Randomized Protocols for Duplicate Elimination in Peer-to-Peer Storage Systems

Ronaldo A. Ferreira, Murali K. Ramanathan, Ananth Grama, and Suresh Jagannathan

Abstract—Distributed peer-to-peer systems rely on voluntary participation of peers to effectively manage a storage pool. In such systems, data is generally replicated for performance and availability. If the storage associated with replication is not monitored and provisioned, the underlying benefits may not be realized. Resource constraints, performance scalability, and availability present diverse considerations. Availability and performance scalability, in terms of response time, are improved by aggressive replication, whereas resource constraints limit total storage in the network. Identification and elimination of redundant data pose fundamental problems for such systems. In this paper, we present a novel and efficient solution that addresses availability and scalability with respect to management of redundant data. Specifically, we address the problem of duplicate elimination in the context of systems connected over an *unstructured* peer-to-peer network in which there is no a priori binding between an object and its location. We propose two randomized protocols to solve this problem in a scalable and decentralized fashion that does not compromise the availability requirements of the application. Performance results using both large-scale simulations and a prototype built on PlanetLab demonstrate that our protocols provide high probabilistic guarantees while incurring minimal administrative overheads.

Index Terms—Peer-to-peer, unstructured networks, duplicate elimination, randomized algorithms.

1 Introduction

Peer-to-peer systems have emerged as cost-effective alternatives for scalable data sharing, backup, and archival storage [6], [14]. Peers contribute data and storage and, in return, gain access to data at other peers. Effective storage management is an important issue in the deployment of such systems. Data replication and caching are key enabling techniques for scalability, performance, and availability. In this context, an important problem relates to pruning unwanted copies of data efficiently and safely. Attempts at aggressive replication may lead to significant overheads associated with thrashing in resource constrained environments. Even if replication at peers is controlled, as in systems such as Samsara [7], the network as a whole must provide mechanisms for eliminating replicas that are not accessed, while leaving a minimum number of replicas in the network to satisfy availability constraints.

In this paper, we investigate the problem of eliminating duplicate data items in peer-to-peer systems. We examine this issue in the context of unstructured networks [13], where no assumptions can be made about the relationship between an object and the peers at which it resides. Unstructured networks differ from their structured counterparts [30], [25], [26], [31] in several important respects. Structured networks provide a simple primitive for locating an object which relies on a distributed hash table (DHT) abstraction. The associated lookup techniques provide bounds on the number of hops as a function of the number of peers. These bounds are achieved by establishing and maintaining a well-defined overlay topology. In networks with a high transient population, the overhead associated with this may be significant. In contrast to structured peer-to-peer networks, unstructured networks are resilient to node failures and incur low overhead on node arrivals and departures. These characteristics make unstructured networks attractive for use in highly transient networks, where peers do not have significant resources. Unfortunately, the issue of object location, which is central to the problem of identifying redundant copies, is significantly more complex in this environment.

The primary focus of this paper is on systems where peers are cooperative and nonmalicious. Peers divide their storage into two spaces: a private and a public space. The private space contains the peer's data and is not subject to duplicate elimination. The public space holds data from other peers and is subject to duplicate elimination. We can view the public space as backup storage or a cache to facilitate availability and performance, respectively. In such an environment, peers can use any of the conventional object location techniques, such as the ones developed in [29] and [10].

We can now formally state our problem as follows: *Consider an unstructured peer-to-peer system, with each peer hosting multiple data items. The problem is to determine whether each of these data items can be tagged for elimination or must be retained, with the associated guarantee that each data item is retained by at least k peers. Here, k is a system parameter chosen to satisfy a client or application's availability requirements. The associated protocol must have minimum network and computing overhead.*

An important issue for the effective elimination of data in a storage system is the identification of common data at different peers. Adya et al. [1] proposes convergent encryption to solve this problem. With convergent encryption, it is possible to safely store encrypted data at other peers and, at the same time, recognize if two files from different users have the same content. Convergent encryption uses an encryption key derived from the content of the file and has been used in [8] to identify blocks of data with

- *The authors are with the Department of Computer Sciences, Purdue University, 250 N. University Street, West Lafayette, IN 47907-2066. E-mail: {rf, rmk, ayg, suresh}@cs.purdue.edu.*

Manuscript received 19 Sept. 2005; revised 21 June 2006; accepted 30 June 2006; published online 9 Jan. 2007.
Recommended for acceptance by D. Trystram.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0407-0905.
Digital Object Identifier no. 10.1109/TPDS.2007.1019.

the same content. Doucer et al. [8] motivates the importance of duplicate elimination problem and proposes SALAD, a distributed data structure to aggregate file content information and location information. In building this data structure, an underlying structure of the files based on their content and location is maintained; this structure is similar to the index employed by structured peer-to-peer networks. It is unclear how SALAD can be adapted to an unstructured environment. SALAD imposes a topology on the network, while, in an unstructured environment, peers are free to connect to each other at random.

To address these concerns, we present the design, implementation, and evaluation of two novel solutions to the problem of identifying and eliminating duplicates in unstructured peer-to-peer storage systems. Our approaches are applicable to diverse storage environments [27], [11], [14]. Our key insight is that the problem of duplicate elimination can be abstracted to a relaxed and probabilistic version of the leader election problem: We aim to elect a group of k leaders for each file. The elected leaders are responsible for storing the file. If a leader must depart the network, it randomly chooses another node to hold its replicas. Our solution to this problem relies on the premise that the network state in an unstructured peer-to-peer system changes fast enough that deterministic guarantees provided by conventional leader election algorithms are not useful. For this reason, we use our probabilistic leader election protocol, which has significantly lower message and time complexity. Our leader election protocol differs from traditional leader election algorithms in two important respects:

- The probabilistic nature of our protocol may produce more or fewer leaders than desired with small probability, $O(\frac{1}{n})$, for a system with n nodes.
- Other nodes participating in the protocol that are not elected as leaders need not know the identity of the elected leaders.

Our first approach to the elimination problem uses probabilistic quorum systems [20]. In this approach, each node creates a quorum for each of its replicas by choosing $\sqrt{n \ln n}$ nodes in the system uniformly at random and queries the quorum on whether it needs to keep the file. If it receives a negative response from any node in the quorum, the file is discarded. The details on how a quorum member responds to a query are provided in Section 2. In the worst case, if a replica is present at all peers, the total message complexity of the elimination process of this protocol is $O(n\sqrt{n \ln n})$.

In the second approach, we use a novel randomized leader election protocol. The protocol proceeds in two phases. In the first phase, consisting of $O(\log n)$ rounds, the number of nodes that are potential leaders is reduced. The second phase is similar to the probabilistic quorum approach: A quorum is formed for each of the remaining nodes, and k leaders are chosen. The elimination process of this protocol has optimal message complexity $O(n)$ for a network with n nodes.

We evaluate these protocols using real-world file system traces from Microsoft Research [9]. We consider both a detailed simulation to study the scalability of the protocols and a prototype implementation on PlanetLab [23] to quantify administrative overheads and efficiency in real network environments. Our experiments measure the amount of space reclaimed, memory and communication overhead, and the average storage gain per node. The main contributions of the paper are:

- two novel randomized protocols for duplicate elimination in peer-to-peer storage systems, along with analytical insights into their performance,
- an investigation, using detailed simulation, of the interaction between the choice of the number of duplicates to be maintained (availability) and the administrative overheads (scalability), and
- an implementation on PlanetLab using real-world file traces, which validates analytical and simulation-based performance characterization.

The rest of the paper is organized as follows: Section 2 presents a detailed description of the protocols. The experimental and simulation setup and corresponding results are given in Section 3. Related work is presented in Section 4. Conclusions are outlined in Section 5.

