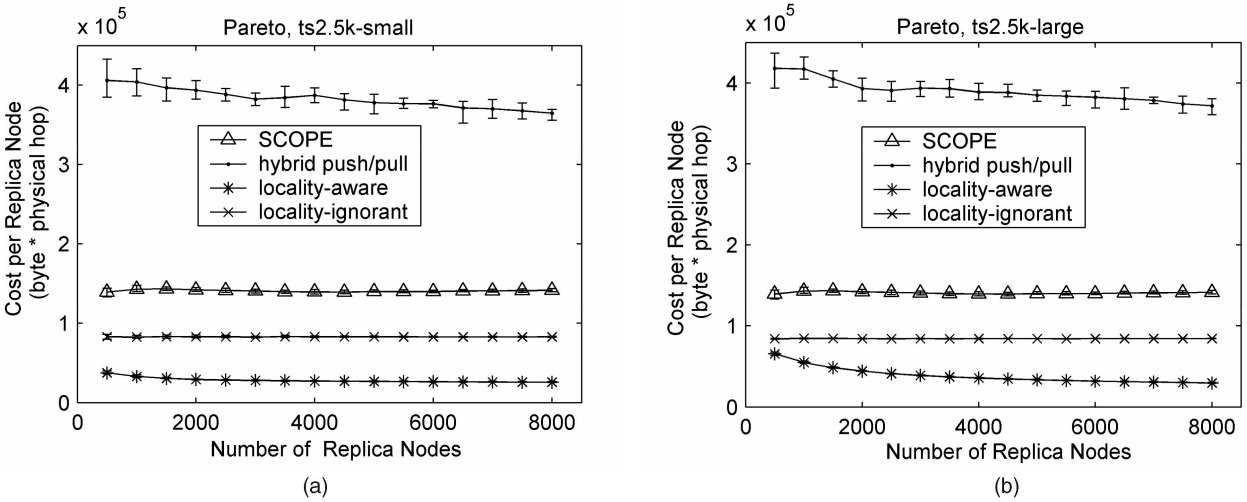


Fig. 8. Cost per replica node for an update operation. (a) Pareto capacity profile and ts2.5k-small topology. (b) Pareto capacity profile and ts2.5k-large topology.

update messages in 30 hops. In Fig. 9b, we can see that the locality-aware scheme transfers about 33 percent of total update messages within 6 hops and about 75 percent within 30 hops, while the locality-ignorant scheme only transfers about 2 percent of total update messages in 30 hops. Such a big difference implies that the locality-aware scheme can effectively transfer update messages between physically close replica nodes. Consequently, the locality-aware scheme greatly reduces the cost of update operations.

Note that, in Fig. 9, the results of hybrid push/pull scheme and SCOPE are similar to the results of locality-ignorant scheme. This is because all of them choose the replica nodes to transfer update messages in a random way. Another interesting result is that the locality-aware scheme can transfer about 23 percent of total update messages in 2 hops in ts2.5k-large topology—that is, a large number of update messages are transferred among the replica nodes attached to the same router node. The reason is that, in ts2.5k-large topology, the number of stub routers is relatively small. Recall that replica nodes can only attach to stub routers. Thus, with the same number of replica nodes, there are more replica nodes that attach to one stub router.

To further quantify the cost for an update, we compute the 99th percentile cost per replica node with various system parameters and show the results in Table 1. These results agree with the ones showed before and illustrate some new findings. These findings can be concluded as follows:

1. The locality-aware scheme is more sensitive to the system scale. This stems from the fact that the larger the scale is, the more the replica nodes can be grouped in a cluster.
2. The locality-aware scheme is more sensitive to the topology type. The reason is that that replica nodes spread over the entire network in "ts2.5k-large" topology. Thus, the number of replica nodes that can be grouped in a cluster is relatively small.
3. The capacity profile has little impact on the update cost for all schemes.
4. Compared with SCOPE, locality-ignorant scheme and hybrid push/pull scheme, the locality-aware scheme

