Redundancy Elimination Within Large Collections of Files

Purushottam Kulkarni
University of Massachusetts
Amherst, MA 01003
purukulk@cs.umass.edu

Fred Douglis Jason LaVoie John M. Tracey
IBM T. J. Watson Research Center
Hawthorne, NY 10532
{fdouglis,lavoie,traceyj}@us.ibm.com

Abstract

Ongoing advancements in technology lead to ever-increasing storage capacities. In spite of this, optimizing storage usage can still provide rich dividends. Several techniques based on delta-encoding and duplicate block suppression have been shown to reduce storage overheads, with varying requirements for resources such as computation and memory. We propose a new scheme for storage reduction that reduces data sizes with an effectiveness comparable to the more expensive techniques, but at a cost comparable to the faster but less effective ones. The scheme, called *Redundancy Elimination at the Block Level* (REBL), leverages the benefits of compression, duplicate block suppression, and delta-encoding to eliminate a broad spectrum of redundant data in a scalable and efficient manner. REBL generally encodes more compactly than compression (up to a factor of 14) and a combination of compression and duplicate suppression (up to a factor of 6.7). REBL also encodes similarly to a technique based on delta-encoding, reducing overall space significantly in one case. Furthermore, REBL uses *super-fingerprints*, a technique that reduces the data needed to identify similar blocks while dramatically reducing the computational requirements of matching the blocks: it turns $O(n^2)$ comparisons into hash table lookups. As a result, using super-fingerprints to avoid enumerating matching data objects decreases computation in the resemblance detection phase of REBL by up to a couple orders of magnitude.

1 Introduction

Despite ever-increasing capacities, significant benefits can still be realized by reducing the number of bytes needed to represent an object when it is stored or sent. The benefits can be especially great for mobile devices with limited storage or bandwidth; reference data (data that are saved permanently and accessed infrequently); e-mail, in which large byte sequences are commonly repeated; and data transferred over low-bandwidth or congested links.

Reducing bytes generally equates to eliminating unneeded data, and there are numerous techniques for reducing redundancy when objects are stored or sent. The most longstanding example is *data compression* [12], which eliminates redundancy internal to an object and generally reduces textual data by factors of two to six. *Duplicate suppression* eliminates redundancy caused by identical objects which can be detected efficiently by comparing hashes of the objects' content [3]. *Delta-encoding* eliminates redundancy of one object relative to another, often an earlier version of the object by the same name [15]. Delta-encoding can in some cases eliminate an object almost entirely, but the availability of base versions against which to compute a delta can be problematic.

Recently, much work has been performed on applying these techniques to pieces of individual objects. This includes suppressing duplicate pieces of files [7, 8, 17, 20] and web pages [22]. Delta-encoding has also been extended to pairs of files that do not share an explicit versioning relationship [6, 9, 18]. There are also approaches that combine multiple techniques; for instance, the *vcdiff* program not only encodes differences between a "version" file and a "reference" file, it compresses redundancy within the version file [11]. Delta-encoding that simultaneously compresses is sometimes called "delta compression" [1].

In fact, no single technique can be expected to work best across a wide variety of data sets. There are numerous trade-offs between the *effectiveness* of data reduction and the resources required to achieve it, i.e. its *efficiency*. The relative importance of these metrics, effectiveness versus efficiency, depends on the environment in which techniques are applied. Execution time for example, which is an important aspect of efficiency, tends to be more important in interactive contexts than in asynchronous ones. In this paper, we describe a new data reduction technique that achieves comparable effectiveness to current delta-encoding techniques but with greater efficiency. It simultaneously offers better effectiveness than current duplicate suppression techniques at moderately higher cost.