2 Duplicate Elimination Protocols

In this section, we describe the duplicate elimination protocols. These protocols rely on leader election as the main kernel, with leaders assigned the responsibility of preserving specified data items, while all other participants are free to remove them.

2.1 A Probabilistic Quorum (PQ) Approach

Our first protocol is based on ideas proposed for building probabilistic quorum (PQ) systems. A quorum is defined as a set of subsets of peers such that any two subsets have a nonempty intersection. To guarantee nonempty intersection with high probability, the number of members in each quorum is equal to $\sqrt{n \ln n}$ [20], where n is the number of peers in the network.

For duplicate elimination, k nodes need to be elected among the nodes holding a specific data item. Each node with the data item creates a quorum of the prescribed size, as specified above. Each node with a candidate data item sends a REQUEST_TO_KEEP message to all its quorum members. It decides to keep or delete the data item based on the response to this message. A REQUEST_TO_KEEP message contains the following information:

- SENDER_ID: identity of the sender,
- DATA_DESCRIPTOR: content hash of the file, and
- RANDOM_NUMBER: a random number locally generated by the node in the range $[0 \dots n^4]$ [19].[1]

Each quorum member waits for a prescribed time period (a system parameter) and processes all the messages received. At the end of the time period, it takes all the messages received for the same DATA_DESCRIPTOR. Positive responses (ACKs) are sent to the k peers that have the largest random numbers among all the messages with the same data descriptor. The ACKs include the random numbers that were selected by the quorum member. Negative responses (NAKs) are sent to the rest of the peers. A peer that receives at least one NAK deletes the data. A peer that receives only positive responses must sort all of the random numbers in the ACKs it receives to verify if its random number is among the top k. If this is the case, the peer retains the data; otherwise, the data is deleted. If a data item is present at all peers in the network

[1] This range is selected for uniqueness, w.h.p.

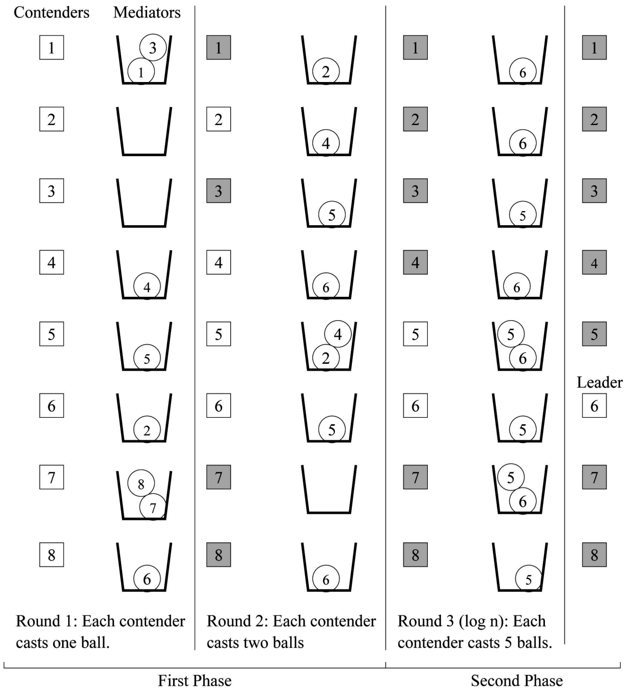

Contenders
Mediators
Leader
Round 1: Each contender casts one ball.
Round 2: Each contender casts two balls
Round 3 (log n): Each contender casts 5 balls.
First Phase
Second Phase

Fig. 1. Illustration of the protocol—contenders are illustrated by squares, mediators by buckets, and messages from contenders to mediators by labeled balls (the label indicating the source of the ball). In all rounds, contenders that are no longer in the running are shaded dark. In the first round, each contender casts one ball and contenders 2, 4, 5, and 6 proceed since their balls uniquely occupy their respective buckets. In round 2 (last round of the first phase), each contender casts two balls and contenders 5 and 6 proceed. Finally, in round 3 (the only round of the second phase), each contender casts five balls and contender 6 is selected the leader since it has a higher node id (in our protocol, we use a random number as the criterion in the second phase. We use node id here for ease of illustration.).

which corresponds to the worst-case scenario, the message complexity of this protocol is $O(n\sqrt{n \ln n})$. Observe that this message complexity does not include the cost for selecting nodes uniformly at random from the network. In Section 2.3, we discuss how a node can obtain uniform samples in an unstructured network. When the PQ protocol is applied separately for each data item, the message complexity must include the number of data items. In Section 2.2.2, we discuss how information about separate data items can be aggregated to keep the message complexity the same as the message complexity for a single data item.

2.2 A Randomized Election (RE) Approach

As an alternative to probabilistic quorum systems, we also consider a randomized leader election protocol based on a *balls and bins* [22] abstraction. The protocol operates in two phases. In the first phase, the number of nodes holding the same data item is reduced. In the second phase, the remaining nodes determine the k leaders that retain the data item. The main features of the protocol are as follows:

- There are k peers holding a copy of the file at the end of the protocol with high probability.
- The message complexity of the elimination protocol is $O(n)$, where n is the number of peers in the system.
- Any peer among the set of peers having a duplicate is equally likely to be elected as one of the k leaders.

To simplify the explanation, we assume that there are multiple copies of a single data item in the entire system and each peer holds one copy of this data item. Although our presentation at this point assumes a synchronous protocol, we show how to relax this assumption in Section 2.2.1.

Before we describe the protocol, we formalize a few terms used in the description. A *contender* is a participating peer in the protocol that holds a copy of the data. A *mediator* is a peer that receives a message from a contender and determines whether the contender participates in subsequent steps of the protocol. The mediator is similar to a quorum member in the probabilistic quorum approach. A *round* is composed of communications between a single contender and a set of mediators. The set of contenders can change with each round of the protocol. A contender that does not proceed to a new round deletes the copy of the data item.

In a realization of this balls-and-bins abstraction, a peer can be a contender as well as a mediator at the same time. Casting a ball into a randomly chosen bin corresponds to sending a REQUEST_TO_KEEP message from a contender to a mediator, picked uniformly at random. The REQUEST_TO_KEEP message has the same fields as the REQUEST_TO_KEEP message in the PQ approach and an extra field (ROUND_NUMBER), indicating the round number.

The protocol is played as a tournament in two phases: The first phase consists of $\log n - \log(c \times k)$ rounds for a system of n nodes, where c is constant and k is a system parameter (the desired number of replicas). In round i of this phase, each contender casts m_i balls into n bins; the precise expressions for m_i in terms of n and i are given in Fig. 2. A contender is said to "win" a bin if its ball is the only one that lands in the bin. If a contender wins all of the bins that its balls land in, it is considered a winner in this round and proceeds to the next round. This process is illustrated in Fig. 1 for $n = 8$, $k = 1$, and $c = 2$. Here, the first phase consists of $\log_2 8 - 1 = 2$ rounds.

The number of mediators that a contender sends messages to (the number of balls to cast) in a round is calculated independently by every contender based on the system size (total number of nodes) and the round number. The formula given in Fig. 2 is an approximation of the exact theoretical value. We present the theoretical value in Section 2.4 with the proofs of the bounds of the algorithm. Each mediator sends an ACK if it receives only a single REQUEST_TO_KEEP message. Otherwise, it sends a NAK to all the contenders that sent a message to it. A contender deletes a data item if it receives at least a single NAK; otherwise, it proceeds to the next round. The number of contenders remaining after each round is reduced by half, on average. The remaining contenders at the end of the first phase proceed to the second phase. The expected number of contenders at the end of the first phase is $c \times k$. The second phase of the protocol is identical to the probabilistic quorum approach, but with a reduced number of contenders. Details about the protocol are summarized in Figs. 2 and 3.

