This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE P2P 2010 proceedings.

PeerDedupe: Insights into the Peer-assisted Sampling Deduplication

Yuanjian Xing, Zhenhua Li, and Yafei Dai
Department of Computer Science and Technology
Peking University, China
{xyj, lzh, dyf}@net.pku.edu.cn

***Abstract*—As the digital data rapidly inflates to a world-wide storage crisis, data deduplication is showing its increasingly prominent function in data storage. Driven by the problems behind the mainstream server-side deduplication schemes, recently there has been a tendency of introducing peer-assisted methods into the deduplication systems. However, this topic is still quite vague at present and lacks thorough research.**

In this paper, we conduct in-depth and quantitative investigation on the peer-assisted deduplication. Through measurements we observe that the inter-peer duplication accounts for a large proportion of the total duplication, and exhibits strong peer locality. Then based on our observations, we propose PeerDedupe, a novel peer-assisted sampling deduplication approach. Experiments show that PeerDedupe can remove over 98% duplication with each peer coordinating with no more than 5 other peers, and it requires much less server RAM usage than the existing works.

I. Introduction

As the digital data rapidly inflates to a world-wide storage crisis, data deduplication has doubtlessly become one of the hottest research topics in data storage [1], [2]. The storage service providers like EMC, Symantec and HP have been adding deduplication technologies to their products in order to save storage capacity, operation cost and energy consumption. For its fine granularity and resilience, *chunk-level deduplication*, which splits a backup data stream into chunks of fixed or variable length and then removes duplicate chunks meanwhile or afterwards, has now become the mainstream deduplication technique. Thus in this paper, we focus on the chunk-level deduplication.

The current mainstream deduplication schemes are mainly deployed at the server side (e.g., the Data Domain File System [3] and HP's Sparse Indexing [4]). Users send their full backup data to the server, and then the duplicate chunks are eliminated by the server. Though this server-side deduplication scheme can easily remove duplicate chunks system widely, it faces bundles of tough problems at scale, such as the *huge bandwidth consumption* and the famous *disk bottleneck problem* [3], [4]. The disk bottleneck problem means at scale, it is impractical to keep a full index of system-wide chunk IDs in memory, while traditional disk-based index is far too slow because of frequent disk access.

As today's numerous PCs are getting more and more powerful, the peer-assisted deduplication scheme begins to catch people's eyes. For example in EMC's Avamar system [5], when a user wants to back up data, he first splits his data into chunks and removes the ones that he has previously stored. Then the *ID*s of the remaining chunks are sent to the server for further global deduplication. Finally, the user only backs up the globally new chunks. The above peer-assisted deduplication approach significantly reduces the bandwidth consumption. However, for performing global deduplication, the server still faces the disk bottleneck problem, which usually requires the server to be high-performance and thus expensive.

Generally, there has been a tendency of introducing peer-assisted methods into the deduplication systems. However, this topic is quite vague at present and lacks in-depth research. The confusions may include: is the process of Avamar's further global deduplication (also called the inter-peer deduplication) necessary, or does a peer already achieve good enough deduplication quality by just removing the chunks that he previously backed up (called the intra-peer deduplication)? If the inter-peer deduplication is necessary, how can we effectively eliminate the inter-peer duplication without requiring an expensive high-performance server?

On the above-mentioned problems, we conduct in-depth investigation by measuring three different types of real-world traces. We gain two important observations: 1) *the inter-peer duplication accounts for a nonnegligible (in fact large) proportion of the total duplication*; 2) *the inter-peer duplication exhibits strong peer locality (i.e., most of a peer's duplicate data is actually covered by only a small number of other peers), which can be well fit by the Weibull distribution*. In fact, the strong peer locality results from the phenomenon that a peer usually interacts with only a limited number of others in reality, while the duplicate data is just generated in the interactions. For example, an engineer's source code is often shared with other members participating in the same project; a researcher's duplicate data can be largely covered by only a few members in the same academic community; most of a user's emails are exchanged among a limited number of people.

Motivated by these observations, we propose PeerDedupe, a novel peer-assisted *sampling* deduplication approach. Distinguishing from EMC's Avamar, in PeerDedupe, a peer sends the *ID*s of his locally new chunks to only a small number of the *most valuable helpers* (MVHs) for further inter-peer deduplication. For the key problem of selecting MVHs, we model it as the *maximum coverage problem*, which is in fact NP-hard. Although the classical greedy algorithm [6] provides

978-1-4244-7141-6/10/$26.00 ©2010 IEEE

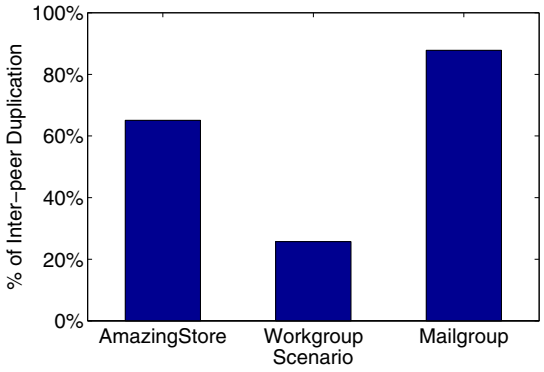


Fig. 1. The proportion of inter-peer duplication in total duplication.

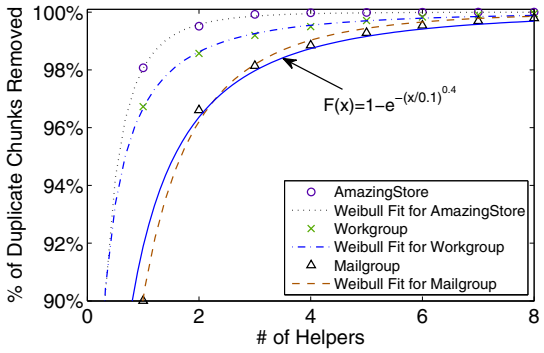


Fig. 2. The cumulative distribution of the inter-peer duplication at other peers.

an approximation solution with a polynomial time complexity, it is still too slow for our practical use, making the MVH selection process a bottleneck. Thus we design a *fast* and *effective* probabilistic estimation algorithm, which relies on the MinHash sampling method [7] to estimate the similarity of two sets. We develop a rule of setting sampling size for the MinHash method, which achieves a target estimation accuracy with the smallest possible sampling size. Experiments on one large-scale synthetic data set and two real-world traces show that PeerDedupe can remove over 98% inter-peer duplication with each peer coordinating with no more than 5 MVHs. Besides, PeerDedupe requires much less server RAM usage than the existing works.

The rest of this paper is organized as follows. Section II shows our measurement observations and explains the PeerDedupe work schema. Section III describes the detailed techniques for MVH selection. Section IV shows our experimental results. Section V reviews the related work. Finally we conclude this paper and point out the future work in Section VI.

