Efficient Similarity Estimation for Systems Exploiting Data Redundancy

Kanat Tangwongsan[1], Himabindu Pucha[2], David G. Andersen[1], Michael Kaminsky[3]
[1]Carnegie Mellon University, [2]IBM Research Almaden, [3]Intel Labs Pittsburgh

***Abstract*—Many modern systems exploit data redundancy to improve efficiency. These systems split data into *chunks*, generate identifiers for each of them, and compare the identifiers among other data items to identify duplicate chunks. As a result, chunk size becomes a critical parameter for the efficiency of these systems: it trades potentially improved similarity detection (smaller chunks) with increased overhead to represent more chunks.**

Unfortunately, the similarity between files increases unpredictably with smaller chunk sizes, even for data of the same type. Existing systems often pick one chunk size that is "good enough" for many cases because they lack efficient techniques to determine the benefits at other chunk sizes. This paper addresses this deficiency via two contributions: (1) we present multi-resolution (MR) handprinting, an application-independent technique that efficiently estimates similarity between data items at different chunk sizes using a compact, multi-size representation of the data; (2) we then evaluate the application of MR handprints to workloads from peer-to-peer, file transfer, and storage systems, demonstrating that the chunk size selection enabled by MR handprints can lead to real improvements over using a fixed chunk size in these systems.

I. Introduction

"We were most concerned about the choice of [chunk size]"
–Spring & Wetherall [22]

Exploiting cross-file redundancy has become an important technique to improve the efficiency of data transfer and storage—so much so that just recently, a filesystem "deduplication" company (Data Domain) was acquired for over two billion dollars. Redundancy elimination is widely used at the packet level (e.g., Spring and Wetherall [22], and WAN optimizers from companies such as Riverbed [21]); at the protocol level (e.g., Web page duplicate elimination [20]); for local [9] and remote filesystems (LBFS [13]); and for file transfers (`rsync` and several peer-to-peer systems [1, 6, 16]). In general, these systems identify chunks of data shared across files, in order to transfer or store them more efficiently.

For this reason, these systems all face the question of how to divide data into chunks. The goal of this chunking is to maximize the probability of finding common chunks while minimizing the overhead of transmitting and looking up chunk identifiers. Choosing the correct chunk size is a fundamental decision in managing this tradeoff. Existing systems choose a static value based upon one or a few intended data sets, and the range of these values spans several orders of magnitude:

System	Chunk Size
eMule (p2p)	9500 KB
BitTorrent (p2p)	256 KB
SET (p2p)	16 KB
LBFS (dist. fs)	8 KB
Shark (dist. fs)	8 KB
REBL (filesystem)	1 KB–8 KB
Rsync	Variable
Packet-level	64 bytes

In this work, we study a seemingly simple question, and provide a general-purpose technique and an empirical evaluation to help applications navigate this tradeoff: *how can two or more entities efficiently determine the best chunk size to use when transferring/storing data?*

Answering this question is both challenging and important. It is challenging because doing so *efficiently* requires improving upon previous work in similarity estimation. It is important because a poor chunk size selection harms efficiency: too large a chunk size reduces the exploitable redundancy between files, but too small a chunk size can greatly increase the overhead of representing and transferring the file. Unfortunately, the right chunk size can vary dramatically even within files of the same type.

To illustrate this point, consider an example from a real p2p file-sharing network: One popular English movie was 75% similar to the same movie in Italian using 1 KByte chunks, but only 3% similar using 128 KByte chunks. The movies have identical video streams, but different audio streams. Thus, the chunk size needs to be smaller than a single video frame to decouple audio and video. The difference in overhead is also large between small and large chunk sizes: A common 800 MByte movie in 1 KByte chunks has 800,000 chunks, each of which must be requested and accounted for separately. Using BitTorrent's 256 KByte chunk size, however, the same file would be represented as only 3,125 chunks.

Our solution involves an application-independent technique that efficiently estimates achievable benefit at different chunk sizes, a building block for optimal chunk size selection. The algorithm builds upon prior work in similarity detection [12, 2, 3]; its key contribution is providing bounds on the size of the handprints that must be compared in order to obtain accurate estimates, and using these bounds to design estimation techniques that have very low overhead. It operates using a compact, multi-size representation of the data's content, called a *multi-resolution* (MR) *handprint*. We show analytically the error bounds on such an estimate and provide theoretical results

indicating how many chunks must be included in the MR-handprint, at different sizes, so that the similarity estimate falls within those bounds (Sections III, IV). Although the techniques presented here are not tied to any particular chunking scheme, in this paper, we focus on Rabin-based chunking and in this particular case, a chunk size refers to the expected chunk size.

We empirically evaluate our algorithm using workloads from a p2p file system and an RPM/ISO mirror site. Our results suggest that an MR-handprint consuming 0.15% of the size of the file can be used to estimate to within 5% (and often much more closely) the similarity between two files at chunk sizes of 1, 2, 4, ..., 128 KBytes (plausible chunk sizes in a p2p file transfer system).

We then illustrate the utility of multi-resolution handprinting in both our workloads. As discussed in Section V, in the p2p scenario, MR-handprinting identifies that only a small fraction of files actually benefit the most at a static chunk size of 16 KBytes (as used in [16]); 40% of the file transfers would have higher performance with larger chunk sizes, and $\sim 60\%$ of the transfers would have higher performance with chunk sizes smaller than 16 KBytes. Similarly, in the RPM/ISO scenario, $\sim 20\%$ of files can be transferred faster with a larger chunk size, while $\sim 80\%$ of files benefit from smaller chunk sizes.

II. Motivation and Background

A. Why different chunk sizes?

We motivate this work by presenting the observation that led to this research: the drastically different similarity observed at different chunk sizes for different types of objects, and for different objects of the same type. We first examine a handful of example files sampled from a p2p file sharing system and a collection of ISO/RPM files to understand similarity at different chunk sizes; we examine a larger collection of files in Section V. We begin with two definitions:

Splitting data into chunks: Rabin Fingerprinting. In this paper, we adopt the now-common technique of dividing objects into chunks using content-determined boundaries, which render the chunk divisions insensitive to small insertions or deletions. This technique, termed Rabin fingerprinting and first used in LBFS [13], runs a sliding-window hash, covering about 40 bytes, over the data, declaring a chunk boundary whenever the k-lowest-order bits of the hash (of those 40 bytes) are zero. For efficiency, it uses Rabin polynomial fingerprints as the hash [19]. The value of k determines the expected chunk size.[1] The effect of choosing boundaries based upon the content of the object is that an edit that adds or removes a small amount of data will only change the chunking in a local area of the object, but the rest of the object will still have the same chunks as before.