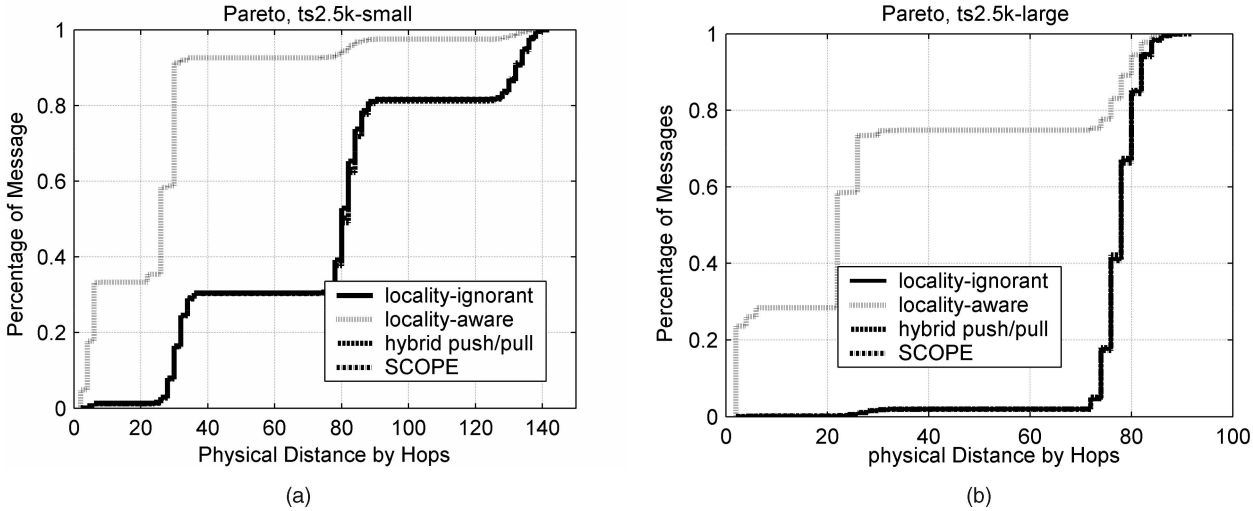


Fig. 9. Cumulative distribution of transferred messages with 5,000 replica nodes. (a) Pareto capacity profile and ts2.5k-small topology. (b) Pareto capacity profile and ts2.5k-large topology.

TABLE 1
99th Percentile Cost per Replica Node ($\times 10^4$ byte $*$ physical hop)

Algorithm	Locality-ignorant			Locality-aware			Hybrid push/pull			SCOPE		
Sys. Scale	1000	5000	8000	1000	5000	8000	1000	5000	8000	1000	5000	8000
Cost with Pareto Cap.	8.45	8.37	8.33	3.50	2.75	2.64	42.1	38.9	37.0	14.79	14.21	14.36
Cost with Gnutella-like Cap.	8.63	8.39	8.42	3.78	2.90	2.75						

(a)

Algorithm	Locality-ignorant			Locality-aware			Hybrid push/pull			SCOPE		
Sys. Scale	1000	5000	8000	1000	5000	8000	1000	5000	8000	1000	5000	8000
Cost with Pareto Cap.	8.47	8.43	8.45	5.65	3.42	3.02	43.2	39.1	38.1	14.47	14.23	14.46
Cost with Gnutella-like Cap.	8.48	8.44	8.45	6.32	3.86	3.34						

(b)

(a) ts2.5k-small topology. (b) ts2.5k-large topology.

reduces the update cost by about 56-83 percent, 25-67 percent, and one order of magnitude, respectively.

5.2.5 Fault Tolerance

When a CRN fails, the updated content should be retransmitted, which may bring some duplicate update messages. Fig. 10 illustrates the number of update messages when some replica nodes fail randomly. The number of update messages per node is the ratio of the total update messages to the number of online replica nodes. In Fig. 10, we see that the number of update messages per node increases proportionally with the percentage of failed replica nodes in both locality-ignorant scheme and locality-aware scheme. Although the failures of replica nodes have a relatively small impact on the hybrid push/pull scheme, the number of update messages is still about two times more than other schemes, even when 20 percent replica nodes fail. Compared with the locality-ignorant scheme, the locality-aware one brings fewer update messages. The reason is that, for the same number of replica nodes, there are more CRNs in locality-ignorant scheme (i.e., equal to the total number of replica nodes), and from Theorem 4, the redundancy increases logarithmically with the number of CRNs.