II. Measurement Observations and PeerDedupe Approach

A. Measurement Observations

We conducted in-depth measurements, concerning the benefits of the inter-peer deduplication and the degrees of peer locality in the real world, which provide the primary support

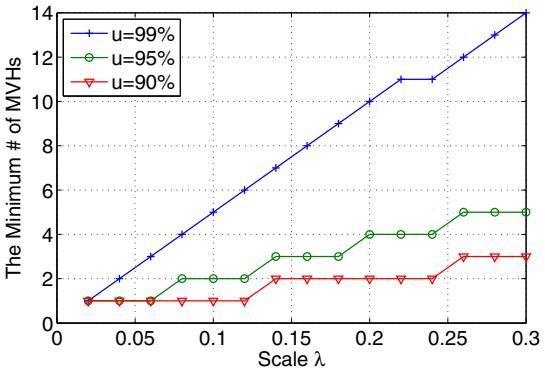


Fig. 3. The least number of required MVHs k^* to achieve different inter-peer deduplication qualities u for the Weibull distribution $F(x;\lambda,\alpha)$ where $\alpha = 0.4$ and λ varies.

for our PeerDedupe approach. We report our measurement results on three *different types* of traces [1]. The first one is a snapshot of 176 users' backup data of AmazingStore [8], [9], a real peer-to-peer storage system deployed by us, which represents a general backup circumstance. The second one, called *Workgroup*, is the backup data of a group of 19 researchers, which represents a small-scale academic group. The third one, called *Mailgroup*, is the email data of 113 users in our research institution, which represents the exchange community. For the three traces, users' data was split into variable-length chunks using Rabin fingerprint [10], ranging from 1KB to 128KB with 8KB on average. Section IV.A describes the detailed characteristics of Workgroup and Mailgroup.

The inter-peer duplication is calculated as follows: for each peer, we first remove his locally duplicate chunks. After that, if a peer's chunk still occurs at other peers, it is identified as an inter-peer duplicate chunk. Fig. 1 depicts the proportion of the inter-peer duplication in total duplication. We see that the inter-peer duplication constitutes a large proportion for all the three traces, which indicates that *only the intra-peer deduplication is far from enough. A storage system will benefit much more if the inter-peer deduplication is introduced.*

We further take a look at the distribution of the inter-peer duplication among peers. For each peer, we first accurately identify the inter-peer duplication globally by brute force, and then the classical greedy algorithm [6] is used to measure the ratio of its inter-peer duplication covered by other peers [2]. Fig. 2 shows the cumulative distribution at the first 8 helpers. We notice that:

1) The distribution of the inter-peer duplication presents a strong peer locality. For the measured three traces, *two helpers are sufficient to cover more than 96% inter-peer duplication.*
2) The Weibull distribution $F(x;\lambda,\alpha) = 1 - \mathrm{e}^{-(x/\lambda)^{\alpha}}$ can well fit the cumulative distribution of the inter-peer du-

[1]The three traces are now available at http://en.amazingstore.org/xyj/.

[2]In fact, the problem of precisely measuring the cumulative distribution is NP-hard. The classical greedy algorithm we used shows a *tight* lower bound of this NP-hard problem and is now widely adopted by others. For the detail, refer to Section III.A.

plication. The (λ, α) pairs for AmazingStore, Workgroup and Mailgroup are (0.0531, 0.4676), (0.0247, 0.3312) and (0.1872, 0.5009). We also examined other distributions, including the exponential and Pareto distributions. None of them play as well as the Weibull distribution.

The Weibull distribution $F(x; \lambda, \alpha)$ is determined by the parameter pair (λ, α), which we define as the *peer locality indicator*. $\lambda > 0$ is the scale parameter. A smaller λ shows higher skewness in the duplication distribution and thus stronger peer locality. $\alpha > 0$ is the shape parameter. When $\alpha = 1$, the Weibull distribution is actually the exponential distribution. For the measured three traces, we find $\alpha < 1$, which indicates a heavier tail than the exponential distribution. Besides, the α values for the three traces do not differ much, which means *the shapes of their Weibull distributions are similar and the peer locality is mainly determined by the scale parameter* λ.

With the inter-peer duplication following Weibull distribution $F(x; \lambda, \alpha) = 1 - \mathrm{e}^{-(x/\lambda)^{\alpha}}$, the least number of required helpers k^* to achieve the inter-peer deduplication quality u (i.e., removing u ratio of the inter-peer duplication) is

$$k^* = \left\lceil F^{-1}(u; \lambda, \alpha) \right\rceil = \left\lceil \lambda \left(-\ln\left(1-u\right)\right)^{\frac{1}{\alpha}} \right\rceil \qquad (1)$$

Fixing $\alpha = 0.4$ (a roughly intermediate value for the measured three traces), we calculate k^* under different λ values to achieve a given inter-peer deduplication quality u. As Fig. 3 shows, more helpers will be needed for high inter-peer duplication quality requirement (large u) and poor peer locality (large λ). For a specific Weibull distribution $F(x; 0.1, 0.4)$, 5 helpers can cover more than 99% inter-peer duplication. Note that $F(x; 0.1, 0.4)$ presents an intermediate peer locality (intermediate λ and α values) among our three measured traces as Fig. 1 shows. We will use $F(x; 0.1, 0.4)$ as an example in our following analysis and experiments.

In summary, we observe that 1) the inter-peer duplication accounts for a large proportion of the total duplication, and 2) for the strong peer locality, a peer can detect most of his inter-peer duplication at only a few MVHs. So, the key problem is how to select MVHs. Before going into the details of MVH selection (in Section III), we first describe the work schema of our proposed PeerDedupe, which is based on the above measurement observations.

B. PeerDedupe Approach

Fig. 4 depicts the work schema of PeerDedupe. The initiator, who starts a backup operation, first performs intra-peer deduplication by removing the chunks that he previously backed up (see step 1). After step 1, a set of his locally new chunks is got, which we call *local increments*. Then the initiator sends a *sample* of the local increments' IDs to the server, asking for a fixed number of MVHs (see step 2). For each peer, the server keeps an index of only a small number of sampling chunks in memory, based on which the server selects MVHs for the initiator (see step 3). After knowing the MVHs (see step 4), the initiator sends the IDs of his full local increments to each of them for the inter-peer deduplication (see step 5). On receiving the deduplication results (the knowledge of which chunks have been previously stored and their access addresses) from the MVHs (see step 6), the initiator sends the really "new" chunks[3] to the remote secondary storage (see step 7).

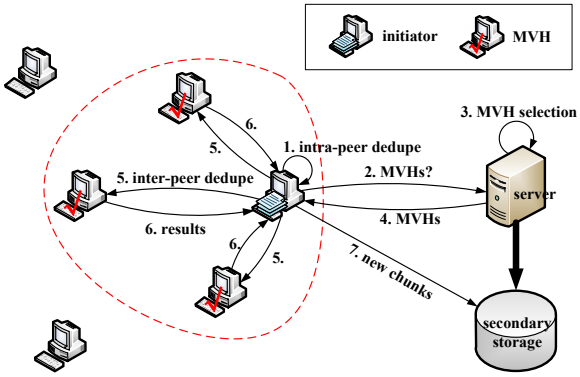


Fig. 4. PeerDedupe work schema. Before the deduplication starts, the initiator already gets the chunk IDs by chunking and hashing.