Similarity Metric. We define the similarity between two sets of chunks A and B as:

$$s(A, B) = \frac{|A \cap B|}{|A|}$$

[1]We follow the LBFS example and also set a minimum and maximum chunk size as a function of the expected chunk size to bound the resulting chunk sizes under pathological inputs.

This definition matches the goals of a system exploiting redundancy—it is, in essence, a measure of how useful object B is when trying to transfer/store object A. (This notion of similarity, originally called *containment*, was proposed and studied by Broder [2] and Broder et al. [3]).

Similarity in Real-World Files: Figures 1(a), 1(b), and 1(c) show similarity vs. chunk size for a small set of audio, video, and software files, respectively. Each graph has four lines, with each line showing how similar one pair of files was as the chunk size increases from 1 KByte to 128 KBytes.

For instance, in the ISO/RPM files graph (Figure 1(c)), the top line (Example 1) shows the similarity between two install discs for Yellowdog Linux, one from March 2003 and one from May 2003. In contrast, the line that drops drastically with chunk size (Example 2) is two source code RPMs of `gcc`, one `gcc-2.96-128.7.2` and the other `gcc-2.96-113`.

These graphs illustrate that among individual files, some have roughly constant similarity across chunk sizes (and thus, could use a large chunk size to reduce overhead). Other files (e.g., Figure 1(a), example 3, and Figure 1(c), example 2) have similarity that "crashes" after a certain chunk size—but that optimal chunk size varies between different file types. Still, others have similarity that decreases more slowly, or linearly, with increasing chunk size.

These were, of course, only sixteen example file pairs out of the entire universe of files that users may wish to transfer, but we observe similar patterns in larger sets of files: In a large collection of video files, a small number of files are nearly 200x more similar at 1 KByte chunk size than they are at 128 KByte chunks. The decrease in video file similarity past 4–8 KBytes is striking, because of the interleaving effect mentioned earlier. In contrast, the similarity of audio files drops less sharply, because most similar audio files differ only in the first or last chunk.

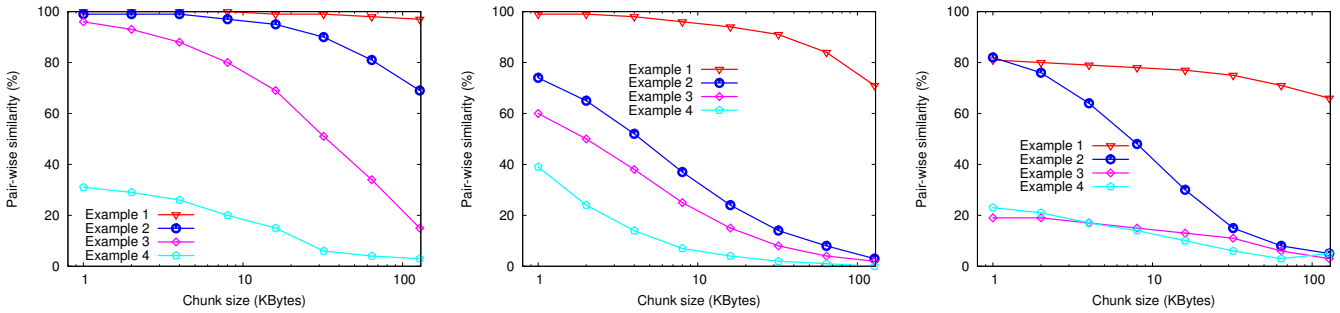
B. Example: P2P File Transfers

To provide context for the remainder of the paper, we briefly outline how we envision MR-handprints to be used in our example case study of p2p file transfer systems. This is neither the only use nor a complete design of a file-transfer system based upon MR handprints. A file transfer system augmented with MR handprints will involve the following steps[2]:

(1) Identify candidate similar files (techniques proposed in [16] can be used);
(2) Retrieve MR handprints for each candidate file;
(3) Estimate similarity to desired file at different chunk sizes;
(4) Determine the best chunk size to balance increased similarity with increased overhead;
(5) Obtain the descriptor lists for the best chunk size; and
(6) Transfer the file.

In this context, our work focuses on providing tools for steps 3 and 4 that keep the MR handprints small enough that step 2 remains practical. The MR handprints can be compact because they are only used for similarity estimation. The descriptor lists used in the actual transfer still contain full hashes of the chunks (e.g., 160-bit SHA-1 hashes as used in [16]).

[2]For details on a similarity based file transfer system, please refer to [16].



(a) Audio files (b) Video files (c) ISO and RPM files

Fig. 1: Pair-wise similarity among files as chunk size is varied.

III. Estimating Similarity

In this section, we present a design for lightweight MR-handprints and show that they provide accurate estimates of similarity. A MR-handprint is a compact representation of a data item (e.g., a file) with the property that given the MR-handprints of *any* two data items, their similarity at all chunk sizes of interest can be estimated efficiently. Our method of generating handprints follows the work of Broder [2] for estimating both our notion of similarity (which he calls containment) and the Jaccard coefficient; however, Broder's analysis did not provide a comprehensive guideline for picking the parameters to control the accuracy and size of the resulting handprints. More recently, Pucha et al. [16] proposed a constant-size handprinting scheme for detecting whether two files share *any* chunk and give a recipe for controlling the size of the handprint while ensuring high accuracy. Their handprinting scheme, however, does not lead to accurate similarity estimates in our setting.

Generating the multi-resolution handprint: The MR handprint contains, for each chunk size of interest, a subset of the hashes of the chunks in the data (e.g., a file). A chunk hash is included in the set if it is zero modulo some number k (e.g., if $k = 8$, then, in expectation, $\frac{1}{8}$th of the chunk hashes will be included in the handprint); i.e., for a set of hashes A, $\mathcal{H}(A) := \{h \in A : h \bmod k = 0\}$.

Important to this definition is to note that inclusion in the set is a property of the chunk: if a chunk is in A's handprint, that chunk will also be in the handprint of any file B that also contains the chunk. If the hash value of a chunk is effectively random (e.g., if the hash is secure), then any particular chunk essentially has a probability $\alpha = 1/k$ of being included in the handprint (independent of other chunks)[3].