5.2.6 Impact of Replica Node Churn

In this set of experiments, we evaluate our scheme under churn. First, there is no node failure. Then, we have 10 percent replica nodes that fail. Finally, we have some other replica nodes join and the number of joining replica nodes is the same as the number of failing replica nodes. For each circumstance, the costs of update operations are computed and plotted in Fig. 11. We see that after 10 percent replica nodes' failure, the cost per replica node is slightly larger than the cost in other two cases. This is because, when a CRN fails, its ORNs select other CRNs as their new CRNs, and the new CRNs are always farther away than the original CRN. After replica nodes rejoin, the cost per node drops back, even smaller than the case without any failure. Another observation is that, as the number of replica nodes increases, the cost per replica node drops almost exponentially. This is because the

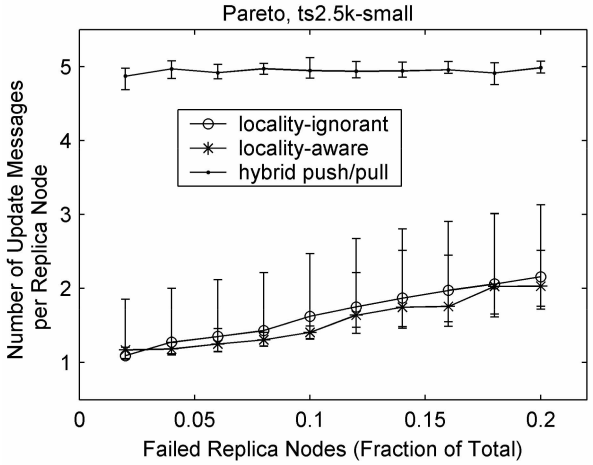


Fig. 10. Number of update messages per replica node. The number of replica nodes is 5,000. The x-axis denotes the number of failed replica nodes normalized by 5,000.

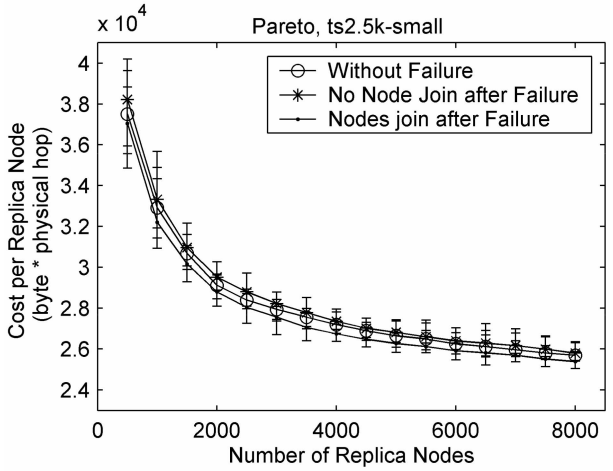


Fig. 11. Cost per replica node under node churn in locality-aware scheme with Pareto capacity profile and ts2.5k-small topology.

larger the scale is, the more the replica nodes can be grouped in a cluster. We can deduce that our scheme is resilient to replica nodes churn.

6 Related Work

6.1 Consistency Maintenance Scheme

Replication and caching have been adequately deployed in distributed systems such as Gnutella [2], CFS [10], and CAN [7]. All these systems resort to naive centralized methods to maintain consistency. The consistency of Web proxy caching is studied in [4] and [5]. However, in their contexts, the proxies are always available. Therefore, these methods are not applicable for dynamic environment, which is a prominent characteristic of P2P systems.

Datta et al. propose a hybrid push and pull consistency maintenance scheme for highly unreliable P2P systems [13]. They take advantage of gossip as a group management protocol, and update messages are rumored to other replica nodes. While suitable for unreliable P2P systems, this method only offers probabilistic guarantee of replica consistency. In addition, it uses a large number of duplicate update messages. In [14] and [25], a hybrid scheme combining flooding-based push and adaptive polling is proposed for P2P file sharing systems. The flooding-based push uses even more duplicate update messages. Moreover, it is hard to determine the polling frequency, thus essentially no guaranteed consistency is provided [11].

