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

Ronaldo A. Ferreira Murali Krishna Ramanathan Ananth Grama Suresh Jagannathan
Department of Computer Sciences – Purdue University
West Lafayette, IN, USA
{rf, rmk, ayg, suresh}@cs.purdue.edu

Abstract

Distributed peer-to-peer storage systems rely on voluntary participation of peers to effectively manage a storage pool. Files are generally replicated in several sites to provide acceptable levels of availability. If disk space on these peers is not carefully monitored and provisioned, the system may not be able to provide availability for certain files. In particular, identification and elimination of redundant data are important problems that may arise in long-lived systems. Scalability and availability are competing goals in these networks: scalability concerns would dictate aggressive elimination of replicas, while availability considerations would argue conversely. In this paper, we provide a novel and efficient solution that addresses both these goals 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 a new randomized protocol to solve this problem in a scalable and decentralized fashion that does not compromise availability requirements of the application. Performance results using both large-scale simulations, and a prototype built on PlanetLab, demonstrate that the protocols provide high probabilistic guarantees of success, while incurring minimal administrative overheads.*

1. Introduction

Peer-to-peer storage systems have been proposed as cost-effective alternatives for providing scalable backup and archival storage [3, 10]. Peers contribute storage space in their disk to the system and in return they are allowed to backup their data at other peers. Effective storage management is a critical issue in the deployment of such systems. A particularly growing concern is how to efficiently and safely prune unwanted copies of data. Greedy peers can eagerly replicate their data at other peers and cause the collapse of the system. Even if a particular peer is restrained from eagerly replicating its data, as dictated in systems such as Samsara [4], the system as a whole must provide a mechanism for removing excess data while leaving a minimum number of replicas in the network that satisfy availability needs.

In this paper, we investigate the problem of removing duplicates in distributed peer-to-peer storage systems. We examine this issue in the context of unstructured networks [9] in which no assumption can be made about the relationship between an object and the peers in which it resides. In contrast to structured peer-to-peer networks [20, 16], unstructured networks are highly resilient to node failures and incur ver low overhead on node arrivals and departures. These characteristics make unstructured networks very attractive for users with limited resources. Unfortunately, the issue of object location, which is central to the problem of identifying redundant copies, is significantly more complex in this environment.

We assume peers in the storage system are cooperative and non-malicious. 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, each peer is responsible to keep its own data locally. The public space holds data from other peers and is subject to the elimination procedure. We see this public space as a backup space that can be periodically reused. A peer that needs to restore data from the network should use object location techniques as the ones developed in [19] and [7].

The problem we address is thus defined as follows: *Consider a peer-to-peer storage system, designed without any assumptions on the structure of the underlying network, and which contains multiple peers, each peer holding numerous files. How can a peer determine which files need to be kept using minimum communication overhead? Furthermore, how can the storage system as a whole make sure that each file is present in at least k peers, where k is a system parameter chosen to satisfy a client or application's availability requirements?*

An important issue for the effective elimination of data in a storage system is the identification of common data in different peers. Adya *et al.* [1] proposes convergent encryption to solve this problem. With convergent encryption, it is possible to safely store encrypted data in 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 it was used in [5] to identify blocks of data with the same content. Doucer *et al.* [5] 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 their approach can be easily adapted to an unstructured environment.

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 many kinds of storage environments [17, 10]. 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 want to elect a group of k leaders for each file. The elected leaders are responsible for storing the file. Our leader election protocol (specifically for this application) is different from traditional leader election algorithms in two important respects:

- The probabilistic nature of the 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 [12]. In this approach, each node creates a quorum for each one 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 is given in Section 2. In the worst case, if a replica is present at all peers, the total message complexity of the elimination algorithm 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 proceeds similar to the probabilistic quorum approach: a quorum is formed for each of the remaining nodes, and the k leaders are chosen. 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 [6]. We consider both a detailed simulation to study the scalability of the protocols, and a prototype implementation on PlanetLab [15] to quantify administrative overheads and efficiency in a realistic network environment. Our experiments measure the amount of space reclaimed, memory and communication overhead, and the average storage gain per node.