2.2.1 An Asynchronous Protocol

We now relax the assumptions made earlier and provide a solution that supports asynchronous communication among mediators and contenders. In the asynchronous case, messages from contenders to a mediator for a specific round need not be received at the same time. Therefore, in every round of the first phase, a mediator sends an ACK to the *first* contender request for that round. A NAK is sent to subsequent requests from other contenders for that round.

```
▷ Set the number of messages $m_j$ for round $j$ as follows
      $m_j = \begin{cases} \sqrt{2^j \ln 2}, & \text{first phase} \\ \sqrt{n \ln n}, & \text{second phase} \end{cases}$
▷ Select a set of $m_j$ mediators uniformly at random among all the nodes in the system.
▷ For each mediator selected in the previous step, send a REQUEST_TO_KEEP message,
  along with a RANDOM_NUMBER $r_i$.
▷ Proceed to the next round, $j + 1$, if and only if ACKs are received from all the mediators.
  Otherwise, delete the copy of the data item locally.
```

Fig. 2. Contender actions.

In the first phase of the asynchronous protocol, a contender sends REQUEST_TO_KEEP messages, along with a round number to a set of mediators picked uniformly at random, as in the synchronous protocol. A mediator M maintains a vector V of size equal to the number of rounds $(\log n)$. All the entries in V are initially set to zero. On receiving a REQUEST_TO_KEEP from a contender C in round j, if entry j in V at M is zero, M sends an ACK to C and sets the entry j to C (signifying that the winner of the $j\text{th}$ round at M is C). Otherwise, a NAK is sent to C. The purpose of this step is to reduce the number of contenders that proceed to subsequent rounds. The contenders that survive (which do not receive even a single negative response) all of the rounds in the first phase proceed to the second phase of the protocol.

The second phase of the asynchronous protocol follows the protocol for probabilistic quorum given in Section 2.1, but with a constant number of contenders. An analytical modeling of this algorithm is presented in Section 2.4.

2.2.2 Aggregation

We now relax the assumption of a single data item in the entire system and explain how the protocol can be extended to deal with multiple data items efficiently. A simple extension would be to perform leader election for each file in the system separately. However, this approach results in a protocol complexity of $O(n * \# \text{ of unique data items})$. A better approach that reduces message complexity and, thus, improves efficiency is to hold a single election. In this case, a contender, say C_1, initially sends messages to mediators with a list of all the local data items. On collision with contender C_2 at the mediator M, M sends ACKs for files held by C_1 that are not present in C_2. C_1 proceeds to the next round with the list of files for which it received an ACK. The complexity of this method is $O(n\sqrt{n \ln n})$, assuming that the list of data items can be packed into a constant number of messages. The multiplicative factor $\sqrt{n \ln n}$ is present because each node, in the worst case, can go to the second phase of the protocol for a disjoint set of files. Given that the number of unique data items is likely to be significantly larger than the number of nodes in the system, the latter approach is expected to be more scalable. Furthermore, observe that the same aggregation scheme can be applied to the PQ approach. The resulting complexity in this case is also $O(n\sqrt{n \ln n})$.

2.3 Algorithmic Issues

Effective realization of the duplicate elimination protocols presented in the paper builds on a number of recent results on generating random walks and efficiently estimating network size for unstructured networks. In our protocols, nodes need to select quorum members or mediators, uniformly at random. To select nodes uniformly at random from the network, we rely on random walks. Gkantsidis et al. [12] show that it is possible to construct a uniform sample of size k in an unstructured network by performing a random walk of length $\Omega(\log n)$ and then proceeding with the random walk for k more hops. The last k peers encountered in the walk represent a uniform random

```
▷ Receive messages from each contender.
▷ First Phase:
    – If exactly one contender sent a REQUEST_TO_KEEP message, send an ACK to that
      contender and proceed to round $j + 1$.
    – Otherwise, send a NAK to all the contenders that sent a request and proceed to round
      $j + 1$.
▷ Second Phase:
    – Perform the actions described for the Probabilistic Quorum protocol.
```

Fig. 3. Mediator actions.

sample. This result, however, applies to network topologies that can be modeled as expander graphs. For more generic networks, such as the ones encountered in real-world networks, an auxiliary algorithm must be executed to determine probabilistic weights for different branches of the network on which a random walk is performed. The problem of uniform sampling in nonuniform graphs has been extensively studied [21], [15], [2]. For the sake of completeness, we briefly discuss the Metropolis-Hastings (MH) algorithm that we use in our simulations.

2.3.1 Uniform Sampling

In a simple random walk, all neighbors of a node p have the same probability of being selected as the next hop, that is, $Pr_{pj} = 1/K$, where j is one of K neighbors of p. It is well known [22] that a simple random walk in an undirected graph of a minimum length proportional to $\log n$ converges to a stationary distribution π^T. The probability of a node i being selected as a sample of the distribution π^T is equal to $d_i/2m$, where d_i is the degree of node i and m is the number of edges of the graph. In real-world networks, the degrees of the nodes vary significantly. This variation in degrees implies that a simple random walk selects nodes with different probabilities, with low degree nodes having lower probabilities of being selected. We present an algorithm that modifies the transition probabilities between nodes to produce a probability transition matrix that has a stationary uniform distribution. Thus, a random walk on a network with node transition probabilities defined using this algorithm results in a uniform sample. The algorithm is traditionally presented in the context of Markov chains, for example, in [5], but its adaptation to a distributed network is straightforward.

Metropolis-Hastings. This algorithm is an adaptation for uniform sampling of the classical Metropolis-Hastings algorithm [21], [15], [5]. In this distributed algorithm, each node i sends a message containing its degree information, d_i, to each of its neighbors $j \in \Psi(i)$, where $\Psi(i)$ is the set of nodes directly connected to i. Once the information of each of the neighbors is received, the transition probabilities are set up as follows:

$$Pr^{mh}_{ij} = \begin{cases} 1/max(d_i, d_j) & \text{if } i \neq j \text{ and } j \in \Psi(i), \\ 1 - \sum_{j\in\Psi(i)} (Pr^{mh}_{ij}) & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

The result of this algorithm in the entire network is the implicit construction of a doubly stochastic matrix. A random walk of a given minimum length using the probabilities above results in uniform samples. We refer the reader to [21], [15], [5], [2] for more details on the algorithm and its analytical underpinnings. Observe that the message complexities of the elimination protocols presented earlier in this section do not include the cost for uniform sampling. If random walks are used for uniform sampling, an $O(\log n)$ factor must be added to the complexities since we need to perform a random walk of length $O(\log n)$ to each random sample. In this case, the complete RE protocol would have worst-case complexity $O(n \log n)$ and the PQ protocol $O(n\sqrt{n} \log n)$. This cost, however, can be amortized across multiple runs of the elimination protocols.

2.3.2 Network Size

Another input to our protocol is the size of the network. An estimate of the number of processes in the system is needed to determine the number of messages that need to be sent in each round of the protocol. Horowitz and Malkhi [16] presents a scheme that allows a node to estimate the size of its network based only on local information with constant overhead by maintaining a logical ring. Bawa et al. [3] proposes an estimation scheme based on the birthday paradox [22, p. 45]. A peer estimates the network size by sending a message on a random walk and using the hop count when the message returns to the peer. It is shown that it takes approximately $\sqrt{n}$ hops for a message to return to its sender. In our current implementation, we have the system size as a parameter.