We vary α, which determines the handprint's size, based upon the chunk size for reasons we explain below. For implementation convenience, we round up to the nearest power of two. For example, in a p2p file sharing system, α can be chosen as follows:

Chunk Size	1KB	2KB	4KB	8KB	$\geq$ 16KB
α	1/16	1/8	1/4	1/2	1

[3]Formally, each chunk e is associated with a Bernoulli random variable $b_e \sim \mathsf{Be}(\alpha)$; we define $\mathcal{H}(A) = \{h \in A : b_h = 1\}$.

These values are slightly more conservative than necessary (the fraction of chunks included could be somewhat smaller), but they produce highly accurate estimates while resulting in an MR-handprint that is only 0.15% the size of the file (using 40-bit hashes as described in Section III-B).

Estimating similarity using two handprints: The process is simple: Count the fraction of fingerprints in A's handprint that also appear in B's handprint. Use this fraction as an estimate of how similar A and B are. In other words,

$$\text{use } \frac{|\mathcal{H}(A) \cap \mathcal{H}(B)|}{|\mathcal{H}(A)|} \text{ as an estimate of } \frac{|A \cap B|}{|A|}$$

A. MR-Handprint Accuracy

We need to answer two key questions before using MR-handprinting in a system:

(1) What fraction of chunk hashes should be included in the handprint for a particular chunk size?
(2) How accurate is the similarity estimate?

More concretely, we wish to understand the probability that the estimated similarity differs from the true similarity by more than a small amount δ. In particular, we study $\mathbf{Pr}\,[|\widehat{s} - s| \leq \delta]$, where s is the true similarity, and $\widehat{s}$ the estimated similarity.

The variable we control is α, the probability that a given chunk is included in the handprint. By increasing this probability, we expect to decrease the error.

We attack this problem in two ways. First, we numerically compute the probability that an estimate is within δ for a few values of δ, α, and $|A|$ that we are particularly concerned with. Because this numerical evaluation is computationally intensive, we then derive a weaker, but tractable, lower bound on the confidence that can provide an understanding of how the confidence scales with chunk size, the number of chunks, and α. (The theoretical bounds are also useful for estimating confidence for data whose size is too large to directly compute.)

Confidence interval result summary: Table I shows the probability that the similarity estimates are within 5% for a large data item (32K chunks). Two things are clear: First, the confidence values are high. Using a handprint that contains only 1 chunk in 32 still results in an estimate that is within 5% of the true value 98% of the time. Second, the theoretical

Inclusion Probability (α value)	Confidence (bounds)	Confidence (numerical)
$\alpha = 1/2$	≈ 1	≈ 1
$\alpha = 1/4$	> 0.999	> 0.999
$\alpha = 1/8$	0.982	> 0.999
$\alpha = 1/16$	0.758	> 0.981
$\alpha = 1/32$	0.234	> 0.978

TABLE I: Confidence that similarity estimate is within 5% ($\delta = 0.05$) as the fraction of chunk hashes included in the MR-handprint varies (α). $|A| = 32,768$; files are 50% similar ($s = 0.5$).

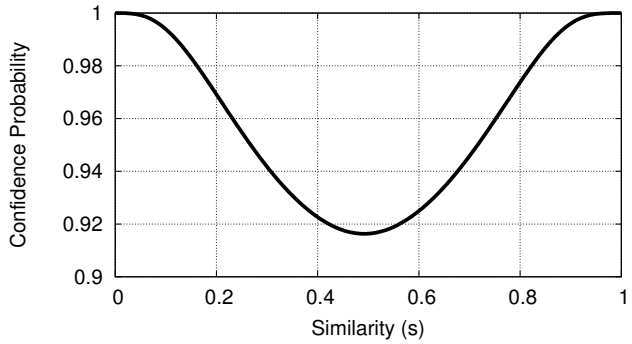


Fig. 2: Numerically calculated confidence probability with 2048 chunks, $\delta = 0.05$, and $\alpha = 1/8$.

bound is not very tight for smaller values of α, but it provides a useful answer when the number of chunks is large: choosing $\frac{1}{8}$ or $\frac{1}{16}$th of the 1 KByte chunks provides very high confidence that the estimate is within a few percent. As described later, using an MR-handprint with this fraction of chunks results in the MR-handprint with $\approx 0.15\%$ of the original content size, which seems a reasonable overhead.

Directly calculating the confidence values: We derive an expression for the probability that $|\mathcal{H}(A) \cap \mathcal{H}(B)| = x$ and $|\mathcal{H}(A)| = y$. Let

$$f(x,y) = \alpha^y (1-\alpha)^{|A|-y} \binom{s|A|}{x} \binom{(1-s)|A|}{y-x}.$$

The confidence probability, then, is $\sum_{(x,y)\in\Gamma} f(x,y)$, where $\Gamma = \{(x,y) : |s - \frac{x}{y}| \le \delta,\ 0 \le y \le |A|,\ \text{and } y + |A \backslash B| \le x \le \min(s|A|, y)\}$. To compute this exact confidence probability, we sum the probability of all possible outcomes in which the additive error is at most δ. This calculation takes about 20 minutes to perform for $|A| = 10,000$—fast enough for offline computation, but too slow for on-line use. Since the computation time grows rapidly with $|A|$, when the number of chunks is large, we are limited to the theoretical bounds established in the following section.

To understand how the confidence probability varies with similarity level, Figure 2 shows the confidence probability at different similarity level for $|A| = 2048$ and $\alpha = 1/8$. The lowest confidence is when $s = 0.5$. We therefore use this value of similarity in Table I.

Confidence Bounds: We establish a theoretical bound for $\mathbf{Pr}\left[|\widehat{s} - s| \le \delta\right]$, the probability that the similarity estimate is within $\pm\delta$ of the true similarity, as a function of the data size $|A|$, the true similarity s, and the fraction of chunk hashes in the handprint α.

Let us begin by stating a useful observation: the intersection of the two handprints is the handprint of the intersection of two set of chunks (i.e., $\mathcal{H}(A) \cap \mathcal{H}(B) = \mathcal{H}(A \cap B)$. This is because as noted earlier, inclusion in the handprint is a property of the chunk.