We argue that performing comparisons at the granularity of files can miss opportunities for redundancy elimination, as can techniques that rely on large contiguous pieces of files to be identical. (Henson's study of issues relating to comparing blocks by hashes of their content made a similar argument [10].) Instead, we consider

what happens if some of the above techniques are further combined. Specifically, we describe a system that supports the union of three techniques: compression, elimination of identical content-defined chunks, and delta-compression of similar chunks. We refer to this technique as Redundancy Elimination at the Block Level (REBL). The key insight of this work is the ability to achieve more effective data reduction by exploiting relationships among similar blocks, rather than only among identical blocks, while keeping computational and memory overheads comparable to techniques that perform redundancy detection with coarser granularity.

We compare our new approach with a number of baseline techniques, which are summarized here and described in detail in the next section:

Whole-file compression. With whole-file compression (WFC), each file is compressed individually. This approach gains no benefit from redundancy across files, but it scales well with large numbers of files and is applicable to storage and transfer of individual files.

Compressed tar. Joining a collection of files into a single object which is then compressed has the potential to detect redundancy both within and across files. This approach tends to find redundancy across files only when the files are relatively close to one another in the object. We abbreviate this technique TGZ, for `tar+gzip`, the combination we used.

Block-level duplicate detection. There are a number of approaches to identifying identical pieces of data across files more generally. These include using fixed-size blocks [20], fixed-size blocks with rolling checksums [8, 23, 29], and content-defined (and therefore variable-sized) chunks [7, 17].

Delta-encoding using resemblance detection. Resemblance detection techniques [4] can be used to find similar files, with delta compression used to encode them effectively [9, 18].

There are also cases where effectiveness is dramatically improved by combining multiple techniques, such as adding compression to block-level or chunk-level duplication detection.

The remainder of this paper is organized as follows: Section 2 describes current techniques and their limitations. Details of the REBL technique are presented in Section 3. Section 4 describes the data sets and methodology used to evaluate REBL. Section 5 presents an empirical evaluation of REBL and several other techniques. Section 6 concludes.

2 Background and Related Work

We discuss current techniques in Section 2.1 and elaborate on their limitations in Section 2.2.

2.1 Current Techniques

A common approach to storing a collection of files compactly is to combine the files into a single object, which is then compressed on-the-fly. In Windows™, this function is served by the family of *zip* programs, though they do not necessarily identify inter-file redundancy in addition to intra-file redundancy. In UNIX™, files can be combined using *tar* with the output compressed using *gzip* or another compression program. However, TGZ does not scale well to extremely large file sets. Access to a single file in the set can potentially require the entire collection to be uncompressed. Furthermore, traditional compression algorithms maintain a limited amount of state information. This can cause them to miss redundancy between sections of an object that are distant from one another thus reducing their effectiveness. Historically the window used to detect redundancy is small, for instance 32 KB, but at least one new compression program uses memory-mapped I/O and increased state to find repeated substrings across hundreds of MB [24].

There are two general methods for compressing a collection of files with greater effectiveness and scalability than TGZ. One involves finding identical chunks of data within and across files. The other involves effective encoding of differences between files.

2.1.1 Duplicate Elimination

Finding identical files in a self-contained collection is straightforward through the use of strong hashes of their content. For example, Bolosky et al. have described a system to save only one instance of duplicate files in a Windows file system [3]. Mogul et al. have described a method for computing checksums over web resources (HTML pages, images, etc.) and eliminating retrieval of identical resources, even when accessed via different URIs [16].

Suppressing redundancy within a file is also important. One simple approach is to divide the file into fixed-length blocks and compute a checksum for each. Identical blocks are detected by searching for repeated checksums. The checksum algorithm must be "strong" enough to decrease the probability of a collision to an negligible value. SHA-1 [26] ("SHA") is commonly used for this purpose.

Venti [20] is a network-based storage system intended primarily for archival purposes. Each stored file is broken into fixed-sized blocks, which are represented by their SHA hashes. As files are incrementally stored, duplicate blocks, indicated by identical SHA values, are stored only once. Each file is represented as a list of SHA hashes which index blocks in the storage system.