Wang et al. [26], [33] propose a replica chain-based scheme for consistency maintenance in unstructured P2P systems. Replica nodes of a key are organized in a chain structure, and each node maintains information of several (typically 10) other replica nodes. Their method only guarantees a probabilistic consistency. The authors also propose a consistency scheme through virtual servers (VS) [27]. The scheme resorts to VS of a key to synchronize update operation and propagate updated content. The maintenance of VS is a time and cost-consuming work, which limits the scalability.

CUP [28] is an incentive-based algorithm to cache lookup results and keep them updated in a structured P2P system. However, it only caches the metadata, not the object itself, along the lookup path with limited consistency support. Thus, it cannot maintain consistency among the replicas of an object [11]. DUP [29] improves CUP by eliminating intermediate nodes on the message propagation path. It suffers from the same limitations as CUP.

SCOPE [11] is for structured P2P systems. Each replica group is associated with a single *RPT* to track the replica nodes of the same key and propagate updated content. A single tree for all the replica nodes of a key is not a good choice. This is because an updated content may emerge on every replica node in the replica group, if only one tree is used, then the traffic is concentrated on the links of the tree and the tree may be partitioned beyond repair for a dynamic group. In addition, an RPT for a key involves other nodes in the systems, which would prolong the delivery delay and increase the network resource consumption. Another limitation of RPT is that a node (e.g., the root node) may reside at several levels, which would make this node overloaded and vulnerable. Our method also builds tree (UMPT) structure by partitioning the identifier space. However, UMPT is built on the hierarchical structure associated with a replica group. Therefore, it only involves the replica nodes of the same key, shortening the deliver delay and saving the network resource consumption. As to tree building procedure, UMPT differs from RPT in two aspects. First, current node ID is excluded from the region for further partition. So, a replica node only appears once in UMPT. Second, in UMPT, the first replica node of the partition region is chosen as representative node. So, the representative node can locate any node in the region very quickly. Note that in some DHT protocols, such as Chord, the query message can only proceed along the clockwise direction.

A major and common limitation of the consistency maintenance schemes mentioned above is that they ignore the proximity information. Thus, they would consume unnecessary backbone bandwidth and adversely affect the convergence time of consistency maintenance operation.

6.2 Locality-Aware Structure Construction

Hierarchy architecture has been implemented in Gnutella [2] and also used in [12] and [16]. In Gnutella, nodes with higher reliability and capacity are elected as supernodes. In [12], supernodes in Gnutella are organized into a structured P2P fashion to increase the hit rates of rare data objects. However, it mainly focuses on locating rare data items and ignores the network locality. Cluster-based schemes are proposed in [16]. However, both the upper layer and second layer are organized in structured P2P fashions. In addition, the performance under node failure is not analyzed or simulated.

Shen and Xu [30] propose a landmark clustering-based hierarchical structure to account for proximity information. However, they directly use node Hilbert number as logical node ID in the auxiliary DHT-based structure. Thus, the identifier randomness in the structure does not exist, which is a necessary condition for the balance of UMPTs in our scheme. In addition, no upper bound is imposed on the number of regular nodes that can be assigned to a supernode. Therefore, the supernodes may be overloaded. Zhu and Hu [31] use landmark clustering to achieve proximity-aware load movement. Each node publishes its load information into the DHT overlay with the landmark number as the DHT key. Its structure is not hierarchical, and it is possible that large number of nodes publish their load information to one node, which may overload the node.

Tewari and Kleinrock [17] analyze the benefits of proportional replication in P2P networks. They focus on replica creation. In [18], a distributed membership management service is proposed for QoS-sensitive P2P applications. Liu et al. [34] propose a bipartite overlay scheme to solve the topology mismatch problem by identifying and replacing the mismatched connections. The authors in [32] point out that a proximity optimization can be imposed on Chord ring without affecting the query complexity. These works are largely complementary to the work presented in this paper.