Now we can lower-bound the confidence probability of the overall estimate by bounding the error probability of two "bad events"—an excessive error in the denominator (the size of the handprint of A) and in the numerator (the intersection of the two data items) of the similarity estimate—and then bounding the probability that neither error occurs. Let λ be a parameter to be set later.

(1) Bad event for denominator:

$$\mathcal{E}_1 := \{\big||\mathcal{H}(A)| - |A|\alpha\big| > \lambda|A|\alpha\}$$

(2) Bad event for the numerator: Recall $|A \cap B| = |A|s$.

$$\mathcal{E}_2 := \{\big||\mathcal{H}(A \cap B)| - |A|s\alpha\big| > \lambda|A|s\alpha\}$$

If neither bad event occurs, then the estimate error δ is less than $\frac{2\lambda}{1-\lambda}s$, so we set $\lambda = \frac{\delta/s}{2+\delta/s}$. Note that the overall probability of both conditions is at least the probability of them both occurring independently, and so we can calculate the confidence probability as $(1 - \mathbf{Pr}\left[\mathcal{E}_1\right])(1 - \mathbf{Pr}\left[\mathcal{E}_2\right])$. This is possible because the events $\overline{\mathcal{E}}_1$ and $\overline{\mathcal{E}}_2$ are negatively dependent—e.g., the size of A's handprint *must* be larger than the size of the intersection of A and B's handprint, so the probability of $\overline{\mathcal{E}}_1$ occurring, given that $\overline{\mathcal{E}}_2$ occurred, is *at least* as high as the probability of $\overline{\mathcal{E}}_1$ occurring alone.

Finally, we bound the above bad probabilities using the Chernoff bounds, resulting in the following theorem:

Theorem 1. *Let* $\lambda = \frac{\delta/s^2}{2+\delta/s^2}$. *The probability* $\mathbf{Pr}\left[|\widehat{s} - s| \le \delta\right]$ *is at least*

$$\left(1 - 2e^{-\frac{\lambda^2}{2} \cdot \frac{\alpha|A|}{(1-\alpha)+\lambda(1-2\alpha)/3}}\right)\left(1 - 2e^{-\frac{\lambda^2}{2} \cdot \frac{\alpha s|A|}{(1-\alpha)+\lambda(1-2\alpha)/3}}\right)$$

Proof Sketch. First, write the size of a handprint as a sum of indicator random variables:

$$|\mathcal{H}(A)| = \sum_{e\in A} \mathbf{1}_{\{e\in\mathcal{H}(A)\}} = \sum_{e\in A} X_e,$$

where $X_e \sim \mathsf{Be}(\alpha)$. Thus, $\mathbf{E}\left[|\mathcal{H}(A)|\right] = \alpha|A|$ and $\mathbf{E}\left[|\mathcal{H}(A) \cap \mathcal{H}(B)|\right] = \alpha|A \cap B|$. Then we lower bound $\mathbf{Pr}\left[\mathcal{E}_1\right]$ and $\mathbf{Pr}\left[\mathcal{E}_2\right]$ using the Chernoff bounds (e.g., [10]). As observed earlier, $\mathbf{Pr}\left[|\widehat{s} - s| \le \delta s\right] \ge (1 - \mathbf{Pr}\left[\overline{\mathcal{E}}_1\right])(1 - \mathbf{Pr}\left[\overline{\mathcal{E}}_2\right])$, so plugging in the lower bounds yields the desired result. □

Understanding the bounds. First, and most importantly, *as the chunk size increases, we must increase the fraction of chunks included in the handprint to maintain a constant error probability.* Note that in the formula, the numerator depends on $\alpha|A|$—that is, the accuracy decreases as the number of chunk hashes decreases. The denominator also depends on α—the accuracy increases as more chunks are included. As a

result, doubling the chunk size halves $|A|$, so to keep the error probability, α must increase to $(\frac{2\alpha}{1+\alpha})$. As noted earlier, in our implementation, we round these numbers to nearby powers of two fractions (1/8, 1/16, etc.) for efficient computation.[4]

Second, *the error probability decreases with increasing number of chunks.* Because finding similar items is most important for larger data items whose transfer time is long, this means that a setting of α that works for the minimum size data of interest will work for all data sizes of interest.

Finally, *the confidence of the bound increases with increasing similarity—but this differs from the directly calculated numbers.* This difference shows part of the reason the theoretical bounds are not tight: they ignore the dependence information between the numerator and the denominator; our analysis simply made use of negative dependence. This provides guidance for future work in tightening the bounds we have established.

B. Reducing MR Handprint Size

Next we explore how many bits are needed to represent each chunk's hash in order to maintain high confidence in the accuracy of the similarity estimate, while reducing the size of the MR-handprint. The answer, which we derive below, is that it suffices to use 40 bits to represent each chunk ID in the MR-handprint. (Recall that the MR handprint for a 1-gigabyte file would contain up to 256,000 chunk hashes, so this savings is substantial.)

Thus far, we have relied upon the common assumption that the output of a cryptographically strong hash function (i.e., 160-bit SHA-1) produces a unique identifier for each chunk. Although this is certainly a safe way to proceed, our similarity estimation does *not* require cryptographically strong identifier generation, merely IDs that, with high probability, are unique. More concretely, assuming we wish our estimates of similarity to be within δ (e.g., 1%, 5%, etc), then we can tolerate a small possibility of collision, as long as the collisions do not reduce the estimate's accuracy past a tolerance. Our analysis below closely parallels Broder's analysis for the Jaccard coefficient [2].

Formally, let f be a (random) function from the set of all possible chunks to the set of ℓ-bit binary strings. Intuitively, if $\ell \gg \log_2(|A|)$, then the output of the hash function should be unique over the possible set of input chunks. But how large is large enough, given our goal of not having more than a small percentage error? To answer this question, first we note that

$$\mathbf{E}\left[|f(A)|\right] \approx |A| - \binom{|A|}{2} \Big/ 2^{\ell}. \tag{1}$$

Therefore, in the context of similarity estimation, if ℓ is large enough, the quantity $\frac{|f(A)\cap f(B)|}{|f(A)|}$ is a good approximation for $\frac{|A\cap B|}{|A|}$. To quantify this relationship more precisely, we first examine the probability that there are substantially more unique input chunks than output hashes when hashing to ℓ-bit strings. Using Markov's inequality and Equation (1), we can show that

$$\mathbf{Pr}\left[|A| - |f(A)| \geq 1/\delta\right] \leq \delta \cdot \binom{|A|}{2} \Big/ 2^{\ell}.$$

[4] α can actually increase somewhat more slowly: $\frac{1}{16}, \frac{2}{17}, \frac{4}{19}, \frac{8}{23}, \frac{16}{31}, \ldots$.