In PeerDedupe, each peer maintains an index of his stored chunks locally (the server will also maintain on disk a copy of the index in case the peer's local index is damaged or lost), and thus the deduplication is actually searching chunk IDs in the index. The process of intra-peer deduplication is straightforward and can easily remove all the locally duplicate chunks. We now describe the MVH selection, the key problem in the inter-peer deduplication.

III. MVH Selection Techniques

In this section, we describe our techniques of selecting MVHs for the inter-peer deduplication. We first model the process of MVH selection. Then we propose our fast probabilistic estimation algorithm for MVH selection which relies on sampling technique to estimate two sets' similarity. Finally we discuss on the sampling method and size in detail.

A. Modeling the MVH Selection

Assume there are n peers in the system. Let C_0 be the chunk set that the initiator wants to deduplicate in the inter-peer deduplication process and let C_i $(1 \le i \le n-1)$ be the chunk set that the i-th peer maintains locally. Given a fixed number k $(1 \le k \le n-1)$ and a collection of sets $C = \{C_1, C_2, \ldots, C_{n-1}\}$, the objective of the MVH selection is to find a subset $C' \subseteq C$, such that $|C'| = k$ and the number of covered chunks

$$\left| C_0 \cap \left(\bigcup_{C_i \in C'} C_i \right) \right|$$

is maximized.

[3]Some of the transferred chunks may still already exist in the system. They are those not covered by the selected MVHs.

Let $X_i = C_0 \cap C_i$ and $X = \{X_1, X_2, \ldots, X_{n-1}\}$. The objective is equivalent to

$$\begin{array}{ll} \text{maximize} & \left| \bigcup_{X_i \in X'} X_i \right| \\ \text{subject to} & X' \subseteq X \\ & |X'| = k \end{array}$$

which is the *maximum coverage problem* (MCP), a NP-hard problem [6]. The classical greedy algorithm which selects k sets by iteratively picking out the set that covers the maximum number of currently uncovered elements has a *tight* $(1-\frac{1}{e})$-approximation lower bound [6], [11]. But for the costly operations on the original large sets (mainly the intersection and union operations on large sets, and the frequent disk accesses due to the disk bottleneck problem [3], [4]), it is still too slow for practical use, making the MVH selection process a bottleneck.

B. Our Probabilistic Estimation Algorithm

We develop the classical greedy algorithm, which is deterministic, into a probabilistic estimation algorithm to speed up the MVH selection process. Our probabilistic estimation algorithm relies on sampling technique to firstly estimate the similarities between the initiator's chunk set and other peers' chunk sets, and then MVHs are selected from the "similar" peers. For the definition of similarity, we adopt the classical definition which is given by [12]: the similarity of sets A and B, $r(A, B)$, is $\frac{|A \cap B|}{|A \cup B|}$.

Given n peers $P = \{P_1, P_2, \ldots, P_n\}$ with their samples $S = \{S_1, S_2, \ldots, S_n\}$ and the sample of the initiator's chunks to be deduplicated S_0, our method of selecting k (k is a system configured value) MVHs is performed in k rounds. In each round, we choose the peer P_i from P whose sample S_i is most similar to S_0. Then we update P with $P - \{P_i\}$ and S_0 with $S_0 - S_i$. The process continues until k MVHs are all found. If more than one peers have the highest estimated similarity with the initiator in one round, we randomly pick one as the MVH.

Obviously, by generating a small number of sampling chunks on the original large chunk set, we dramatically speed up the MVH selection. We now discuss our sampling method and the rule of setting sampling size, which enable us to achieve desirable estimation accuracy compared to the classical greedy algorithm.

C. Sampling Method

There are varieties of sampling methods [13], but few of them are fit for our deduplication circumstance. In our deduplication circumstance, backup data is split into very fine granularity, so the number of chunks each peer stores is huge. However, the number of the sampling chunks should be small enough to reduce the server RAM usage and make the process of MVH selection run fast. Therefore, *an ideal sampling method should have strong ability of detecting shared chunks in small-size sampling sets.*

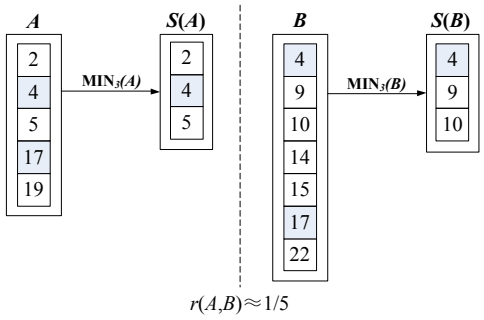


Fig. 5. Broder's MinHash to unbiasedly estimate the similarity of A and B.

Simple random sampling (SRS), which chooses a few elements from a given set uniformly at random, is a straightforward sampling method and was used in Pastiche [14] to detect *highly* similar peers (in term of their common data). However, with a small sampling size, SRS often misses shared elements leading to underestimation of sets' similarity. Consider two *identical* sets A and B, each with n elements. From each set we independently pick out only one element using SRS. Now we have two sample sets $S(A)$ and $S(B)$. We can see that though the similarity of set A and B, $r(A, B)$, is in fact 1, SRS's estimation is only $r(S(A), S(B)) = \frac{1}{n}$.

Broder proposed MinHash sampling [7], which has now been widely used in detecting similar objects [15], [16], [17] in large data repository. Fig. 5 shows a simplified example of estimating the similarity of set A and B with MinHash. First, every element is assigned with an ID, which meets the min-wise independent permutation requirement [18]. Then from each set, the smallest s elements (in the lexicographic order of their IDs) are picked out as the sample, where s is constant. Define $\text{MIN}_s(W)$ as the set of the smallest s chunks in set W. Let $S(A) = \text{MIN}_s(A)$ and $S(B) = \text{MIN}_s(B)$, then it is concluded that the value

$$\frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

is an *unbiased estimation* of the similarity of A and B [7].

For the strong ability of detecting shared elements, we adopt MinHash in PeerDedupe to generate the same number of sampling chunks for each peer. We now describe the key process of setting sampling size for MinHash in our deduplication circumstance.

D. Sampling Size

Generally, the larger the sampling size is, the more accurate the similarity estimation will be, but the more overhead the server pays. Thus, the sampling size should balance the estimation accuracy with the server overhead. Our rule of setting sampling size is *to make MinHash achieve the target estimation accuracy with the smallest possible sampling size*.