2.4 Analysis

In this section, we establish performance bounds of our algorithms. We consider a distributed system of n processes represented by the set $\Gamma_n = \{a_i | 1 \leq i \leq n\}$, for processes $a_1, a_2, \ldots, a_n$. The set of contenders in round j of the protocol is represented by $\Phi_j = \{\rho_i | 1 \leq i \leq |\Phi_j|\}$, where $\Phi_j \subseteq \Gamma_n$, $\Phi_{j+1} \subseteq \Phi_j$, and $\rho_1, \rho_2, \ldots, \rho_{|\Phi_j|}$ are the contenders. The number of messages (*balls*) sent by a contender in a round j is denoted by φ_j. The set of mediators associated with a contender ρ_i in round j is denoted by Ψ_{ij}. We associate a unique integer π_i with every contender ρ_i. The protocol concludes by declaring a unique member in Φ_w to be the final winner, where w corresponds to the final round in the protocol and $\{\forall j : j < w, |\Phi_j| > 1, |\Phi_w| = 1\}$. We show that $w < \log(n)$. For purpose of clarity, we present the proofs for $k = 1$. The generalization for $k > 1$ is straightforward.

Theorem 1. *In the first phase of the protocol, the number of contenders in round $j + 1$ is approximately half the number of contenders in round j.*

Proof. Let X_{ij} be an indicator random variable corresponding to process ρ_i in round j obtaining a positive response from every process in Ψ_{ij}.

$$X_{ij} = \begin{cases} 1, & \text{if } \forall s : s \in \Psi_{ij}, s \text{ sent a positive response} \\ 0, & \text{otherwise.} \end{cases}$$

The expectation of X_{ij} is given by

$$E[X_{ij}] = Pr(X_{ij} = 1) = \left(1 - \frac{\varphi_j}{n}\right)^{\varphi_j(|\Phi_j|-1)} \approx e^{-\frac{\varphi_j^2(|\Phi_j|-1)}{n}}. \tag{1}$$

Let Y_{j+1} be a random variable that represents the number of processes proceeding to round $j+1$. We have $Y_{j+1} = \sum_{i=1}^{|\Phi_j|} X_{ij}$. The expectation of Y_{j+1} is given by

$$E[Y_{j+1}] = E[\sum_{i=1}^{|\Phi_j|} X_{ij}] = \sum_{i=1}^{|\Phi_j|} E[X_{ij}] \approx |\Phi_j| e^{-\frac{\varphi_j^2(|\Phi_j|-1)}{n}}. \tag{2}$$

Since $\varphi_j = \sqrt{\frac{n \ln 2}{|\Phi_j|-1}}$, we have

$$E[Y_{j+1}] \approx \frac{|\Phi_j|}{2}. \tag{3}$$

□

Observe that, in Section 2.2, we present a simplified expression for m_j that uses the fact that the number of contenders is reduced by half in each round.

Theorem 2. *The probability of having at least one contender at the end of the first phase is* $1-\epsilon$, *i.e.,* $Pr(Y_w > 0) \geq 1-\epsilon$, *where* $\epsilon \rightarrow 0$ *as* $n \rightarrow \infty$.

Proof. By using the Second Moment Method, (based on Chebyshev's Inequality), for a nonnegative integer-valued random variable Y_w, we have:

$$Pr(Y_w = 0) \leq \frac{Var(Y_w)}{(E[Y_w])^2} = \frac{E[Y_w^2]}{(E[Y_w])^2} - 1. \tag{4}$$

Let $k = |\Phi_{w-1}|$,

$$\begin{aligned} E[Y_w^2] &= \sum_{i=1}^{k} E\left[Y_{iw}^2\right] + \sum_{i \neq j} E[Y_{iw}Y_{jw}] \\ &= E[Y_w] + k(k-1)E[Y_{1w}Y_{2w}] \\ &= E[Y_w] + k(k-1)\left(\frac{1-\varphi_{w-1}}{n}\right)^{\varphi_{w-1}(2k-3)} \\ &= E[Y_w] + k(k-1)e^{-\frac{\varphi_{w-1}^2}{n}(2k-3)}. \end{aligned}$$

Without loss of generality, assume that $C_n < |\Phi_1|$. At the start of the final round of the first phase, in the worst case, there are at least $C_n + 1$ contenders. By substituting for $|\Phi_{w-1}| = C_n + 1$, $\varphi_{w-1} = \sqrt{\frac{n \ln 2}{|\Phi_{w-1}|-1}}$, $E[Y_w] = \frac{|\Phi_{w-1}|}{2}$ in (4), we have

$$Pr(Y_w = 0) \leq \frac{2}{C_n + 1} + \frac{4C_n}{(C_n+1)2^{\frac{2C_n-1}{C_n+1}}} - 1.$$

Therefore, $Pr(Y_w > 0) \geq 1 - \epsilon$, where

$$\epsilon = \frac{2}{C_n + 1} + \frac{4C_n}{(C_n+1)2^{\frac{2C_n-1}{C_n+1}}} - 1.$$

For $C_n = \Theta(\sqrt{\log n})$, $\epsilon \rightarrow 0$ as $n \rightarrow \infty$. □

Furthermore, if we fix the number of rounds so that the expected number of contenders in the last round of the first phase of the protocol is $\frac{2\log n}{\delta^2}$, where $0 < \delta < 1$ and δ is a constant, then, by the Chernoff bound [22], we have $P(Y_w \leq (1-\delta)E[Y_w]) \leq \frac{1}{n}$. For a typical value of $\delta = \frac{1}{2}$, we have $P(Y_w \leq 4\log n) \leq \frac{1}{n}$. Informally, the number of contenders entering the second phase of the protocol Y_w is close to the expected value w.h.p. Correspondingly, the number of messages in the second phase would be $\frac{2\log n}{\delta^2}\sqrt{n \ln n}$, which still maintains the message complexity.

Theorem 3. *There is exactly one contender remaining at the end of the protocol w.h.p.*

Proof. $\forall u, v : u,\ v \in \Phi_w$, $u \neq v$, the probability that any two sets Ψ_{uw} and Ψ_{vw} intersect is

$$Pr(\Psi_{uw} \cap \Psi_{vw}) = 1 - \left(1 - \frac{\sqrt{n \ln n}}{n}\right)^{\sqrt{n \ln n}} \approx 1 - \frac{1}{n}.$$

Since the sets intersect w.h.p., only the process ρ_i with the highest value of π_i receives positive responses from all processes in Ψ_{iw}. The rest of the contending processes have at least one decline message. Hence, a unique final winner of the protocol is chosen w.h.p. □

Theorem 4. *The number of rounds in the protocol is* $O(\log n)$.

Proof. From (3), it is clear that, in every round of the first phase, the number of contenders reduces by at least half. Since there is only one round in the second phase, the number of rounds in the protocol is $O(\log |\Phi_1|)$. In the worst case, when $|\Phi_1| = n$, the number of rounds of the protocol is $O(\log n)$. □

Theorem 5. *When* $|\Phi_1| = n$, *the total number of messages in the system is* $O(n)$.

Proof. The total number of messages in the system in the first phase is $2\left(\sum_{j=1}^{w-1} |\Phi_j| \sqrt{\frac{n \ln 2}{|\Phi_j|-1}}\right)$. When $|\Phi_1| = n$, the number of contenders in round j is approximately $\frac{n}{2^{j-1}}$, which results in $2n(\ln 2)\left(\sum_{j=1}^{w-1} \frac{1}{2^{\frac{j-1}{2}}}\right)$ messages. Hence, the number of messages in the first phase is $O(n)$ as $\sum_{j=1}^{w-1} \frac{1}{2^{\frac{j-1}{2}}}$ converges.