If we wish to tolerate no more than 1% error in the similarity estimation due to collisions, we can use this bound to compute the minimum size of ℓ. 4 GBytes of data has approximately 2^{22} total 1 KByte chunks, and so the MR-handprint contains $\frac{1}{8}$th of those, or 2^{19} chunks. If we desire accuracy within 1% ($\delta = 0.01$), then, with 40-bit hashes:

$$\begin{aligned}\mathbf{Pr}\left[|A| - |f(A)| \geq \tfrac{1}{0.01\cdot|A|}\right] &\leq \frac{0.01}{2^{19}} \cdot \frac{\binom{2^{19}}{2}}{2^{40}} \\ &\leq 2.38 \times 10^{-9}\end{aligned}$$

In other words, for a 4 GByte file, by using only 40 bits of the hash, we have roughly a one-in-a-million chance of the similarity estimate being wrong by more than 1% due to collisions. (The chance of having *any* collisions is less than 25%, and allowing 1% error in estimation means that we can have roughly 5,000 collisions before causing excess error.) 40 bits gives substantial room for growth: A 128 GByte file would have only a 1 in 100,000 chance of having a similarity estimate wrong by more than 1% due to collisions.[5]

IV. Evaluation: Accuracy

How accurately do MR-handprints estimate similarity, using our chosen parameters to perform handprinting? We evaluate MR-handprinting using a large data set of real files from the Web, FTP sites, and file-sharing networks. We begin by examining the data sets and evaluation methods.

A. Methods

Data sets. We examined several large software collections as well as an existing data set of 1.7 TB of files downloaded from popular file-sharing networks.

ISO/RPM data set: The data in this set was collected via HTTP downloads from various software mirrors. Our data consisted of 17 GBytes of ISO files of various Linux distributions, 11 GBytes of different versions of Linux kernel RPMs, 1.2 GBytes of gcc RPMs and 1 GByte of perl RPMs.

p2p data set: This data set consists of files downloaded from the eDonkey and Overnet networks for three months between November 2006 and February 2007. The set was biased towards possibly-popular content: groups of unique files returned as search results for a given search query. The set includes "popular" content, chosen from Billboard top ten audio tracks and video box office results, and "unpopular" search terms from top song collections from 1990–2000 as well as from one of the data set collector's personal Netflix history. The data set includes download results from 15 popular and

[5] We note that one can achieve a slightly more compact encoding by using a Bloom filter to store the chunk hashes that make up the MR-handprint for a particular size. Using this technique can reduce the per-hash storage required to about 34 bits per chunk. We do not present this optimization here because it requires that a system making use of it now to represent the file using *three* mechanisms—the full chunk list for data transfer; the bloom-filter MR handprint; and a list of small chunk hashes for the target file to use to search within the bloom-filter handprint. This optimization may still be worthwhile in a system that compares the target file against *many* remote files.

Scheme	*P2P Data set*						*ISO/RPM Data set*					
	1-KByte Chunks			**128-KByte Chunks**			**1-KByte Chunks**			**128-KByte Chunks**		
	Avg	Stddev	Max	Avg	Stddev	Max	Avg	Stddev	Max	Avg	Stddev	Max
($\alpha = 1/8$, 40 bits)	0.00	0.001	0.16	0.01	0.027	0.97	0.00	0.002	0.11	0.00	0.003	0.94
($\alpha = 1/8$, 160 bits)	0.00	0.001	0.16	0.01	0.027	0.97	0.00	0.002	0.11	0.00	0.003	0.94
($\alpha = 1/16$, 40 bits)	0.00	0.002	0.16	0.01	0.053	0.98	0.00	0.003	0.45	0.00	0.004	0.94
($\alpha = 1/16$, 160 bits)	0.00	0.002	0.16	0.01	0.053	0.98	0.00	0.003	0.45	0.00	0.004	0.94

TABLE II: Accuracy (absolute error) of different MR handprint sizes when estimating pair-wise similarity.

11 unpopular song title/artist pairs, plus 14 popular and 12 unpopular movie titles.

These data sets are appropriate for use in a similarity study because they include similar files, though it is certainly only a lower-bound on the number of similar files (e.g., for any given file, it includes some likely similar files, but the entire universe of p2p likely would contain additional files that are similar to the target file).

Obviously fake files are removed by eliminating files that `rzip` compressed more than 20%. This step is a good filter with one-sided error, because media files are already compressed and should not be further compressable (non-compressable fake files will, of course, remain). The audio files were then passed through an MP3-to-WAV converter, and only working files were retained.

Evaluation Method. We split each file in the data set with eight different chunk sizes (1, 2, 4, ..., 128 KBytes) using Rabin fingerprinting. For each pair of files in the data set, we compute the true pair-wise similarity using all of the chunk hashes in the files. We then estimate the pair-wise similarity for each pair of files using the MR-handprints. Where appropriate, we vary the α parameter that determines the size of the handprints, and evaluate 160-bit vs. 40-bit hashes. We then measure the absolute error ($|\text{actual} - \text{estimated}|$) in our estimate.

B. Accuracy of Similarity Estimation

The accuracy of similarity estimation varies (potentially) with two factors: The fraction of chunks included in the MR-handprint and the number of bits used to represent the hashes in the handprint. Table II depicts the accuracy of using different handprint sizes (obtained by varying α and ℓ).

40-bit fingerprints are sufficient. Using 40-bit fingerprints instead of 160-bit fingerprints does not measurably degrade the accuracy of similarity estimation, as evidenced by Table II.

α is the most important determiner of accuracy. In general, increasing α increases the accuracy of estimating the pair-wise similarity. As Table II shows, the estimates are not perfect (merely quite close). Increasing the fraction of chunks clearly improves the estimate quality, reducing the mean error and substantially reducing the maximum error. We conclude from these two factors that it is better to spend bits on including more chunk hashes than on reducing the already-tiny chance of hash collision by using 40-bit fingerprints.