Assume each peer gets k (system configured) MVHs for the inter-peer deduplication. Let $F(k)$ be the ratio of the inter-peer duplication that is covered by the first k MVHs selected by the classical greedy algorithm . Let $G(k; r, s)$ be

the ratio of the inter-peer duplication that is covered by the k MVHs selected by our probabilistic estimation algorithm, cover. Thereinto, r is the *inter-peer redundancy degree* (the ratio of a peer's chunks which also occur at other peers), and s is the number of sampling chunks for each peer. We define the estimation accuracy μ as the ratio of $G(k;r,s)$ to $F(k)$ rather the maximum coverage ratio, because 1) solving the maximum coverage ratio is NP-hard (see Section III.A), 2) the classical greedy algorithm shows a *tight* lower bound of this NP-hard problem and is now widely adopted by others, and 3) our probabilistic estimation algorithm is in fact developed from the classical greedy algorithm (see Section III.B).

Because of the error of the similarity estimation, usually $G(k;r,s) \leq F(k)$ (the equality holds when the error of the estimation is small enough to have no impact on MVH selection result). Thus, the objective is formulated to find the smallest value s^* which satisfies $G(k;r,s^*) \geq \mu F(k)$ for a target estimation accuracy μ $(0 < \mu < 1)$.

We define a selected MVH or a MVH selection is *valid* iff at least one shared chunk is detected in a round of the similarity estimation. Otherwise, it is defined as an *invalid* MVH or an *invalid* MVH selection. The invalid MVH has an estimated similarity of 0 with the initiator and is selected randomly from the rest of the available peers. When the number of peers in the system is large, the expected duplication coverage of the invalid MVH is small enough to be ignored.

Let $f(i) = F(i) - F(i-1)$ $i \leq k$ be the deduplication contribution of the i-th MVH (i.e., the duplicate chunks covered by the i-th MVH minus those covered by the first $i-1$ MVHs), selected by the classical greedy algorithm, and $p(i;r,s)$ be the probability of that MVH being valid in our estimation with r inter-peer redundancy degree and s sampling chunks for each peer. Usually, if the i-th MVH is valid, it will be selected as a helper in our probabilistic estimation algorithm. This claim is practically reasonable for the high skewness of the duplication distribution as shown in the three real-world traces (see Fig. 2). So,

$$G(k;r,s) \geq \sum_{i=1}^{k} f(i)p(i;r,s) \tag{2}$$

Theorem 1. *Given an initiator and a peer, each has N unique chunks among which n chunks are in common. Select s chunks from each one using MinHash. The probability of finding at least one shared chunk is $p(s) \geq \left(1-\left(1-\frac{n}{N}\right)^s\right)^2$.*

Proof: Consider two chunk sets A and B where $|A| = |B| = N$ and $|A \cap B| = n$. Using MinHash, we denote the sets of their smallest s chunks (in the lexicographic order of their IDs) as $S(A)$ and $S(B)$. Note that if both $S(A)$ and $S(B)$ contain at least one shared chunk of A and B, that chunk is guaranteed to be the same one [4]. Let $S'(A) = S(A) \cap (A \cap B)$, $S'(B) = S(B) \cap (A \cap B)$. Thus the probability of detecting at least one shared chunk in $S(A)$ and $S(B)$ is

$$\begin{aligned} p(s) &= P\left(S(A) \cap S(B) \neq \emptyset\right) \\ &= P\left(S'(A) \neq \emptyset\right) \cdot P\left(S'(B) \neq \emptyset\right) \end{aligned}$$

Thereinto,

$$\begin{aligned} &P(S'(A) \neq \emptyset) = 1 - P(S'(A) = \emptyset) = 1 - \frac{\mathrm{C}_{N-n}^{s}}{\mathrm{C}_{N}^{s}} \\ &= 1 - \frac{(N-n)!}{s!(N-n-s)!} \cdot \frac{s!(N-s)!}{N!} \\ &= 1 - \frac{(N-n)(N-n-1)\cdots(N-n-s+1)}{N(N-1)\cdots(N-s+1)} \\ &\geq 1 - \left(1-\frac{n}{N}\right)^s \end{aligned}$$

Equally, $P(S'(B) \neq \emptyset) \geq 1 - \left(1-\frac{n}{N}\right)^s$. Therefore,

$$p(s) \geq \left(1-\left(1-\frac{n}{N}\right)^s\right)^2 \text{[5]} \tag{3}$$

■

For simplicity, assume each peer has an equal size of chunk set. From Equation 2 and Equation 3, we easily get

$$G(k;r,s) \geq \sum_{i=1}^{k} f(i)\left(1-(1-rf(i))^s\right)^2$$

Let $\widehat{G}(k;r,s) = \sum_{i=1}^{k} f(i)\left(1-(1-rf(i))^s\right)^2$, where $f(i) = F(i) - F(i-1)$. Thus a conservative setting of sampling size $\widehat{s}^*$ is

$$\widehat{s}^* = \arg\min_{s \in \mathbb{N}} \epsilon(\mu,k,r,s)$$

where

$$\begin{aligned} &\epsilon(\mu,k,r,s) > 0 \\ &\epsilon(\mu,k,r,s) = \widehat{G}(k;r,s) - \mu F(k) \end{aligned} \tag{4}$$

$\widehat{s}^*$ can be easily calculated with the bisection method in logarithmic time. We see that with fixed number of MVHs k and the specific cumulative distribution of the inter-peer duplication $F(x)$, $\widehat{s}^*$ increases with the target estimation accuracy μ and decreases with the inter-peer redundancy degree r.

When applying our rule of setting sampling size, the number of MVHs k is system configured, and the duplication distribution $F(x)$ and the redundancy degree r can be roughly estimated from users' previous backups. Let $k = 5$ and $F(x)$ follow Weibull distribution with $(\lambda,\alpha) = (0.1, 0.4)$, i.e., $F(x) = 1 - \mathrm{e}^{-(x/0.1)^{0.4}}$. To achieve a given target estimation accuracy μ, we plot the value of $\widehat{s}^*$ with varying redundancy degree r. As Fig. 6 shows, less sampling chunks are needed when the inter-peer redundancy degree gets higher or the target estimation accuracy gets lower. For the redundancy degree of 0.5, 300 sampling chunks can achieve 99% estimation accuracy and 50 sampling chunks can achieve 95% estimation accuracy.

[4]This is unlike randomly selecting s chunks, in which even if both two sample sets contain a shared chunk of their original sets, it may not be the same one.

[5]Himabindu *et al.* [19] got a similar result of the detection probability, but their assumptions and formula deduction are different from ours.

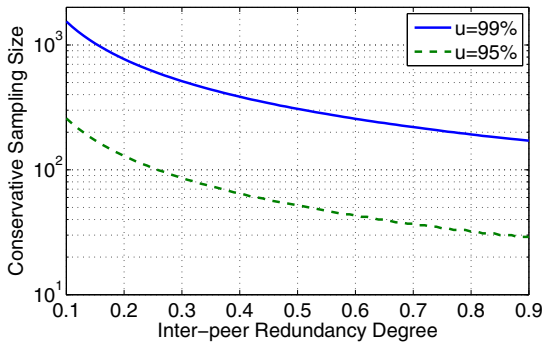


Fig. 6. The conservative sampling size $\widehat{s}^*$ to achieve a target estimation accuracy μ where the peer locality $(\lambda, \alpha) = (0.1, 0.4)$. The inter-peer redundancy degree r varies.