At the end of the first phase, the number of remaining contenders is no more than C_n. Since each contender sends at most $\sqrt{n \ln n}$ messages in the second phase, the maximum number of messages exchanged in the second phase is $2c \ln n \sqrt{n}$, where $C_n = c\sqrt{\ln n}$ and c is a constant. Hence, the total number of messages in the protocol is $O(n)$. □

When $|\Phi_1| = O(1)$, the total number of messages in the system is $O(\sqrt{n \ln n})$.

3 Implementation and Experimental Results

In this section, we present experimental results for the two protocols described in the previous section. Our experiments consider a detailed large-scale simulation, as well as a prototype implementation on PlanetLab. The simulation setup allows us to quantify the performance of our protocols for various system parameters. This is not easily done for the PlanetLab implementation. For our PlanetLab experiments, we use file traces from real systems.

3.1 Simulation Setup and Results

Recent studies show that the topology of unstructured networks can be modeled using power-law random graphs [28]. In our experiments, we use a power-law random graph with 50,000 nodes. To generate the graph, we first generate the degrees of the nodes according to a power-law distribution with parameter $\alpha = 0.8$ and then connect the nodes randomly.

We assume that a single file is the target of duplicate elimination and that the file is duplicated at a fraction β of the nodes. The file is placed at nodes selected uniformly at random across the network. We vary β from 1 percent to 50 percent. These percentages are chosen to simulate files of differing popularity, from rare files to extremely popular ones; in other words, popular files are duplicated at many nodes, while unpopular ones are not. The desired number of replicas k is varied from 1 to 100 and the value of c is fixed and equal to 2. This value of c is sufficient to guarantee the accuracy of the protocol, as discussed in Section 3.1.4, and it also minimizes the network overhead by allowing a small number (twice the value of k) of peers to participate in the second phase of the protocol.

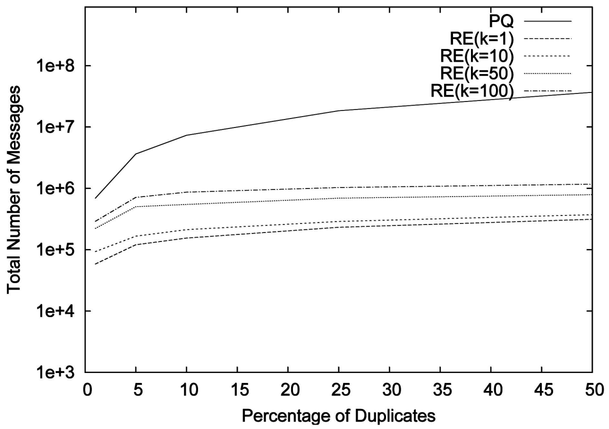


Fig. 4. Total number of messages in the system versus percentage of duplicates.

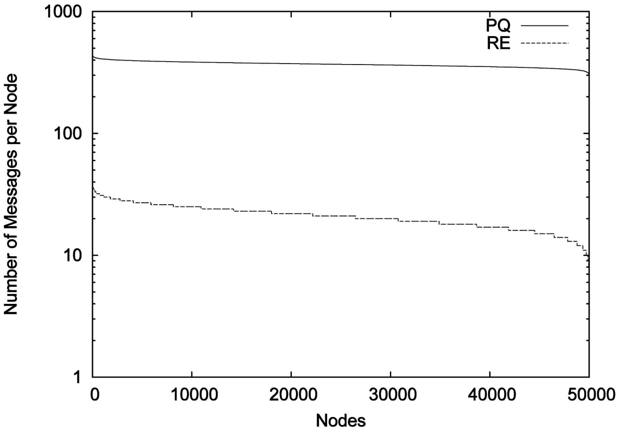


Fig. 5. Number of messages received per node (Y-axis in log scale).

3.1.1 Message Overhead

We first investigate the scalability of our protocols with respect to the total number of messages exchanged in the system. Fig. 4 shows the number of messages for the two duplicate elimination approaches applied to a file with varying replication fraction and with different values of k. In the PQ approach, the parameter k has no impact on the number of messages—the results are identical for all values of k. This is expected since the protocol consists of a single round in which all peers containing a replica send the same number of messages to the quorum members. The decision to delete a file is made locally based on the responses received. This protocol, however, does not scale well when the percentage of duplicates increases in the network. When the file is replicated at 50 percent of the nodes, the number of messages in the PQ protocol is more than an order of magnitude greater than the worst case ($k = 100$) of the RE protocol. This shows that the RE is more resilient to the number of duplicates in the network.

3.1.2 Load Distribution

We evaluate the load at each node in the system with respect to the number of messages handled. We study the load when the percentage of duplicates is 50 percent and $k = 100$, which corresponds to the highest load among the different parameters studied. Fig. 5 shows the number of messages handled by each node. For each approach, the nodes are sorted independently based on the number of messages processed. We only compare the nodes in the figure based on its ranking, ranked by the number of messages processed in the two approaches. The load per node is greater using the PQ protocol than RE due to the larger number of total messages in the former case. Recall that RE eliminates roughly half the contenders at each round, thus significantly reducing the number of messages processed in subsequent rounds.

The load is evenly spread across all nodes in the system for the PQ approach. We can see that the biased random walk using the Metropolis-Hastings algorithm is able to select peers uniformly from the network. We refer the reader to [2] for a survey of uniform sampling algorithms and an extensive evaluation of their parameters. For the RE approach, however, we observe a slight discrepancy in which a small set of nodes process more messages. This is due to an optimization we use in the implementation of the protocol. The winners of each round should pick new sets of mediators in a way that any node can be selected uniformly from the network. However, in our simulation, the winners include the same set of mediators from a previous round in the set of mediators for the new round and only select additional nodes from the network to satisfy the number of mediators for the new round. Therefore, the mediators picked by the nodes that eventually retain the files must process more messages than the rest of the nodes in the system. This simple optimization, however, does not interfere with the accuracy of the protocol, as discussed below.

3.1.3 Different Objects

In a second experiment, we evaluate how the two schemes scale to the number of different objects in the network. We install different objects in the network with different levels of popularity for each object. The number of different objects is varied as a function of the number of nodes in the network. We vary this value from 1 percent of the number of nodes up to 200 percent. In this experiment, the maximum number of different objects is, therefore, 100,000. For each object, we place a different number of replicas. The number of replicas is varied from 10 percent up to 50 percent of the nodes. We perform a single election for each experiment using aggregation, instead of one election per object. We assume that each message can store a maximum of 20 file descriptors. If a node has more than 20 different files, multiple messages have to be sent to the mediators. Fig. 6 shows the comparison of the two protocols in terms of number of messages sent in the network. The number of messages sent by RE is divided by the number of messages sent by PQ. We can see that RE performs much better than PQ when the number of different objects in the network is relatively small, sending less than 20 pecent of the number of messages sent by PQ in the best case shown. As the number of different objects increases, PQ outperforms RE. This result is expected since a single election is performed and a node with multiple objects may lose in the first phase for some objects, but still go to the second phase with other objects. Since the second phase of RE is an execution of PQ with an expected smaller number of contenders, if the contenders are not pruned due to different objects, the second phase of RE may, in the worst case, include all the initial contenders. In this case, the first phase becomes an extra overhead.