2. Duplicate Elimination Protocols

In this section, we present a detailed description of our duplicate elimination protocols. These protocols assume that all nodes are interested in eliminating duplicates and are non-malicious. Central to our protocol is a leader election mechanism in which leaders elected to preserve a file are obligated to do so, while all other participants are free to remove it.

2.1. A Probabilistic Quorum (PQ) Approach

Our first strategy is based on ideas proposed for building probabilistic quorum systems. A quorum system is defined as a set of subsets of peers, every two of which intersect. To guarantee that every two quorums intersect with high probability, the number of members in each quorum is equal to $\sqrt{n \ln n}$ [12], 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 file. Each node with the file creates a quorum with the size of the quorum defined as above; by the properties enjoyed by quorum systems, each quorum intersects the quorum of any other node that has the same file with high probability.

Each node with a file that needs to be eliminated sends a REQUEST_TO_KEEP message to all its quorum members. It decides to keep or delete the file based on the response to this message. A REQUEST_TO_KEEP message contains the following information:

- SENDER_ID: identity of the sender;
- FILE_DESCRIPTOR: content hash of the file;
- RANDOM_NUMBER: a random number locally generated by the node.

Each quorum member waits for a time period (which is 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 FILE_DESCRIPTOR. Positive responses (ACKs) are sent to the k peers that have the largest random numbers among all the messages with the same file 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 file. A peer that receives only positive responses must sort all 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 keeps the file, otherwise the file is deleted. If a file is present in all peers in the network, which corresponds to the worst case scenario, the message complexity of this protocol is $O(n\sqrt{n \ln n})$. When this protocol is applied separately for each file, the message complexity must include the number of files. In Section 2.2.2, we discuss how information about separate files can be aggregated to keep the message complexity the same as the message complexity for a single file.

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* [14] abstraction. The protocol operates in two phases. In the first phase, the number of nodes holding the same file is reduced. In the second phase, the remaining nodes decide the k leaders that should keep the file. 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 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 is one file in the entire system and each peer holds one copy of this file. Although our presentation assumes a synchronous protocol, we consider how to relax this assumption in Section 2.2.1.

A *contender* is a participating peer in the protocol that holds a copy of the file. 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 communication 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 file.

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 randomly chosen 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) with the round number.

```
▷ 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 from 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 file locally.
```

Figure 1. Contender actions.

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 Figure 1. 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 the bins that its balls land in, it is considered a winner in this round and proceeds to the next round.

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. 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 file 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 exactly the probabilistic quorum approach but with a reduced number of contenders. Details about the protocol are summarized in Figures 1 and 2.

```
▷ 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.
```

Figure 2. Mediator actions.

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.

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 jth 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 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. Due to space limitations, we omit the proofs of the algorithms. We refer the reader to [8] for the proofs and a more detailed set of experiments.

2.2.2 Aggregation

We now relax the assumption on the presence of a unique file in the entire system and explain how the protocol can be extended to deal effectively with multiple files. A simple extension would be to perform the leader election for each file in the system separately. However, this approach results in a protocol complexity of $O(n * \#$ of unique files). A better approach that reduces message complexity, and thus improves efficiency, is to hold a single election. In this case, each contender C_1 initially sends messages to mediators with a list of all the files in its local node. On a 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 $O(n\sqrt{n \ln n})$. 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 files is likely to be significantly larger than the number of nodes in the system, the latter approach is expected to be more scalable. Also, 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 approaches 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, the nodes need to pick quorum members or mediators uniformly at random. We accomplish this by performing a biased random walk using the Metropolis-Hastings (MH) algorithm [13, 2], which provides a method to hasten the mixing time of a random walk to reach a stationary uniform distribution. We have implemented the Metropolis-Hastings algorithm as part of our protocols.

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 *et al.* [11] presents an estimation 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. In our current implementation, we have the system size as a parameter. The use of an estimation scheme is part of our future work.

3. Implementation and Experimental Results

In this section, we present experimental results for the two protocols described in the previous section. Our experiments consider both a detailed large-scale simulation, as well as a prototype implementation on PlanetLab. For the PlanetLab experiments, we use file traces from real machines.