IV. Performance Evaluation

A. Experiment Setup

We used one large-scale synthetic data set and two real-world traces, to verify the feasibility and effectiveness of PeerDedupe. The first data set, called *Simugroup*, is a collection of generated backup data of $100,000$ peers. Each peer possesses 8GB locally unique data, which is split into on average 8KB chunks, and thus has 1M data chunks. In total, Simugroup contains about 800TB user data, which can well simulate a large or medium-sized data center at present. The cumulative distribution of the inter-peer duplication follows Weibull distribution and we chose a reasonable peer locality $(\lambda, \alpha) = (0.1, 0.4)$ for the Weibull distribution (see Section II.A). Three scenarios of different inter-peer redundancy degrees, $r = 0.1, 0.5, 0.9$, are examined for Simugroup.

The two real-world traces are called *Workgroup* and *Mailgroup* [6], whose key statistics are given in Table I. Workgroup consists of full backups of 19 researchers' PCs, lasting 4 weeks. Since machines are only powered up during work days, altogether there are 380 full backups in this 2.51TB data set. On average, each peer backed up 6.77GB data every day with 85.86MB local increments (new to himself) and 49.95MB global increments (new to the whole system). Mailgroup consists of full backups of 113 email users' inboxes, lasting 38 days. Altogether there are 4294 full backups in this 569.81GB data set. On average, each peer backed up 135.88MB email data every day with 0.90MB local increments and 0.35MB global increments.

The inter-peer redundancy degrees of Workgroup and Mailgroup differ significantly: 24.5% and 67.0%. Our experiments will show that using MinHash and our rule of setting sampling size, we can achieve desirable inter-peer deduplication quality with PeerDedupe for both traces. The experiments on Workgroup and Mailgroup started with an empty archive, took peers' ⟨chunk ID, chunk length⟩ pairs as input and deduplicated them in the order of their respective backup time. We chose SHA-1 to compute a chunk's ID.

[6]Workgroup and Mailgroup present relatively poorer peer locality than AmazingStore, and thus are more challenging for PeerDedupe.

[7]Backup files were split into variable-length chunks using Rabin fingerprint [10], ranging from 1KB to 128KB with 8KB on average. In our trace, the average chunk size is slightly smaller because there are many small files.

TABLE I
Key Statistics of Workgroup and Mailgroup Traces

	Workgroup	***Mailgroup***
No. of users	19	113
time span (day)	20	38
data set size (TB)	2.51	0.57
avg. daily backup per user (GB)	6.77	0.14
avg. chunk size (KB)[7]	6.45	7.29
daily local increments per user (MB)	85.86	0.90
daily global increments per user (MB)	49.95	0.35

B. Metrics

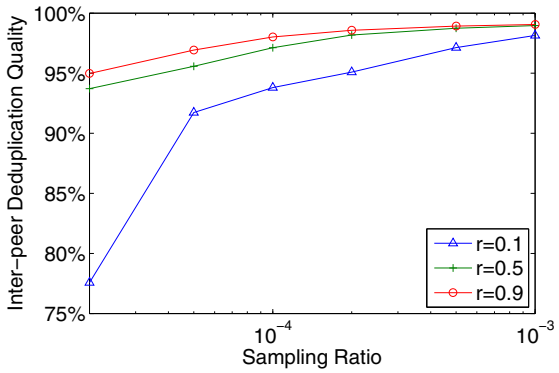
1) *Deduplication quality*, measured by the percentage of duplicate chunks removed, is our most concerned property. We evaluate the effectiveness of the MinHash sampling method and the rule of setting sampling size, and the overall performance of the duplication elimination.
2) *Server RAM reduction* shows how much RAM that the server can save. The mainstream deduplication methods relies on the server to perform global deduplication. However, they need too much server RAM to hold chunk IDs, which is their performance bottleneck.
3) *Peer-side overheads* include peer RAM usage, CPU time and communication cost. They are critical for PeerDedupe's practicality.

C. Deduplication Quality

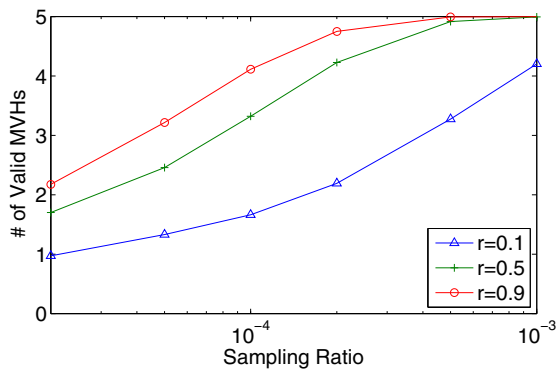
1) Simulation Results: For simugroup, we let each peer coordinate with 5 MVHs, making it possible to eliminate 99% inter-peer duplication (see Fig. 2 shows). Fig. 7 depicts the achieved inter-peer deduplication quality and the number of valid MVHs by applying our probabilistic estimation algorithm. As can be seen, with a larger sampling ratio or a higher redundancy degree, MinHash is more likely to identify shared chunks, and thus our probabilistic estimation algorithm gets more valid MVHs and removes more inter-peer duplication. With a sampling ratio of 0.02%, 95.1% inter-peer duplication could be removed for 0.1 redundancy degree, and more than 98% inter-peer duplication could be removed for 0.5 (or higher) redundancy degree.

Fig. 8 shows the effectiveness of our rule of setting sampling size with 5 MVHs. For the 3 inter-peer redundancy degrees and 2 target estimation accuracies (altogether $3 \times 2 = 6$ cases), Fig. 8(b) shows the theoretical sampling sizes using Equation 4. Then we perform the inter-peer deduplication using MinHash with the corresponding sampling size. As Fig. 8(a) shows, all the achieved deduplication qualities slightly exceed the targets, which verifies the correctness and goodness of our rule of setting sampling size.

2) Trace-driven Results: To eliminate the duplication for Workgroup and Mailgroup, we should first fix their sampling sizes. As Fig. 2 shows, 4 MVHs could cover 99% inter-peer

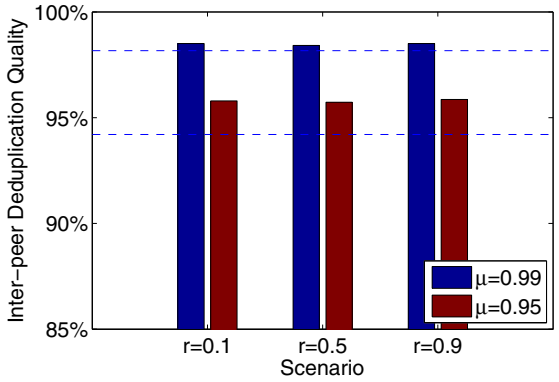


(a) The achieved inter-peer deduplication as the sampling ratio varies.