Another algorithm used to minimize the bandwidth required to propagate updates to a file is *rsync* [23, 29]. With *rsync*, the receiver divides its out-of-date copy of

a file into fixed-length blocks, calculates two checksums for each block (a weak 32-bit checksum and a strong 128-bit MD4 checksum), and transmits the checksums to the sender to inform it which blocks the receiver possesses. The sender calculates a 32-bit checksum along a fixed-length window that it slides throughout the file to be sent. If the 32-bit checksum matches a value sent by the receiver, the sender confirms that the receiver already possesses the corresponding block by computing and comparing the 128-bit checksum. Use of a rolling checksum over fixed-sized blocks has recently been extended to large collections of files regardless of name [8]. We refer to this as the SLIDINGBLOCK approach, which is one of the techniques against which we compare REBL below.

One can maintain a large replicated collection of files in a distributed environment using a technique similar to SLIDINGBLOCK [27]. Suel et al. point out two main parameters for *rsync* performance: block size and location of changes within the file. To enhance the performance of *rsync*, they propose a multi-phase protocol in which the server sends hashes to the client and client returns a bitmap indicating the blocks it already has (similar to *rsync*). In this approach, the server uses the bitmap of existing blocks to create a set of reference blocks used to encode all blocks not present at the client. The delta sent to the client by the server is used in conjunction with blocks in the bitmap to recreate the original file. This technique has some similarity to Spring and Wetherall's approach to finding redundant data on a network link by caching "interesting" fingerprints of sliding windows of data and then finding the fingerprints in a cache of past transmissions [25].

The Low-Bandwidth File System (LBFS) [17] is a network file system designed to minimize bandwidth consumption. LBFS divides files into *content-defined chunks* using Rabin fingerprints [21]. Fingerprints are computed over a sliding window; a subset of possible fingerprint values denotes chunk boundaries, with the subset determining a probabilistic average chunk size. LBFS also imposes a minimum and maximum chunk size, irrespective of fingerprint values.

LBFS computes and stores an SHA hash for each content-defined chunk stored in a given file system. Before a file is transmitted, the SHA values of each chunk in the file are sent first. The receiver looks up each hash value in a database of values for all chunks it possesses. Only chunks not already available at the receiver are sent; chunks that are sent are compressed. Content-defined chunks have also been used used in the web [22] and backup systems [7]. We refer to the overall technique of eliminating duplicate content-defined chunks and compressing remaining chunks as CDC, and we compare REBL with this combined technique in the evaluation section below.

2.1.2 Delta-encoding and File Resemblance

A second general approach to compressing multiple data objects is delta-encoding. This approach has been used in many applications including source control [28], backup [1], and web retrieval [14, 15]. Delta-encoding has also been used on web pages identified by the similarity of their URIs [6].

Effective delta-encoding relies on the ability to detect similar files. Name-based file pairing works only in very limited cases. With large file sets, the best way to detect similar files is to examine the file contents. Manber [13] discusses a basic approach to finding similar files in a large collection of files. His technique summarizes each file by computing a set of polynomial-based fingerprints; the similarity between two files is proportional to the fraction of fingerprints common between them. Rabin fingerprints have been used for this purpose in numerous studies. Broder developed a similar approach [4], which he extended with an heuristic to coalesce multiple fingerprints into *super-fingerprints*. A single matching super-fingerprint implies high similarity, allowing similarity detection to scale to very large file sets such as web search engines [5].

While these techniques allow similar files to be identified, only recently have they been combined with delta-encoding to save space. Douglis and Iyengar describe "Delta-Encoding via Resemblance Detection" (DERD), a system that uses Rabin fingerprints and delta-encoding to compress similar files [9]. The similarity of files is based on a subset of all fingerprints generated for each file. Ouyang et al. also study the use of delta compression to store a collection of files [18]; their approach to scalability is discussed in the next subsection.

2.2 Limitations of Current Techniques

Duplicate elimination exploits only files, blocks or chunks that are exactly the same. Thus, a version of a file that has many minor changes scattered throughout sees no benefit from the CDC or SLIDINGBLOCK techniques. Section 5.1 includes graphs of the overlap of fingerprints in CDC chunks that indicate how common this issue can be.