3.1. Simulation Setup and Results

Recent studies show that the topology of unstructured networks can be modeled using power-law random graphs [18]. 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 one file is the target of duplicate elimination and that the file is duplicated at a percentage β of the nodes. The file is placed at nodes selected uniformly at random from the network. We vary β from 1% to 50%. These percentages are chosen to simulate files of differing popularity, from very 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 3.1.3, 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.

3.1.1 Message Overhead

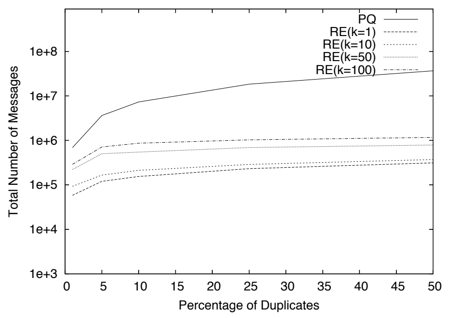

Figure 3. Total number of messages in the system vs Percentage of Duplicates.

We first investigate the scalability of the approaches with respect to the total number of messages exchanged in the system. Figure 3 shows the number of messages for the two duplicate elimination approaches applied to a file with varying duplicate percentages, and with different values of k. In the PQ approach, the parameter k has no impact in the number of messages, for all values of k the results are the same. This is expected, since the protocol consists of a single round where all peers that contain a replica send the same number of messages to the quorum members. The decision of deleting 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 in 50% of the nodes, the number of messages in the PQ approach is more than one order of magnitude greater than the worst case ($k = 100$) of the RE approach. This shows that the RE is more resilient to the number of duplicates in the network, and therefore much more scalable.

3.1.2 Load Distribution

We evaluate the load on each node in the system with respect to the number of messages handled. We study the load when the percentage of duplicates is 50% and $k = 100$, which corresponds to the highest load among the different parameters studied. Figure 4 shows the number of messages for each node. For each approach, the nodes were sorted independently based on the number of messages processed. We only compare the nodes in the figure based on its ranking 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.

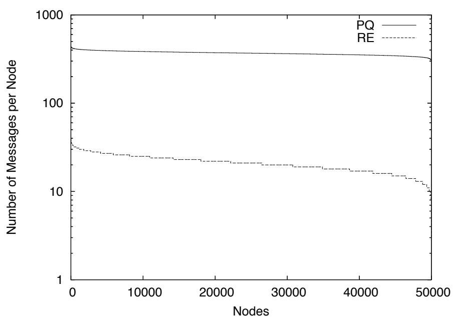

Figure 4. Number of messages received per node (Y-axis in Log Scale).

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 algorithms is able to select peers uniformly from the network. 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 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 which are picked by the nodes which eventually have to keep 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 Accuracy of the Protocols

The final goal of both protocols is to delete all duplicates while leaving k copies in the network. To evaluate how close to this goal the protocols perform, we fix the object popularity to 1% (500 copies) and vary k. In 10,000 different runs with different seeds, the protocols always matched the values of k, for k from 1 to 100.

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 a real implementation which are not captured in a simulation study.

In PlanetLab, nodes are spread around the world, and are connected to the Internet with varying bandwidths. The message delays are what a large scale application would encounter in a realistic scenario. Nodes in PlanetLab consist of PCs with Pentium IV processors with at least 512MB of RAM and run Linux as their operating system [15].

In our experiment, we use a separate application (command center) to control data collection and initiation of the nodes. The command center informs the nodes which file system they should use and how they should connect to each other. The number of connections of each node is generated using a zipf-like distribution.

3.2.1 File Traces

The file traces we use in our experiments are data sets obtained from [6]. These traces correspond to $10,568$ file systems from 4,801 Windows machines in a commercial environment. The trace contains 140 million files totaling 10.5 TB of data. For our experiments in PlanetLab, we were able to consistently acquire 132 nodes to run our application; we use this number for the benchmarks shown below. We randomly choose 132 file systems from the trace and assign one file system to each PlanetLab node. 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. Even though we could run multiple processes per machine and build a larger overlay network, we decided not do it to avoid the introduction of artificial delays between processes running in the same machine.

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 relationships 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 given in Table 1.

3.2.2 Experimental Results

Since the overheads in the PQ protocol are much higher than the RE and because our simulation results indicate that RE exhibits better scalability characteristics, our experiments in PlanetLab are conducted using only the RE protocol.