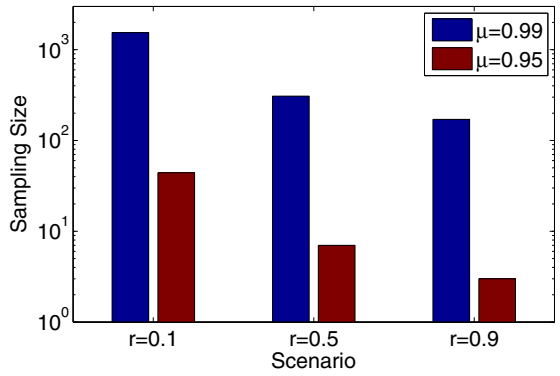


(b) The number of valid MVHs as the sampling ratio varies.

Fig. 7. The performance of our probabilistic estimation algorithm. 3 scenarios of different inter-peer redundancy degrees are examined. The X axis is logarithmic scale.



(a) The achieved deduplication qualities in comparison with the targets.



(b) The theoretical sampling size $\widehat{s}^*$ for the different deduplication cases.

Fig. 8. The effectiveness of our rule of setting sampling size. The dashed line refers to the deduplication quality under the target estimation accuracy μ. The up one is for $\mu = 99\%$ and the low one is for $\mu = 95\%$.

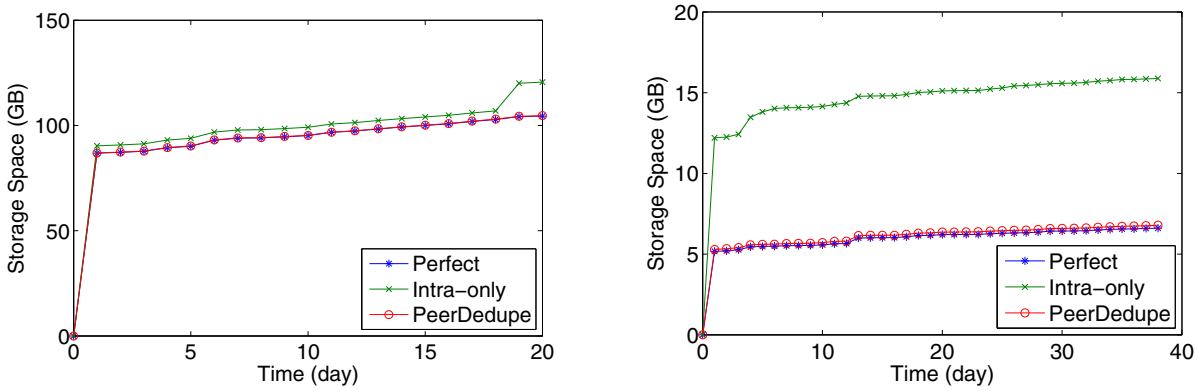
duplication. We target to achieve 99% estimation accuracy with 4 MVHs, i.e., eliminating 98% ($\approx 99\% \times 99\%$) inter-peer duplication. The inter-peer redundancy degree r and the duplication distribution $F(x; \lambda, \alpha)$ can be estimated with the knowledge of users' previous backups. Using Equation 4, the theoretical sampling size is 518 for Workgroup (a sampling ratio of 0.05%) and 162 for Mailgroup (a sampling ratio of 0.8%).

Fig. 9 tracks the storage consumption of the intra-only deduplication (i.e., traditional client-side deduplication which only perform the intra-peer deduplication) and PeerDedupe (with 4 MVHs). For comparison, we also draw the perfect deduplication curve (all duplicate chunks are removed system widely). For Workgroup, after 20 days, perfect deduplication used 104.47GB storage space, while the intra-only deduplication used 120.61GB (introducing 15.45% extra storage space) and PeerDedupe used 104.62GB (only introducing 0.14% extra storage space). For Mailgroup, the benefit of performing the inter-peer deduplication with PeerDedupe is more prominent. After 38 days, perfect deduplication used 6.61GB storage space, while the intra-only deduplication used 15.88GB (introducing 140.24% extra storage space) and PeerDedupe used 6.78GB (only introducing 2.6% extra storage space). We see that for both data sets, PeerDedupe achieves near-perfect deduplication quality. Note there are two outlier points on the 19th day for Workgroup and the 4th day for Mailgroup. The former is because of an update operation of SVN source code and the latter is because of a dissemination of a large email.

We take a closer look at PeerDedupe's key process of performing the inter-peer deduplication. As Fig. 10 shows, PeerDedupe removes 99.1% inter-peer duplication with 4 MVHs for Workgroup and 98.1% with 4 MVHs for Mailgroup, both of which meet our expectation ($> 98\%$). Therefore, PeerDedupe can achieve a desirable high deduplication quality with each peer coordinating with only a few other peers.

D. Server RAM Reduction

We set the sampling ratios for the Simugroup to achieve more than 98% inter-peer deduplication quality with 5 MVHs. They are 0.1%, 0.02% and 0.005% for the redundancy degrees of 0.1, 0.5 and 0.9. The server RAM reduction factor for



(a) For Workgroup. (b) For Mailgroup.

Fig. 9. Deduplication quality on a day to day basis. For Workgroup and Mailgroup traces.

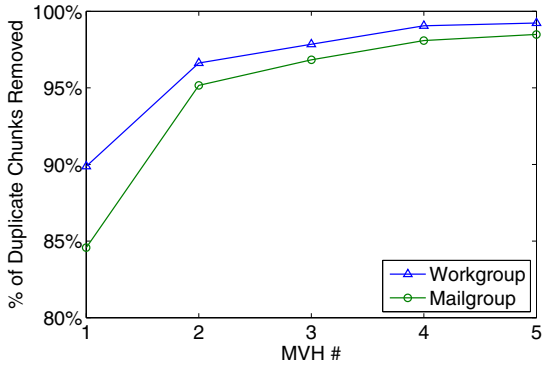


Fig. 10. The inter-peer deduplication quality for Workgroup and Mailgroup traces.

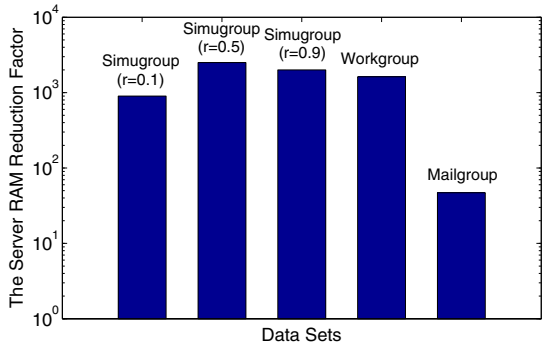


Fig. 11. The the server RAM reduction factors for Simugroup, Workgroup and Mailgroup.