DERD uses delta-encoding, which eliminates redundancy at fine granularity when similar files can be identified. Resemblance detection using Rabin fingerprints is more efficient than the brute force approach of delta-encoding every possible pair of files. A straightforward approach to identifying similar files is to count the number of files that share even a single fingerprint with a given file. However, repeating this for every fingerprint of every file results in an algorithm with O(n^2) complex-

ity in the worst case, where n is the number of files.[1] For large file sets, run time is dominated by the number of pairwise comparisons and can grow quite large even if the time for each comparison is small. DERD's performance therefore does not scale well with large data sets.

The computational complexity of delta-encoding file sets motivates cluster-based delta compression [18]. With this approach, large file sets are first divided into clusters. The intent is to group files expected to bear some resemblance. This can be achieved by grouping files according to a variety of criteria including names, sizes, or fingerprints. (Douglis and Iyengar used name-based clusters to make the processing of a large file set tractable in terms of memory consumption [9]; similar benefits apply to processing time.) Once files are clustered, the techniques described above can be used to identify good candidate pairs for delta-encoding within each cluster. Clustering reduces the size of any file set to which the O(n^2) algorithm described above is applied. When applied over all clusters, the technique results in an approximation to the optimal delta-encoding. How close this approximation is depends on the amount of overlap across clusters and is therefore extremely data-dependent.

Another important issue is that DERD does not detect matches between an encoded object and pieces of multiple other objects. Consider for example, an object A that consists of the concatenation of objects B-Z. Each object B-Z could be encoded as a byte range within A, but DERD would likely not detect any of the objects B-Z as being similar to A. This is due, in part, to the decision to represent each file by a fixed number of fingerprints regardless of file size. Because Rabin fingerprint values are uniformly distributed, the probability of a small file's fingerprints intersecting a large containing file's fingerprints is proportional to the ratio of their sizes. In the case of 25 files contained within a single 26th file, if the 25 files are of equal size but contain different data, each will contribute about $\frac{1}{25}$ of the fingerprints in the container. This makes detection of overlap unlikely.

The problem arises because of the distinction between resemblance and containment. Broder's definition of *containment* of B in A is the ratio of the intersection of the fingerprints of the two files to the fingerprints in B, i.e. $\frac{F(A) \cap F(B)}{F(B)}$ [4]. When the number of fingerprints for a file is fixed regardless of size, the estimator of this intersection no longer approximates the full set. On the other hand, deciding there is a strong resemblance between the two is reasonably accurate, because for two documents to resemble each other, they need to be of similar size.

Finally, it is possible that extremely large data sets would not lend themselves to "compare-by-hash" because of the prospect of an undetected collision [10]: hashes can be collision-resistant but there will be collisions given enough inputs. In a system that is deciding whether two local objects are identical, a hash can be used to find the two objects before expending the additional effort to compare the two objects explicitly. Our data sets are not of sufficient scale for that to pose a likely problem, so we did not include this extra step. While we chose to follow the protocols of past systems, explicit comparisons could easily be added.

3 REBL Overview

We have designed and implemented a new technique that applies aspects of several others in a novel way to attain benefits in both effectiveness and efficiency. This technique, called Redundancy Elimination at the Block Level (REBL), includes features of compression, CDC, and DERD. It divides objects into content-defined chunks, which are identified using SHA hashes. First duplicate chunks are removed, and then resemblance detection is performed on each remaining chunk to identify chunks with sufficient redundancy to benefit from delta-encoding. Chunks not handled by either duplicate elimination or resemblance detection are simply compressed. A more detailed description of REBL appears in Section 4.1.

Key to REBL's ability to achieve efficiency comparable to CDC, instead of suffering the scalability problems of DERD, are optimizations that allow resemblance detection to be used more effectively on chunks rather than whole files. Resemblance detection has been optimized for use in Internet search engines to detect nearly identical files. The optimization consists of summarizing a set of fingerprints into a smaller set of *super-fingerprints*, possibly a single super-fingerprint. Objects that share even a single super-fingerprint are extremely likely to be nearly identical [5].