Our experiments examine how 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

Maximum number of files in a FS	31,533
Minimum number of files in a FS	2
Average number of files per FS	6,719.15
Maximum FS size	4,678MB
Minimum FS size	100KB
Average FS size	653MB
Total space used	86,232MB
Unique files in the system	420,487

Table 1. System parameters.

is varied from 10KB to 10MB. The following parameters are investigated during the experiment:

- **Total Space Reclaimed:** This gives 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 gives the total number of messages including REQUEST_TO_KEEP, ACK, and NAK 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.

Figure 5 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 approaches for varying minimum file sizes from 10KB to 10MB. Only files greater than the established minimum file size are considered for duplicate elimination. After the 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 increase in the minimum file size, the final storage size deviates from the optimum storage. When the minimum file size is 10KB, the final storage size is close to the optimum, which is around 50% 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 axis of the figure. With increase in the minimum file size, the number of messages

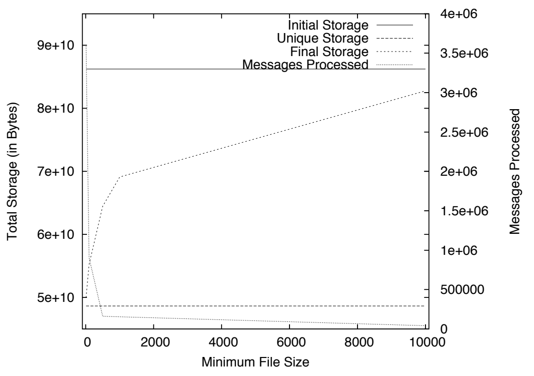

Figure 5. The X-axis represents the minimum file size for which the duplicate elimination was performed, Left axis represents the system storage size, Right axis represents the total number of messages processed in the system. Initial and final storage system sizes along with the unique storage size is also given.

exchanged in the system reduces. The presence of the messages processed along with the final storage space gives an approximate estimate on the minimum file size that needs to be used. For example, when the minimum file size is set at 100KB, 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.

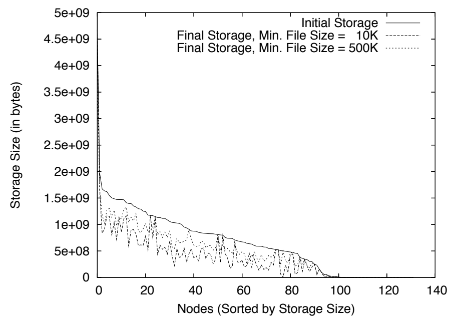

Figure 6. Storage size per node for varying minimum file sizes. Nodes are sorted based on the initial file size.

We next study the amount of storage reclaimed per node. Figure 6 presents the initial storage size for each node in the system. The results for minimum file size 10K and 500K are shown in the Figure. We use only these two for clarity in the presentation. The results for other minimum file sizes show 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 10K is greater than when it is 500K. This can be explained due to the elimination of more smaller- sized files.

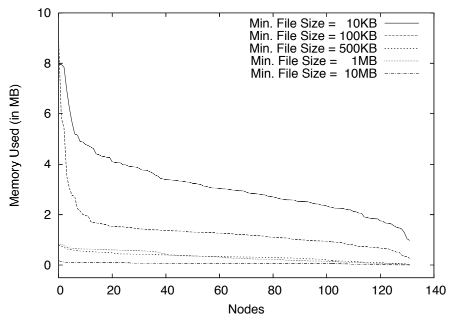

Figure 7. Memory overhead (in MB) per node for varying minimum file sizes. Nodes are sorted based on the memory overhead.

The amount of memory consumed per node as a mediator is shown in Figure 7. 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 10K, the average amount of memory consumed across all nodes is approximately 5MB. Though most of the nodes have equal memory consumption, there are a few nodes that have more memory overhead. This is similar to the observations made in the simulation results. The mediators which are part of the mediator sets for the nodes that eventually keep the file receive more messages than the rest. The same observations make in Section 3.1.2 about our implementation apply in this case. Observe, though, that the deviation from the average is not significant. The memory consumed is smaller with increase in the minimum file size, which is due to fewer number of files selected for duplicate elimination.