Simugroup is calculated as follows.

Assume there are n peers in the system, each peer backs up b local unique chunks, and the inter-peer redundancy degree is r. With a sampling ratio of σ, PeerDedupe only needs to index $nb\sigma$ chunks at the server side. The number of global unique chunks is denoted as D, having $nb(1-r) \leq D \leq nb$. Thus, the server RAM reduction factor is $\tau = \frac{D}{nb\sigma}$, having $\frac{1-r}{\sigma} \leq \tau \leq \frac{1}{\sigma}$.

Fig. 11 shows the theoretically conservative server RAM reduction factor for Simugroup and the real reduction factors for Workgroup and Mailgroup. As can be seen, For Simugroup and Workgroup, we can achieve the reduction factor of about 1000 (or more). The reduction factor for Mailgroup is much less than that for Simugroup and Workgroup. That is because peers in Mailgroup back up much less data. Recall that the setting of the sampling size is not related to a peer's backup size (see Equation 4). Therefore, the more data peers back up, the lower the sampling ratio is and the higher the server RAM reduction factor we can achieve.

E. Peer-side Overheads

The peer-side overheads are critical for PeerDedupe's practicality. In this part we evaluate the overheads according to three aspects: RAM usage, CPU time and communication cost. We consider a typical scenario where a peer has 8GB backup data with 100MB local increments every day (a peer who has a smaller size of backup data or local increments will have smaller overheads). The data is split into chunks of 8KB length on average using Rabin fingerprint.

In total, each peer has no more than 1M locally unique chunks. We choose SHA-1 to compute a chunk's ID (20 bytes per ID). Therefore, keeping 1M SHA-1 hash codes only consumes 20MB memory, which is affordable for most of today's PCs.

The CPU time overhead for PeerDedupe includes chunking, hashing and searching chunk IDs in the memory-based index. For the first time, it may take a while because the peer has to perform chunking and hashing for all of his backup data. For his following backups, chunking and hashing are only for the local increments. We evaluated the CPU time overhead on our desk-top (Windows XP operating system, Intel(R) Pentium(R) D CPU 4 2.80GHz 2.79GHz and 1.00GB RAM), where the Red-Black tree (implemented in java.util.TreeMap) is used to index the chunks. The result shows that the CPU time overhead for a peer's incremental backup is only 34.2 seconds (16.1 seconds for chunking, 2.2 seconds for hashing and 15.9 seconds for chunk ID searching) which is a very light burden for the peer.

Besides deduplicating his own chunks, a peer also serves as

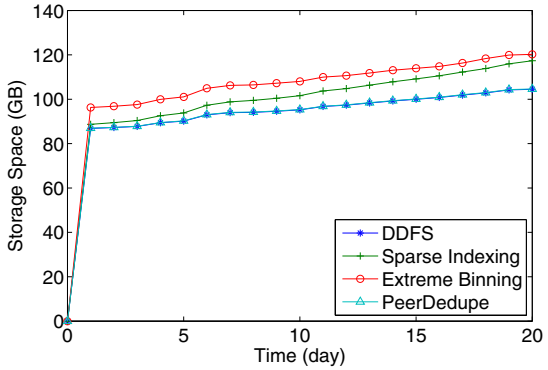


Fig. 12. Deduplication quality of DDFS, Sparse Indexing, Extreme Binning and PeerDedupe. For Workgroup trace.

the MVH to help other peers deduplicate chunks. However, as his received chunk IDs are of the initiator's local increments whose size is very small compared to a peer's full backup size, this part of chunk ID searching time can be ignored.

In PeerDedupe, the initiator and the selected MVHs negotiate to eliminate the inter-peer duplication by sending *chunk IDs*, which introduces extra bandwidth consumption and probably increases the backup time. However, because of the negotiation, most of the inter-peer duplication is removed, which saves the bandwidth and decreases the backup time on the other hand. It is obvious that this communication cost is too small compared to the brought benefit, because the cost is measured on chunk ID (20 bytes per ID) while the benefit is measured on chunk content (about 8 kilobytes per chunk).

F. Comparisons with Existing Works

In this part, we compare PeerDedupe with three recent data deduplication methods, DDFS [3], Sparse Indexing [4] and Extreme Binning [20], all of which perform chunk-level deduplication. DDFS achieves perfect deduplication by eliminating all the duplicate chunks. It keeps an in-memory bloom filter of the chunk IDs and a small cache. Sparse Indexing groups chunks into larger segments and keeps in-memory a low ratio (e.g., 1/64) of sampling chunk IDs for each segment. Incoming segments are deduplicated against a limited number of the most similar segments. Extreme Binning chooses one representative chunk for each file and groups similar files (having the same presentative chunk) into one index entry. For an incoming file, it is only deduplicated against its similar files (i.e., the ones having the same representative chunk).

We first compare PeerDedupe with the three works according to the deduplication quality. As in Fig. 12, PeerDedupe behaves almost as well as DDFS (perfect deduplication), while Sparse Indexing and Extreme Binning leave more deduplication in the system.

PeerDedupe also uses much less server RAM than the three works. Consider a digital repository with 100TB disk storage and 8KB chunk length, DDFS uses 12.5GB RAM and Sparse Indexing uses 3.9GB RAM [4], achieving the server RAM reduction factor of 20 and 64 respectively. The RAM usage of Extreme Binning depends on specific characteristics of the backups (e.g., the number of chunks per file and the similarity of files). Take Workgroup as the case study. There are 376,626 unique files indexed by 354,790 entries (40 bytes per entry [20]). Thus the server RAM reduction factor for Extreme Binning is 23. Having been discussed in Section IV.D, PeerDedupe achieves the server RAM reduction factor of 1624 (the highest) for Workgroup trace.

Higher server RAM reduction factor means lower cost of the back-end server. Though 12.5GB RAM usage seems affordable for a corporation who provides deduplication appliance, it will be a big cost when the data volume increases to petabyte level [21]. Also, higher server RAM reduction factor means that with an equal size of the server RAM, data is allowed to be split into much finer grained chunks, so more identical data can be removed. Bobbarjung *et al.* [22] and Policroniades *et al.* [23] have performed extensive experiments on various data sets, confirming the remarkable storage saving by changing the expected chunk size from 8KB length to 4KB or 2KB length.

V. Related Work

Chunking has been widely used to detect inner file redundancy in deduplication system. Fixed-length chunking, as used in Venti [24], splits files into equal size of chunks and removes the duplicate copies. It is simple to implement but suffers from the well known *boundary-shift problem* [25]: if one chunk in a file changes (e.g., addition or deletion), its subsequent fixed-length chunks will be all impacted, which dramatically decreases the detection volume of the duplication.