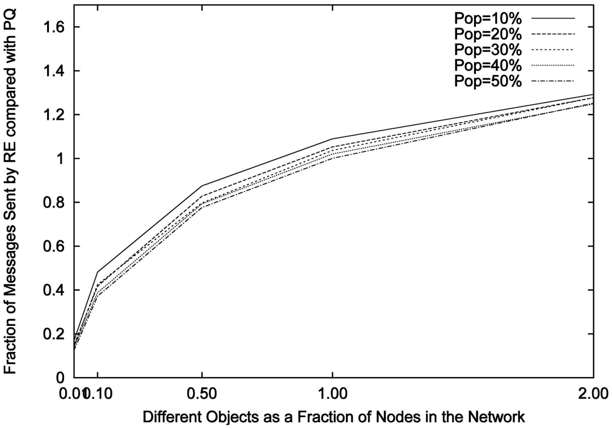


Fig. 6. Comparison of the number of messages sent by the two elimination algorithms. The X-axis shows the number of different objects as a function of the number of nodes in the network. The number of different objects is varied from 0.01 to 2.00 times the number of nodes in the network. The Y-axis show the comparison of the two algorithms, the total number of messages sent by RE is divided by the total number of messages sent by PQ. The different curves represent the number of copies of each object as a function of the number of nodes in the network.

3.1.4 Accuracy of the Protocols

The eventual goal of both protocols is to delete all duplicates while retaining k copies in the network. To evaluate how close each protocol gets to this goal, we fix the object popularity to 1 percent (500 copies) and vary k from 1 to 100. In 10,000 different runs with different seeds, the number of copies left in the network matched the values of k in 99.8 percent of the runs. This indicates that both protocols are highly accurate in achieving the target replica count.

3.2 PlanetLab Experiment

Although PlanetLab is smaller than what we would expect from a large-scale storage system (and what our simulation results measure), it is nonetheless a useful testbed for evaluating the performance of our application and addressing important issues that arise in real implementations that are not captured in a simulation study. These include memory requirements and realistic message delays. In PlanetLab, nodes are spread around the world and are connected to the Internet with varying bandwidths. The message delays are actual processing and network delays and mimic the delays that a large scale application would encounter in a real scenario. Nodes in PlanetLab consist of PCs with Pentium IV processors with at least 512MB of RAM and run Linux as their operating system [23].

In our experiments, we use a separate application (command center) to control data collection and initialization of the nodes. The command center informs the nodes which file systems they should use and how they should connect to each other. The number of (overlay) connections at each node is generated using a zipf-like distribution. Once the nodes are connected, the command center informs the nodes that they can start the elimination protocol. After receiving the start notification, the nodes wait for a random time interval before starting the elimination protocol.

3.2.1 File Traces

The file traces we use in our experiments are data sets obtained from [9]. These traces correspond to 10,568 file systems from 4,801 Windows machines in a commercial environment. The trace contains information about 140 million files with total storage requirement of 10.5 TB. The trace does not contain file contents, only the names and the sizes in bytes of the files. For our experiments in PlanetLab, we were able to consistently acquire 250 nodes to run our application; we run four processes per machine, resulting in a network with 1,000 peers. We randomly choose 1,000 file systems from the trace and assign one file system to each process. We process the obtained trace to eliminate any duplicates within the same file system and, thus, our results do not reflect duplicate elimination of files that belong to the same file system.

A unique ID of 20 bytes is generated for each file within the file system by hashing the file name plus the file size. The file systems are assigned at random to the PlanetLab nodes without considering the storage, bandwidth, and processing power of the nodes, or any specific topological relationship among the nodes. We assume these traces correspond to backed up data on the peers that are subject to duplicate elimination. The characteristics of the file systems used in our experiments, as well as the other parameters are shown in Table 1.

3.2.2 Experimental Results

Our main goal in the PlanetLab experiments is to measure administrative overheads, such as memory usage and messages processed, in a realistic scenario. Since RE is a more complex protocol and includes PQ as its second phase, we show only the results for RE. Our experiments examine the effectiveness and overheads of the protocol as the minimum file size for duplicate elimination is varied. Files below a minimum size are not considered by the protocol. The minimum file size is varied from 1 KB to 10 MB. The following parameters are investigated in our experiments:

- **Total Space Reclaimed:** This reflects the total amount of space reclaimed in the entire system. We show this value to be close to optimal, which is the sum of all the sizes of all unique files in the system.
- **Communication Overhead:** This indicates the total number of messages including REQUEST_TO_KEEP, ACK, and NAK messages in the system.
- **Storage per node:** This parameter gives the file size distribution with respect to the nodes in the system. At the end of the protocol, the space used at each node should decrease as a percentage of the number of duplicates it originally held.
- **Memory Overhead per node:** Each node in the system can potentially act as a mediator. Measuring memory load, the maximum memory used at any point of the execution, helps us in identifying the additional memory needed by the system to support the protocol.

Fig. 7 presents the main results of the experiments. We calculate the initial storage system size by obtaining file system sizes from each node. Similarly, from the file systems used, we find the number of unique files separately for all the file systems used and the corresponding storage size. The unique storage system size is the optimal storage size. The initial and unique storage system sizes are represented by horizontal lines in the figure. We perform the RE protocol for varying minimum file sizes, from 10 KB to 10 MB. Only files greater than the established minimum file size are considered for duplicate elimination. Upon

TABLE 1
PlanetLab Experiment System Parameters

Maximum number of files in a FS	31,533
Minimum number of files in a FS	0
Average number of files per FS	6,719.15
Maximum FS size	4,678MB
Minimum FS size	0KB
Average FS size	653MB
Total space used	86,232MB
Unique files in the system	420,487
PlanetLab nodes	250
Peers	1,000

conclusion of the protocol, we obtain the final file system sizes for each node and compute the final storage system size. As the figure shows, with an increase in the minimum file size, the final storage size deviates from the optimum storage. When the minimum file size is 10 KB, the final storage size is close to the optimum, which is around 50 percent of the initial storage size. In other words, the protocol performs better when the percentage of files considered for duplicate elimination increases, as expected.

The total number of messages processed in the system is also presented on the right Y-axis of the figure. With an increase in the minimum file size, the number of messages exchanged in the system reduces. The presence of the messages processed along with the final storage space gives an approximate estimate of the minimum file size that needs to be used. For example, when the minimum file size is set at 100 KB, the number of messages processed is quite low with final storage space slightly deviating from the optimum storage space. There is a clear trade-off between the space reclaimed as a function of the minimum file size and the message overhead. By increasing the minimum file size, the message overhead is reduced, but the number of duplicates in the system is increased.

We also study the amount of storage reclaimed per node. Fig. 8 presents the initial storage requirement for each node in the system. The results for minimum file size of 10 K and 500 K are shown in the figure. We use only these two for clarity in presentation. The results for other minimum file sizes show a similar pattern. It is clear that, for both minimum file sizes, the amount of storage per node decreases for the majority of the nodes. Furthermore, the storage reclaimed per node when the minimum file size is 10 K is greater than when it is 500 K. This is due to the elimination of a larger number of smaller-sized files.

The amount of memory consumed per node as a mediator is shown in Fig. 9. The nodes are sorted based on the amount of memory consumed and the experiment is executed for different minimum file sizes. When the minimum file size is 10 K, the average amount of memory consumed across all nodes is approximately 7 MB. Though most of the nodes have similar memory consumption, there are a few nodes that have more memory overhead. This is similar to the observations made in the simulation results. Mediators that are part of the mediator sets for the nodes that eventually keep the file receive more messages than the rest. The same observations made in Section 3.1.2 about our implementation apply in this case. Observe, though, that the deviation from average is not significant. The memory consumed reduces with increase in the minimum file size, which is due to fewer number of files selected for duplicate elimination. Due to the uniform selection of mediators, the memory consumed per node is not correlated to the initial storage size per node. Recall that the initial storage at each node is nonuniform (Fig. 8).