7 Conclusion

This paper presents a novel scalable consistency maintenance scheme for heterogeneous P2P systems. We mainly

focus on fast delivering updated contents to all the replica nodes with lower cost. To achieve these two goals, we organize replica nodes of a key in a two-layer locality-aware hierarchy model: the upper layer is DHT based, consisting of powerful and stable replica nodes. Replica nodes at the lower layer attach to physically close upper layer nodes. We build update trees dynamically to propagate the updated contents. An efficient failure recovery mechanism is also proposed to improve fault tolerance. On average, for n replica nodes of a key with n_c upper-layer replica nodes, an update operation completes in $O(\log^2 n_c)$ time and only $O(n)$ update messages are required. Theoretical analyses and simulation results have shown that our scheme has a good quality in terms of scalability and fault tolerance. Specially, experiment results show that, compared with locality-ignorant scheme, our approach reduces the cost by about 25-67 percent.

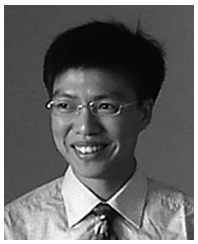
ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable comments. This work was supported by the National Basic Research Program of China with Grant 2007CB310702 and by the National Natural Science Foundation of China with Grants 60403031 and 90604015. An early version of this work [23] was presented in the *Proceedings of IPDPS '07*.

REFERENCES

[1] M. Waldman, A.D. Rubin, and L.F. Cranor, "Publius: A Robust, Tamper-Evident, Censorship-Resistant Web-Publishing System," *Proc. Ninth USENIX Security Symp.,* Aug. 2000.
[2] *Gnutella Protocol Specification v0.6,* http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html, 2008.
[3] *KaZaA,* http://www.kazaa.com, 2008.
[4] V. Duvvuri, P. Shenoy, and R. Tewari, "Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 5, pp. 1266-1276, Sept./Oct. 2003.
[5] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, "Hierarchical Cache Consistency in a WAN," *Proc. Second USENIX Symp. Internet Technologies and Systems (USITS '99),* Oct. 1999.
[6] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-Free Global Data Storage," *IEEE Internet Computing,* vol. 5, no. 5, pp. 40-49, Sept. 2001.
[7] S. Ratnasamy, P. Francis, M. Handley, and R. Karp, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM '01,* Aug. 2001.
[8] K. Aberer and Z. Despotovic, "Managing Trust in a Peer-2-Peer Information System," *Proc. 10th Int'l Conf. Information and Knowledge Management (CIKM '01),* Oct. 2001.
[9] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM '01,* Aug. 2001.
[10] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS," *Proc. 18th ACM Symp. Operating Systems Principles (SOSP '01),* Oct. 2001.
[11] X. Chen, S. Ren, H. Wang, and X. Zhang, "SCOPE: Scalable Consistency Maintenance in Structured P2P Systems," *Proc. IEEE INFOCOM '05,* Mar. 2005.
[12] B.T. Loo, R. Huebsch, I. Stoica, and J.M. Hellerstein, "The Case for a Hybrid P2P Search Infrastructure," *Proc. Third Int'l Workshop Peer-to-Peer Systems (IPTPS '04),* Feb. 2004.
[13] A. Datta, M. Hauswirth, and K. Aberer, "Updates in Highly Unreliable, Replicated Peer-to-Peer Systems," *Proc. 23rd Int'l Conf. Distributed Computing Systems (ICDCS '03),* May 2003.
[14] J. Lan, X. Liu, P. Shenoy, and K. Ramamritham, "Consistency Maintenance in Peer-to-Peer File Sharing Networks," *Proc. Third IEEE Workshop Internet Applications (WIAPP '03),* June 2003.
[15] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. 16th ACM Int'l Conf. Supercomputing (ICS '02),* June 2002.
[16] R. Tian, Y. Xiong, Q. Zhang, B. Li, B.Y. Zhao, and X. Li, "Hybrid Overlay Structure Based on Random Walk," *Proc. Fourth Int'l Workshop Peer-To-Peer Systems (IPTPS '05),* Feb. 2005.
[17] S. Tewari and L. Kleinrock, "Proportional Replication in Peer-to-Peer Networks," *Proc. IEEE INFOCOM '06,* Apr. 2006.
[18] J. Liang and K. Nahrstedt, "RandPeer: Membership Management for QoS Sensitive Peer-to-Peer Applications," *Proc. IEEE INFOCOM '06,* Apr. 2006.
[19] Z. Xu, C. Tang, and Z. Zhang, "Building Topology-Aware Overlays Using Global Soft-State," *Proc. 23rd Int'l Conf. Distributed Computing Systems (ICDCS '03),* May 2003.