4. Related Work

Douecer *et al.* [5] addresses the duplicate elimination problem for a wide-area storage system. In their paper, they propose 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. The records are sorted into buckets and the buckets are assigned to a cell. By building this data structure, an underlying structure is formed and has to be maintained. This is similar to the distributed indexing approach present in many popular structured peer-to-peer networks [20, 16]. We address the more general problem of trying to eliminate duplicates without making any assumptions on the structure of the overlay network.

Pastiche [3] 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 [3] and a few solutions were proposed. The solutions initially proposed were based on equivalence classes, solution of cryptographic puzzles, and some form of electronic currency. All solutions were considered complicated and with significant overhead. The authors revisit this problem in [4] and propose Samsara, a fairness enforcement mechanism. In this scheme, each peer that requests storage space of another peer must agree to reserve some space in its own disk to the peer it is requesting the storage. For the problem of greedy peers, the scheme performs well. However, there is no solution to the problem of common data that are highly replicated by different peers in the network.

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 file to one of electing leaders in a distributed system. We show using both simulation and a prototype implementation in PlanetLab that the protocols are scalable with respect to message complexity and to node resource utilization. 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 the National Science Foundation grant CNS 0509387. The first author has been partially 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. In *Proceedings of OSDI'02*, Dec 2002.

[2] A. Awan, R. Ferreira, A. Grama, and S. Jagannathan. Distributed Uniform Sampling in Large Real-World Networks. Technical Report, Department of Computer Sciences, Purdue University, October 2004.

[3] L. Cox, C. Murray, and B. Noble. Pastiche: Making Backup Cheap and Easy. In *Proceedings of OSDI '02*, Boston, MA, December 2002.

[4] L. Cox and B. Noble. Samsara: Honor Among Thieves in Peer-to-Peer Storage. In *Proceedings of SOSP 2003*, Bolton Landing, NY, October 2003.

[5] J. Douceur, A. Adya, W. Bolosky, D. Simon, and M. Theimer. Reclaiming Space from Duplicate Files in a Serverless Distributed File System. In *Proceedings of ICDCS'02*, Vienna, Austria, July, 2002.

[6] J. Douceur and W. Bolosky. A Large-Scale Study of File-System Contents. In *Proceedings of SIGMETRICS'99*, pages 59–70, 1999.

[7] R. Ferreira, M. K. Ramanathan, A. Awan, A. Grama, and S. Jagannathan. Search with Probabilistic Guarantees in Unstructured Peer-to-Peer Networks. In *Proceedings of IEEE P2P'05*, Konstanz, Germany, August 2005.

[8] R. Ferreira, M. K. Ramanathan, A. Grama, and S. Jagannathan. Randomized Protocols for Duplicate Elimination in Peer-to-Peer Storage Systems. Technical Report, Department of Computer Sciences, Purdue University, 2005.

[9] Gnutella. http://gnutella.wego.com/.

[10] A. Goldberg and P. Yianilos. Towards an Archival Intermemory. In *Proceedings of IEEE Advances in Digital Libraries (ADL)*, 1998.

[11] K. Horowitz and D. Malkhi. Estimating Network Size from Local Information. *Information Processing Letters*, 88(5):237–243, December 2003.

[12] D. Malkhi, M. Reiter, and R. Wright. Probabilistic Quorum Systems. In *Proceedings of PODC '97*, pages 267–273, Santa Barbara, CA, August 1997.

[13] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[14] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[15] PlanetLab. http://www.planet-lab.org/.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM '01*, pages 247–254, San Diego, CA, August 2001.

[17] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming Aggressive Replication in the Pangaea Wide-Area File System. In *Proccedings of OSDI '02*, Boston, MA, December 2002.

[18] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of MMCN '02*, San Jose, CA, USA, January 2002.

[19] N. Sarshar, P. Boykin, and V. Roychowdhury. Percolation Search in Power-Law Networks: Making Unstructured Peer-to-Peer Networks Scalable. In *Proceedings of IEEE P2P'04*, Zurich, Switzerland, August 2004.

[20] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM '01*, pages 149–160, San Diego, CA, August 2001.