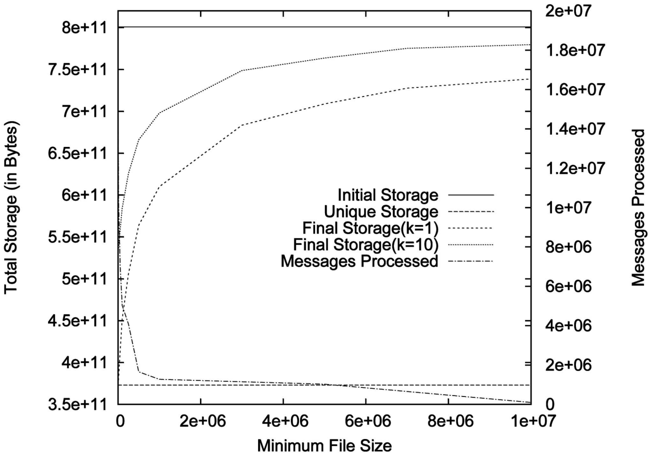


Fig. 7. The X-axis represents the minimum file size for which the duplicate elimination was performed, the Y-axis scale on the left represents the system storage size, the Y-axis scale on the right represents the total number of messages processed in the system. Initial and final storage system sizes along with the unique storage size are also shown.

We evaluate the total number of messages processed per node with varying minimum file size in Fig. 10. For smaller minimum file size, the number of messages processed per node is relatively higher than that for larger minimum file sizes. This corresponds to the total message complexity as shown in Fig. 7. The nonuniformity in the number of messages processed per node for a given minimum file size can be attributed to the nonuniform nature of the initial storage distribution.

4 Related Work

The duplicate elimination problem for wide-area storage systems has been addressed by Douceur et al. [8]. They develop SALAD, a Self Arranging, Lossy, Associative Database, which is a distributed data structure to aggregate file content and location information. The database is distributed across machines, where each machine is a leaf and a group of leaves form a cell. Records are sorted into buckets and buckets are assigned to a cell. By building this data structure, an underlying structure is formed and must be maintained. This is similar to the distributed indexing approach present in many popular structured peer-to-peer networks [30], [25]. We address the more general problem of eliminating duplicates without making any assumptions on the structure of the overlay network.

Pastiche [6] is a peer-to-peer backup system where each peer tries to minimize storage overhead by selecting peers that share a significant amount of data. The problem of greedy hosts was identified in [6] and solutions have been proposed. These solutions are based on equivalence classes, solution of cryptographic puzzles, and various forms of electronic currency. These solutions tend to be complicated

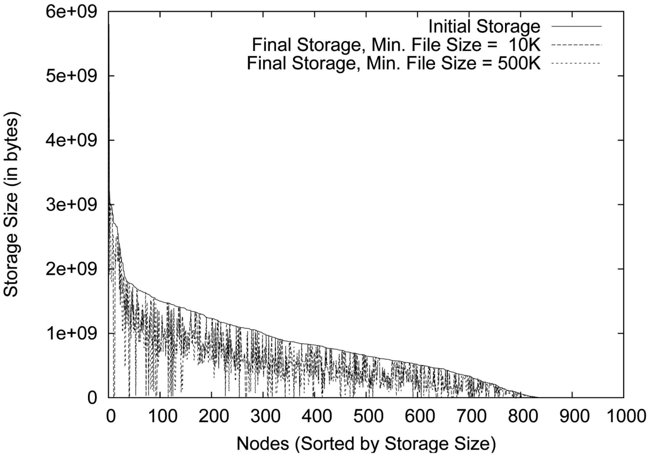


Fig. 8. Storage size per node for varying minimum file sizes. Nodes are sorted based on the initial file size.

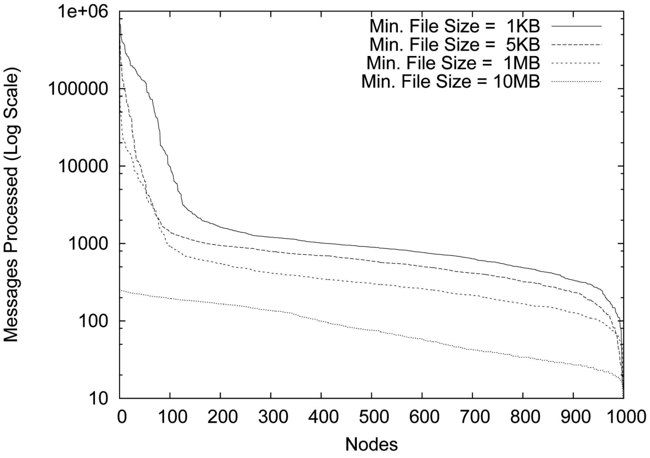


Fig. 10. Messages processed per node with varying minimum file sizes. Nodes are sorted by the number of messages processed.

and involve significant overhead. Cox and Noble [7] revisit this problem and propose Samsara, a fairness enforcement mechanism. In this scheme, each peer that requests storage space from another peer must agree to reserve some space in its own disk to the peer it is requesting the storage from. The scheme addresses the problem of greedy peers, but provides no solution to the problem of common data that are highly replicated by different peers in the network.

Lillibridge et al. [18] propose a cooperative peer-to-peer backup system. The problem of cheating peers is addressed using periodic random challenges to ensure peers hold the data they committed to store. There is no mechanism for identifying and removing duplicate data in the system.

A number of researchers have addressed the problem of duplicate data elimination in storage systems. Kulkarni et al. [17] proposes a Redundancy Elimination at the Block Level (REBL) scheme to eliminate redundant data. This approach uses data compression, duplicate block suppression, and delta encoding among other techniques. Policroniades and Pratt [24] study three different techniques: whole file content hashing, fixed size blocking, and a chunking strategy that uses Rabin fingerprints to discover identical portions of data. Both [17] and [24] focus on eliminating redundancy within a single site. Our work is aimed at eliminating redundant data from a wide-area unstructured storage system with multiple sites. We can certainly employ the techniques presented in these other efforts before executing our protocol so that duplicates within a site are eliminated. Our current implementation uses content hash of entire files. However, it can be extended to eliminate duplicates at the block level, but with an added communication cost.

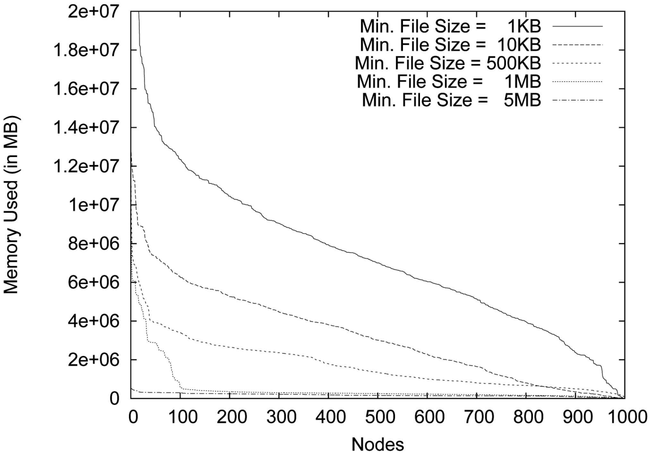


Fig. 9. Memory overhead (in MB) per node for varying minimum file sizes. Nodes are sorted based on the memory overhead.

Bolosky et al. [4] address the problem of multiple copies of the same file in a file system. The Single Instance Store (SIS), a component of Windows 2000 that implements links with the semantics of copies for files, is proposed. SIS uses copy-on-close to implement the copy semantics of its links. This again is a solution that manages space within a single site, while our paper solves the problem of managing space efficiently across multiple sites.