[20] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier, "Space Filling Curves and Their Use in the Design Geometric Data Structures," *Theoretical Computer Science,* vol. 181, no. 1, pp. 3-15, July 1997.
[21] E.W. Zegura, K.L. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," *Proc. IEEE INFOCOM,* 1996.
[22] S. Saroiu, K. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking (MMCN '02),* Jan. 2002.
[23] Z. Li, G. Xie, and Z. Li, "Locality-Aware Consistency Maintenance for Heterogeneous P2P Systems," *Proc. 21st Int'l Parallel and Distributed Processing Symp. (IPDPS '07),* Mar. 2007.
[24] Q. Lv, S. Ratnasamy, and S. Shenker, "Can Heterogeneity Make Gnutella Scalable," *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* Mar. 2002.
[25] X. Liu, J. Lan, P. Shenoy, and K. Ramamritham, "Consistency Maintenance in Dynamic Peer-to-Peer Overlay Network," *Computer Networks,* vol. 50, no. 6, pp. 859-876, Apr. 2006.
[26] Z. Wang, S.K. Das, M. Kumar, and H. Shen, "Update Propagation through Replica Chain in Decentralized and Unstructured P2P Systems," *Proc. Fourth Int'l Conf. Peer-to-Peer Computing (P2P '04),* Aug. 2004.
[27] Z. Wang, M. Kumar, S.K. Das, and H. Shen, "File Consistency Maintenance through Virtual Servers in P2P Systems," *Proc. 11th IEEE Symp. Computers and Comm. (ISCC '06),* June 2006.
[28] M. Roussopoulos and M. Baker, "CUP: Controlled Update Propagation in Peer-to-Peer Networks," *Proc. USENIX Ann. Technical Conf.,* 2003.
[29] L. Yin and G. Cao, "DUP: Dynamic-Tree Based Update Propagation in Peer-to-Peer," *Proc. 21st IEEE Int'l Conf. Data Eng. (ICDE),* 2005.
[30] H. Shen and C. Xu, "Hash-Based Proximity Clustering for Load Balancing in Heterogeneous DHT Networks," *Proc. 20th Int'l Parallel and Distributed Processing Symp. (IPDPS),* 2006.
[31] Y. Zhu and Y. Hu, "Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 4, pp. 349-361, Apr. 2005.
[32] K.P. Gummadi, R. Gummadi, S.D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The Impact of DHT Routing Geometry on Resilience and Proximity," *Proc. ACM SIGCOMM '03,* Aug. 2003.
[33] Z. Wang, S.K. Das, M. Kumar, and H. Shen, "An Efficient Update Propagation Algorithm for P2P systems," *Computer Comm.,* vol. 30, no. 5, pp. 1106-1115, Mar. 2007.
[34] Y. Liu, L. Xiao, and L.M. Ni, "Building a Scalable Bipartite P2P Overlay Network," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 9, pp. 1296-1306, Sept. 2007.

Zhenyu Li received the BS degree in automation from Nankai University, Tianjin, China, in 2003. He is currently working toward the PhD degree in computer science at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. He is also with the Graduate University of the Chinese Academy of Sciences, Beijing. His research interests include peer-to-peer computing, distributed systems, and network measurement. He is a student member of the IEEE and the IEEE Computer Society.



Gaogang Xie received the BS degree in physics, and the MS and PhD degrees in computer science from Hunan University in 1996, 1999, and 2002, respectively. From 2005 to 2006, he did postdoctoral research in INRIA supported by FFCSA. He is currently a professor at the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) and the Deputy Director of Network Technology Research Center at ICT. His research interests include distributed and mobile computing, network measurement, and modeling. He has published more than 70 scientific papers. He is a member of the IEEE and the IEEE Computer Society.

Zhongcheng Li received the BS degree in computer science from Peking University, in 1983 and the MS and PhD degrees from the Chinese Academy of Sciences, in 1986 and 1991, respectively. From 1996 to 1997, he was a visiting professor at the University of California, Berkeley. He is currently a professor in the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His current research interests include next-generation Internet, dependable systems and networks, and wireless communication. He has published more than 100 technical papers. He is the holder of 10 patents. He is a member of the IEEE.