α must increase as the chunk size increases. Table II highlights the fact that the MR-handprints must contain a greater fraction of the chunks for a larger chunk size. The lines for 128 KByte chunks show substantially higher error at $\alpha = \frac{1}{8}$ than the 1 KByte chunk lines do. Figure 3 plots the 98th percentile error when using $\alpha = \frac{1}{8}$, showing that the error grows roughly linearly with each doubling of the chunk size. Using the variable-α MR handprints, the 98th percentile error is close to zero across all chunk sizes.

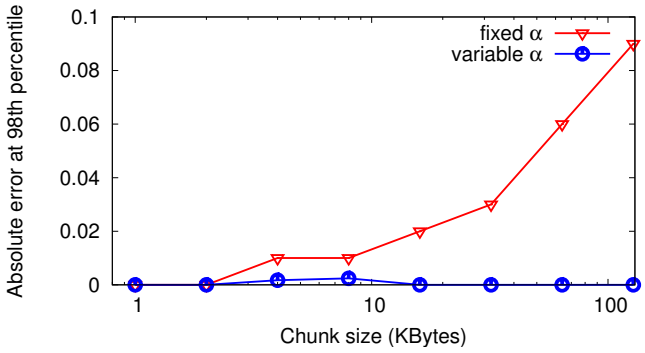


Fig. 3: 98th percentile absolute similarity estimation error in the p2p data set using 40-bit chunk hashes. The upper line shows the increasing error with large chunk sizes when using a fixed $\alpha = \frac{1}{8}$. The error using variable-α MR handprints is close to zero for all chunk sizes.

α for 1 KByte chunk sizes. In the P2P data set, using $\alpha = \frac{1}{32}$ of the chunks provides an accurate estimate (the stddev is at most $\frac{1}{3}$ of a percent off). In the ISO/RPM data set, the files are smaller, so the error at $\alpha = \frac{1}{32}$ is higher. Our default value of $\alpha = \frac{1}{16}$ works well for both the P2P data set and the small RPMs. Based both upon this empirical success and the theoretical bounds from earlier, we use this as the starting α value for the smallest chunks in the MR-handprint.

V. Evaluation: Effectiveness

The previous section showed that the similarity estimates provided by MR-handprints are accurate. In this section, we examine the application of MR-handprints to peer-to-peer file transfers (examining how many more sources of chunks an application could get by using MR-handprints instead of using a static chunk size) and storage systems (examining how much space the system could similarly save). The study reveals that commonly selected "compromise" chunk sizes (4, 8, or 16 KBytes for many research p2p systems) are very rarely the optimal chunk size—instead, a mix of larger chunk sizes (for files with large spans of identical data) and smaller chunk sizes (for files with tightly interleaved changes) provides substantially

more similarity for these systems to exploit with lower total overhead.

Using MR handprints to accurately estimate application metrics: A key result of this section is that the accurate similarity estimates from MR handprints are a good building block for determining metrics of interest to applications. We examine in this section two such potential metrics, selected from existing systems—creating realistic cost-benefit metrics is highly application-dependent and is not the focus of this paper. We view the development of a general cost-benefit analysis framework for evaluating this tradeoff as an interesting subject for future work, along with numerous other implementation issues of incorporating MR-handprints into real-world systems.

Metric 1: Parallelism Gain. The first metric is relevant to peer-to-peer file transfers. *Parallelism gain* (or pgain) captures the expected benefit from exploiting similarity [16]. pgain measures the increase in the number of sources from which a p2p receiver can draw when exploiting similarity. For example, if every chunk is available from two sources, the *pgain* is 2; if 99% of chunks are available from only one source, and the remaining chunks are available from 1000 sources, the *pgain* is only barely above 1. For the ISO data set, we compute a file-level pgain (the number of files from which a chunk can come) as a coarse estimate of the number of sources.

The measure *pgain* is the harmonic mean of the number of sources/copies available for each chunk: Given sets $A_0, A_1, \ldots, A_k$ of chunks, the pgain of A_0 with respect to the pool of chunks $\{A_0, \ldots, A_k\}$ is given by

$$pgain_{A_0} := \frac{|A_0|}{\sum_{e \in A_0} 1/O_e}, \tag{2}$$

where O_e is the number of sources or copies available for each chunk (i.e., $O_e := \sum_{i=0}^{k} \mathbf{1}_{\{e \in A_i\}}$).

Metric 2: Compression Ratio. The second metric we examine is relevant to both point-to-point transfers and storage systems. The compression ratio of a file represents what fraction of the file would need to be sent (or stored) if the receiver already possessed the other files in the set. Formally, the compression ratio of A_0 with respect to a pool of chunks $\{A_1, \ldots, A_k\}$ is computed as

$$cr_{A_0} := \frac{|A_0 \cap \cup_{i=1}^{k} A_i|}{|A_0|} \tag{3}$$

For both metrics, we ask two questions: First, how accurately can the MR-handprints be used to estimate the true value of the metric? Second, how much is the achieved value of the metric improved by using the best chunk size determined by MR handprints instead of using a statically-chosen chunk size?

MR-handprints accurately estimate pgain and compression ratio. Given the MR handprints, we estimate pgain in the same manner as similarity; using

$$\frac{|\mathcal{H}(A_0)|}{\sum_{e \in \mathcal{H}(A_0)} 1/C_e} \quad \text{where } C_e = \sum_{i=0}^{k} \mathbf{1}_{\{e \in \mathcal{H}(A_i)\}}$$

as an estimate for the pgain of A_0 with respect to a pool of chunks $\{A_0, \ldots, A_k\}$. Note that only the handprints are needed to compute this expression. Compression ratio is estimated similarly. The resulting estimates are quite accurate—in fact, almost 70% of the files' estimates are perfect. Figures 4 (left) and 5 (left) show that the estimated pgain using MR handprints is quite close to the actual pgain value. The compression ratio estimates (Figure 6 (left)) have even higher accuracy: across all chunk sizes, the worst 98th percentile error was only 0.04.

Using the optimal chunk size improves pgain and compression. For MR handprint-based systems, we estimate the pgain and compression ratio using MR handprints at different chunk sizes, and pick the largest chunk size that gives the maximum benefit. We compare MR handprint-based systems to systems that statically pick a constant chunk size for all their files.