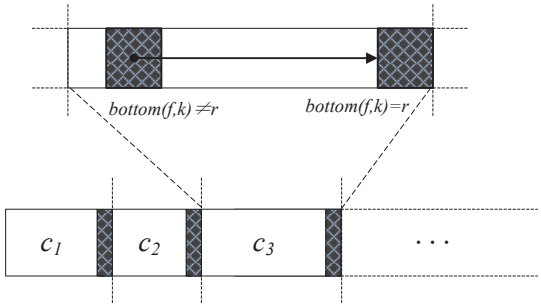


Fig. 13. Variable-length chunking.

Variable-length chunking avoids the boundary-shift problem by using the content-defined chunking algorithm (CDC) [26]. Fig. 13 depicts its basic mechanism. A sliding window of fixed length (e.g., 48 bytes) moves, one byte every time, from the beginning of a file to the end. At every position, a hash of the sliding window, f, is computed using Rabin fingerprint [10]. If the lowest k bits of the hash match a predetermined value r (i.e., $bottom(f,k) = r$), the offset is marked as an anchor. As can be seen, anchors divide a file into variable-length chunks which are generated according to the content, so any addition or deletion of a chunk would at most affect the chunking result of itself and its neighbors.

Recent server-side deduplication methods focus on overcoming the throughput and scalability limitations by solving

the disk bottleneck problem. The Data Domain File System (DDFS) [3], Sparse Indexing [4] and Extreme Binning [20] design compacted memory indexes which reduce the server RAM usage and the disk access frequency to some extent. The details have been given in Section IV.F. Debar [21] proposes a two-phase deduplication scheme. In phase one, a user's incoming backup data is deduplicated against his historical backups to perform preliminary filtering. The remaining chunks are deduplicated through sequentially searching the disk index in phase two, among which the chunk locality [3], [4] are explored to reduce the disk access. As to the peer-assisted deduplication method, a simple form has been introduced into EMC's Avamar system and Symantec's NetBackup system, but few in-depth researches on this topic have been done.

The idea of organizing peers in a system to cooperatively work has been well researched in peer-to-peer systems [9], [27], [14]. The key problem concerning the cooperation is the partner (or called helper) selection. For FriendStore [27], a peer stores the data at his friends in reality to improve the data durability. For AmazingStore [9], a peer stores the data at other peers with different online patterns to improve the data availability. For Pastiche [14], a peer stores the data at his most similar peers (in terms of the common data) to save the system storage space. PeerDedupe is somewhat similar to Pastiche, but their key processes of selecting helper peers are different because of their different objectives. PeerDedupe is designed for deduplication system, in which a peer deduplicates against his most valuable helpers to maximize the duplication elimination. Note that the most valuable helpers in PeerDedupe are different from the most similar peers in Pastiche (see Section III.A).

VI. Conclusion and Future Work

Through in-depth investigations on three different real-world traces, we observe that the inter-peer duplication accounts for a large proportion of the total duplication and exhibits strong peer locality. Based on these observations, we present PeerDedupe, a novel peer-assisted sampling deduplication approach. For the key problem of selecting MVHs, we present a probabilistic estimation algorithm, which relies on an effective sampling method and the corresponding rule of setting sampling size. Experiments on one large-scale synthetic data set and two real-world traces show that PeerDedupe can remove over 98% inter-peer duplication with each peer coordinating with no more than 5 MVHs. Compared to three existing works, PeerDedupe uses much less server RAM.

In our future work, we would implement PeerDedupe into AmazingStore [8], [9] system to further verify PeerDedupe's practicality and effectiveness. Issues, such as data consistency and privacy protection, will be carefully treated. The data consistency problem is: the inter-peer deduplication process of PeerDedupe relies on the helpers' historical backups, so if a helper changes his backup statuses (e.g., delete a file) within his initiator's backup window, conflicts may occur (e.g., leave out the file which is deleted by the status-changed helper). The privacy protection problem is: the helper who has identical data, let us say file A, with the initiator will know the initiator also has file A, but the initiator may not want others to know what kind of data he has.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. 60873051.

References

[1] E. Francen, "Hot IT topics for 2008," http://www.allbusiness.com/technology/computer-software/4967761-1.html, Oct. 2007.
[2] B. Pariseau, "Data deduplication and backup approaches in enterprise storage today," http://searchdatabackup.techtarget.com/generic/0,295582,sid187_gci1353079,00.html, Apr. 2009.
[3] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *FAST '08*.
[4] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: large scale, inline deduplication using sampling and locality," in *FAST '09*.
[5] "Efficient data protection with emc avamar global deduplication software," EMC, Tech. Rep., 2010.
[6] D. Hochba, "Approximation algorithms for NP-hard problems," *ACM SIGACT News*, 1997.
[7] A. Broder, "On the resemblance and containment of documents," in *SEQUENCES '97*.
[8] "Amazingstore: a smart client for sharing and storing," http://en.amazingstore.org/.
[9] Z. Yang, B. Y. Zhao, Y. Xing, S. Ding, F. Xiao, and Y. Dai, "Amazingstore: Available, low-cost online storage service using cloudlets," in *IPTPS '10*.
[10] M. O. Rabin, "Fingerprinting by random polynomials," Center for Research in Computing Technology, Harvard University, Tech. Rep., 1981.
[11] D. Hochbaum and A. Pathria, "Analysis of the greedy approach in problems of maximum k-coverage," *Naval Research Logistics*, 1998.
[12] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bulletin del la Société Vaudoise des Sciences Naturelles*, 1901.
[13] W. G. Cochran, *Sampling techniques*, 3rd ed. Wiley, 1977.
[14] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: making backup cheap and easy," *SIGOPS Oper. Syst. Rev.*, 2002.
[15] D. Bhagwat, K. Eshghi, and P. Mehra, "Content-based document routing and index partitioning for scalable similarity-based searches in a large corpus," in *SIGKDD '07*.
[16] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *COM '00*.
[17] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," *ToN '04*, 2004.
[18] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *Journal of Computer and System Sciences*, 2000.
[19] H. Pucha, D. G. Andersen, and M. Kaminsky, "Exploiting similarity for multi-source downloads using file handprints," in *NSDI '07*.
[20] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *MASCOTS '09*.
[21] T. Yang, J. Hong, F. Dan, N. Zhongying, Z. Ke, and W. Yaping, "Debar: A scalable high-performance de-duplication storage system for backup and archiving," in *IPDPS '10*.
[22] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, "Improving duplicate elimination in storage systems," *Trans. Storage*, 2006.
[23] C. Policroniades and I. Pratt, "Alternatives for detecting redundancy in storage systems data," in *ATEC '04*.
[24] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *FAST '02*.
[25] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content-based chunking algorithms," Hewlett-Packard Laboratories, Palo Alto, CA, Tech. Rep., 2005.
[26] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *SOSP '01*.
[27] D. N. Tran, F. Chiang, and J. Li, "Friendstore: cooperative online backup using trusted nodes," in *SocialNets '08*.