5 Conclusion

This paper addresses the problem of duplicate elimination in storage systems in the context of unstructured peer-to-peer networks in which there is no a priori binding between an object and its location. We abstract the problem of retaining a copy of a data item to one of electing leaders in a distributed system. We show analytically, as well as using simulation and a prototype implementation in PlanetLab, that our protocols are scalable with respect to message complexity and to node resource utilization. The experimental results show that RE performs better than PQ when the number of duplicates in the network is high and the content is similar among the nodes. When the number of different objects in the network is high (nodes have unique objects), PQ performs better than RE. The reason for this behavior is that the first phase of RE is not able to prune the number of contenders and, therefore, becomes an extra overhead. To the best of our knowledge, our work is the first to address the duplicate elimination problem in unstructured networks.

Acknowledgments

This research has been partially funded by US National Science Foundation grants CNS 0551658, CCF 0444285, CNS 0334141, CCF 0325227, and STI-5011078. The first author has also been funded by CNPq and UFMS, Brazil.

REFERENCES

[1] A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer, and R.P. Wattenhofer, "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," *Proc. Symp. Operating Systems Design and Implementation (OSDI '02),* Dec. 2002.

[2] A. Awan, R. Ferreira, A. Grama, and S. Jagannathan, "Distributed Uniform Sampling in Large Real-World Networks," technical report, Dept. of Computer Sciences, Purdue Univ., Oct. 2004.

[3] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani, "Estimating Aggregates on a Peer-to-Peer Network," technical report, Computer Science Dept., Stanford Univ., 2003.

[4] W.J. Bolosky, S. Corbin, D. Goebel, and J.R. Douceur, "Single Instance Storage in Windows 2000," *Proc. Fourth USENIX Windows Systems Symp.,* Aug. 2000.

[5] S. Boyd, P. Diaconis, and L. Xiao, "Fastest Mixing Markov Chain on a Graph," *SIAM Rev., Problems and Techniques Section,* vol. 46, no. 4, pp. 667-689, Dec. 2004.

[6] L. Cox, C. Murray, and B. Noble, "Pastiche: Making Backup Cheap and Easy," *Proc. Symp. Operating Systems Design and Implementation (OSDI '02),* Dec. 2002.

[7] L. Cox and B. Noble, "Samsara: Honor among Thieves in Peer-to-Peer Storage," *Proc. ACM Symp. Operating Systems Principles (SOSP '03),* Oct. 2003.

[8] J. Douceur, A. Adya, W. Bolosky, D. Simon, and M. Theimer, "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS '02),* July 2002.

[9] J.R. Douceur and W.J. Bolosky, "A Large-Scale Study of File-System Contents," *Proc. SIGMETRICS '99,* pp. 59-70, 1999.

[10] R. Ferreira, M.K. Ramanathan, A. Awan, A. Grama, and S. Jagannathan, "Search with Probabilistic Guarantees in Unstructured Peer-to-Peer Networks," *Proc. IEEE Int'l Conf. Peer-to-Peer Computing (P2P '05),* Aug. 2005.

[11] Freenet, http://freenet.sourceforge.net/, 2007.

[12] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks," *Proc. IEEE INFOCOM '04,* Mar. 2004.

[13] Gnutella, http://gnutella.wego.com/, 2007.

[14] A.G. Goldberg and P.N. Yianilos, "Towards an Archival Intermemory," *Proc. IEEE Advances in Digital Libraries (ADL),* 1998.

[15] W. Hastings, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," *Biometrika,* vol. 57, no. 1, pp. 97-109, 1970.

[16] K. Horowitz and D. Malkhi, "Estimating Network Size from Local Information," *Information Processing Letters,* vol. 88, no. 5, pp. 237-243, Dec. 2003.

[17] P. Kulkarni, F. Douglis, J. LaVoie, and J.M. Tracey, "Redundancy Elimination within Large Collection of Files," *Proc. USENIX Ann. Technical Conf.,* pp. 59-72, June 2004.

[18] M. Lillibridge, S. Elnikety, A. Birrel, and M. Burrows, "A Cooperative Internet Backup Scheme," *Proc. Usenix Technical Conf.,* June 2003.

[19] N.A. Lynch, *Distributed Algorithms.* Morgan Kaufmann Publishers, 1996.

[20] D. Malkhi, M. Reiter, and R. Wright, "Probabilistic Quorum Systems," *Proc. 16th Ann. ACM Symp. Principles of Distributed Computing (PODC '97),* pp. 267-273, Aug. 1997.

[21] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equations of State Calculations by Fast Computing Machines," *J. Chemical Physics,* vol. 21, pp. 1087-1091, 1953.

[22] R. Motwani and P. Raghavan, *Randomized Algorithms.* Cambridge Univ. Press, 1995.

[23] PlanetLab, http://www.planet-lab.org/, 2007.

[24] C. Policroniades and I. Pratt, "Alternatives for Detecting Redundancy in Storage Systems Data," *Proc. USENIX Ann. Technical Conf.,* pp. 73-86, June 2004.

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM '01,* pp. 247-254, Aug. 2001.

[26] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. ACM SIGCOMM '01,* pp. 247-254, Aug. 2001.

[27] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam, "Taming Aggressive Replication in the Pangaea Wide-Area File System," *Proc. Symp. Operating Systems Design and Implementation (OSDI '02),* Dec. 2002.

[28] S. Saroiu, P.K. Gummadi, and S.D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. SPIE/ACM Conf. Multimedia Computing and Networking (MMCN '02),* Jan. 2002.

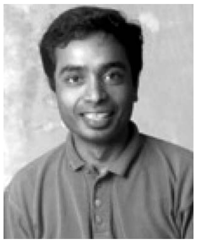
[29] N. Sarshar, P. Boykin, and V. Roychowdhury, "Percolation Search in Power-Law Networks: Making Unstructured Peer-to-Peer Networks Scalable," *Proc. Fourth IEEE Int'l Conf. Peer-to-Peer Computing (P2P '04),* Aug. 2004.

[30] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM '01,* pp. 149-160, Aug. 2001.

[31] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," Technical Report UCB/CSD-0101141, Computer Science Division, Univ. of California Berkeley, Apr. 2001.

Ronaldo A. Ferreira received the MSc degree in computer science from the University of Campinas, Brazil, and the BSc degree in computer science from the Federal University of Mato Grosso do Sul, Brazil. He is a PhD candidate in the Department of Computer Science at Purdue University. Prior to joining Purdue, he worked as a lecturer at Federal University of Mato Grosso do Sul from 1995 to 2000. His research interests are in networking and distributed systems.

Murali K. Ramanathan received the MS degree from Purdue and the BE degree from CEG, Anna University, India. He is a PhD candidate in the Department of Computer Science at Purdue University. His research interests are in program analysis and distributed systems.



Ananth Grama received the PhD degree from the University of Minnesota in 1996. He is currently a professor of computer sciences and a University Faculty Scholar at Purdue University. His research interests span the areas of parallel and distributed computing architectures, algorithms, and applications. On these topics, he has authored several papers and coauthored a text book *Introduction to Parallel Computing* with Anshul Gupta, George Karypis, and Vipin Kumar. He is a member of the American Association for Advancement of Sciences.

Suresh Jagannathan received the PhD degree from the Massachussetts Institute of Technology. He is currently an associate professor at Purdue University. Prior to joining Purdue, he was a senior research scientist at the NEC Research Laboratories, Princeton, New Jersey. His interests span programming languages and concurrent and distributed systems.