Figures 4 and 5 show the results for the p2p and ISO/RPM workloads. The leftmost figures show that the handprints nearly always provide optimal pgain—higher than that achieved by a 16 KByte static chunk size—while the rightmost figures show that they do so while allowing a large fraction of the files to use a substantially *larger* chunk size. Thus, as shown in the right panel of these figures, using MR-handprints to estimate the optimal chunk size can permit p2p applications to download from a larger number of sources while simultaneously having lower overhead for 20–40% of the files (percent of files that MR-handprint picks a size larger than 16K).

The results for compression ratio estimation are similar: Figure 6 (right) shows the distribution of optimal chunk sizes for transferring or storing the ISO repository. Like the p2p chunk sizes, the MR-handprint selected chunks were sometimes substantially larger than a static 8 KByte chunk size often used for filesystem compression. As shown in Figure 6 (left), compared to static 8 KByte chunks, using the optimal chunk size added substantial compression: 20% of the files compressed by an additional 30% of their original size (i.e., they went from being 70% of their original size to being 40% of their original size).

In summary, the case study shows that multi-resolution handprints can be used to efficiently select a chunk size that is always better than a static chunk size. The largest benefit from doing so is in selecting a large chunk size when there is little benefit from similarity, and selecting a small chunk size when files are quite similar. As a result, we believe that MR-handprints provide an attractive technique for dynamic chunk size determination in systems exploiting data redundancy.

VI. Related Work

Estimating File Similarity The work most related to ours is seminal work in estimating similarity, starting from Manber's technique of *shingling* [12], Broder's resulting use of it for clustering web pages by similarity [3], and its subsequent use for duplicate elimination [8] by Douglis et al. Shingling runs a sliding-window hash along the contents of a file, outputting a hash value each time the low-order bits are zero. The technique is much like that we explore, except that we consider the

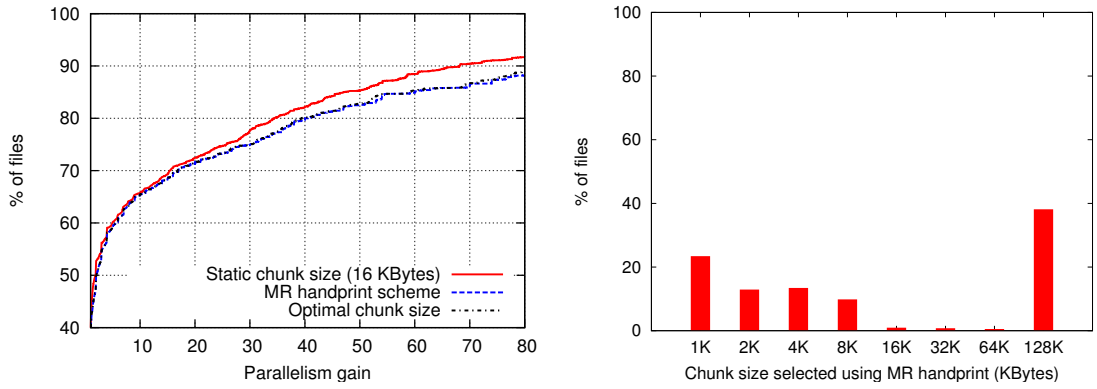


Fig. 4: Performance of MR-based scheme in a p2p file-transfer scenario (p2p data set); left: CDF of *pgain*, right: distribution of chunk sizes.

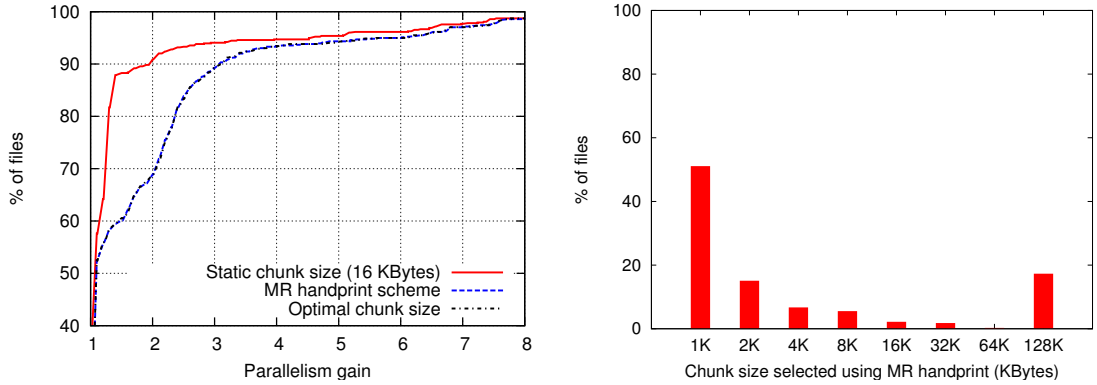


Fig. 5: Performance of MR-based scheme in a file-transfer scenario (ISO/RPM data set); left: CDF of *pgain*, right: distribution of chunk sizes.

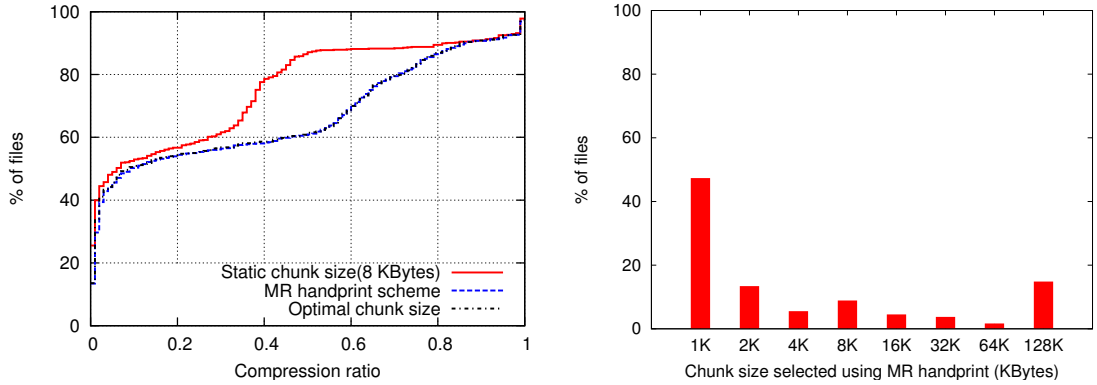


Fig. 6: Performance of MR-based scheme in a storage deduplication scenario; left: CDF of compression ratio, right: distribution of chunk sizes.

question of *exploitable* similarity, looking at entire chunks of data, whereas Shingling examined possibly overlapping chunk hashes to establish a more fine-grained estimate of similarity.

Exploiting Similarity to Speed Transfers Several systems exploit similarity to reduce the amount of data that needs to be transferred when the receiver already has a copy of a similar file. `rsync` [25] selects a candidate file at the receiver (typically with the same name as the file being transferred), and runs a hash through this file to identify shared chunks. Remote Differential Compression [23] and dsync [17] extend this idea to efficiently search many files on the remote filesystem, while czip [14] proposes a file format tailored for chunk-based search and transfer. These systems and others (such as the Low-Bandwidth filesystem [13], SET [16], and the Cooperative Filesystem [6]) typically pick statically sized chunks for their transfers, and can benefit from the techniques we propose.

Content-Based Naming The techniques we described should apply well to a large emerging class of systems that use content-based naming (naming data based upon its hash, and frequently based upon lists of chunks, each named by their hash). This technique forms the basis for many modern peer-to-peer systems

such as BitTorrent [4], as well as many filesystems [9, 6, 18, 5], and as a replacement for data transfer mechanisms [24].

Studies of Similarity Finally, we took some of our motivation from three studies that examined similarity across a wide range of files. Denehy and Hu found that 32% of data in an email corpus was duplicated [7]. Policroniades et al. made similar observations about compressed software distributions [15], and present a good summary of studies of chunk-level similarity across different workloads. (We note that this study, like many others, fixed the chunk size to 4 KBytes, while noting that many files in the collection were identical or nearly identical—exactly the observation we take advantage of in our work.) Kulkarni and Douglis et al. compare an additional large set of similar files [11]; they find 1 KByte chunk size to be a reasonable default but note that 4 KByte chunk size improves efficiency for large data sets that contain large files. Our work builds upon this observation for even larger types of files, creating techniques to build systems that can automatically determine the proper chunk size, without the need for human study or intervention.

VII. Conclusion

Many systems that exploit data redundancy statically choose a chunk size that is "good enough" for most of their data, due to lack of efficient techniques that identify optimal chunk size that results in maximum efficiency. Multi-resolution handprinting is an application-independent technique that provides an efficient and accurate way to determine the similarity of two data items over a wide range of chunk sizes, and, in turn, efficiently and accurately compute several metrics of interest to applications. We show through theoretical bounds and evaluation using a large p2p and software data set that MR-handprints are compact, requiring roughly 0.15% of the size of the file, but provide accurate similarity estimates (errors less than 5%). MR handprints can provide optimal chunk size selection for peer-to-peer, data transfer, and storage systems that permit these systems to exploit additional parallelism and compression, while reducing overhead by using large chunk sizes for files with little similarity. MR handprints thus represent a promising building block for chunk-based storage and transfer systems.

Acknowledgments. We would like to thank Anupam Gupta for inspiring discussions, and Stephanie Rosenthal and Donald Sheeny for their preliminary work on this project. This work was supported in part by NSF CAREER award CNS-0546551 and DARPA Grant HR0011-07-1-0025.

References

[1] S. Annapureddy, M. J. Freedman, and D. Mazières. Shark: Scaling file servers via cooperative caching. In *Proc. 2nd USENIX NSDI*, May 2005.

[2] A. Broder. On the resemblance and containment of documents. In *SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997*, page 21. IEEE Computer Society, 1997. ISBN 0-8186-8132-2.

[3] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the web. In *Proceedings of the 6th International WWW Conference*, pages 1157–1166, Apr. 1997.

[4] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[5] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. In *Proc. 5th USENIX OSDI*, Dec. 2002.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2001.

[7] T. E. Denehy and W. W. Hsu. Duplicate management for reference data. Research Report RJ10305, IBM, Oct. 2003.

[8] F. Douglis and A. Iyengar. Application-specific delta-encoding via resemblance detection. In *Proceedings of the USENIX Annual Technical Conference*, June 2003.

[9] *EMC Centera Content Addressed Storage System*. EMC Corporation, 2003. http://www.emc.com/.

[10] S. Janson. Large deviation inequalities for sums of indicator variables. Technical report, Uppsala University, 1994.

[11] P. Kulkarni, F. Douglis, J. LaVoie, and J. M. Tracey. Redundancy elimination within large collections of files. In *Proc. USENIX Annual Technical Conference*, June 2004.

[12] U. Manber. Finding similar files in a large file system. In *Proc. Winter USENIX Conference*, pages 1–10, Jan. 1994.

[13] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2001.

[14] K. Park, S. Ihm, M. Bowman, and V. S. Pai. Supporting practical content-addressable caching with CZIP compression. In *Proceedings of the USENIX Annual Technical Conference*, June 2007.

[15] C. Policroniades and I. Pratt. Alternatives for detecting redundancy in storage systems data. In *Proc. USENIX Annual Technical Conference*, June 2004.

[16] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. 4th USENIX NSDI*, Apr. 2007.

[17] H. Pucha, M. Kaminsky, D. G. Andersen, and M. A. Kozuch. Adaptive file transfers for diverse environments. In *Proc. USENIX Annual Technical Conference*, June 2008.

[18] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In *Proc. USENIX Conference on File and Storage Technologies (FAST)*, pages 89–101, Jan. 2002.

[19] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[20] S. C. Rhea, K. Liang, and E. Brewer. Value-based web caching. In *Proc. Twelfth International World Wide Web Conference*, May 2003.

[21] Riverbed. Riverbed. `http://www.riverbed.com/`.

[22] N. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proc. ACM SIGCOMM*, Sept. 2000.

[23] D. Teodosiu, N. Bjorner, Y. Gurevich, M. Manasse, and J. Porkka. Optimizing file replication over limited bandwidth networks using remote differential compression. Technical Report MSR-TR-2006-157, Microsoft Research, Nov. 2006.

[24] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil. An architecture for Internet data transfer. In *Proc. 3rd Symposium on Networked Systems Design and Implementation (NSDI)*, May 2006.

[25] A. Tridgell and P. Mackerras. The rsync algorithm. Technical Report TR-CS-96-05, Department of Computer Science, The Australian National University, Canberra, Australia, 1996.