Optimized resemblance detection works well for Internet search engines where the goal is to detect documents that are nearly identical. Detecting objects that are merely similar enough to benefit from delta-encoding is harder. We hypothesized that applying super-fingerprints to full files in DERD would significantly improve the time needed to identify similar files but would also dramatically reduce the number of files deemed similar, resulting in lower savings than the brute force technique that counts individual matching fingerprints [9]. In practice, we found using the super-fingerprint technique with whole files works better than we anticipated, but it is still not the most effective approach (see Section 5.1.4 for details).

In contrast, REBL can benefit from the optimized resemblance detection because it divides files into chunks and looks for near duplicates of each chunk separately.

Data set	Size (MB)	# files	# chunks	
			1 KB	4 KB
Slashdot	38.37	885	21,991	11,629
Yahoo	27.77	3,850	28,542	8,632
Emacs	106.60	5,490	70,640	24,960
MH	602.10	93,867	421,501	203,518
Users	6625.43	185,722	3,949,780	1,367,619

Table 1: Details of data sets used in our experiments. 1 KB and 4 KB are the targeted average chunk sizes. For 1-KB averages, the minimum chunk size is set to 512 bytes; for 4-KB averages, it is set to 1 KB. The maximum is 64 KB.

This technique can potentially sacrifice some "marginal" deltas that would save some space. We quantify this sacrifice by comparing the super-fingerprint approach with the DERD technique that enumerates the best matches from exact fingerprints.

3.1 Parameterizing REBL

REBL's performance depends on several important parameters. We describe the parameters and their default values here and provide a sensitivity analysis in Section 5.

Average chunk size. Smaller chunks detect more duplicates but compress less effectively; they require additional overhead to track one hash value and numerous fingerprints per chunk; and they increase the number of comparisons and delta-encodings performed. We found 1 KB to be a reasonable default, though 4 KB improves efficiency for large data sets with large files. Throughout this paper, references to REBL with a specific chunk size refer to a probabilistic average chunk size.

Number of fingerprints per chunk. The more fingerprints associated with a chunk, the more accurate the resemblance detection but the higher the storage and computational costs. Douglis and Iyengar used 30 fingerprints, finding no substantial difference from increasing that to 100 [9].

Number of fingerprints per super-fingerprint. With super-fingerprints, a given number of base fingerprints are distilled into a smaller number of super-fingerprints. We use 84 fingerprints, which are grouped into 3-42 super-fingerprints.

Similarity threshold. How many fingerprints (or super-fingerprints) must match to declare two chunks similar? If the threshold is fixed, how important is it to find the "best" match rather than any adequate match? Ouyang et al. addressed this by finding adequate matches rather than best matches [18]; Douglis and Iyengar did a more computationally expensive but more precise determination [9]. We take a middle ground by approximating the "best" match more efficiently via super-fingerprints. A key result of our work is that using a sufficiently large number of fingerprints per super-fingerprint allows any matching chunk to be used rather than having to search for a good match. This results in nearly the same effectiveness but with substantially better efficiency (see Section 5.1.2).

Base minimization. Using the best base against which to delta-encode a chunk can result in half the chunks serving as reference blocks.[2] Allowing approximate matches can substantially increase the number of version blocks encoded against a single reference block, thereby improving overall effectiveness (see Section 5.3.2).

Shingle size. Rabin fingerprints are computed over a sliding window called a shingle and used for two purposes. First, CDC uses specific values of Rabin fingerprints to denote a chunk boundary. Second, DERD uses them to associate features with each chunk. A shingle should be large enough to generate many possible substrings, which minimizes spurious matches, but it should be small enough to keep small changes from affecting many shingles. Common values in past studies like DERD have ranged from four to twenty bytes [9, 18]. We used a default of twelve bytes but found no consistent trend other than a negative effect from sizes of four or eight bytes in one of the data sets (see Section 5.3.3).

4 Data Sets and Methodology

We used several data sets to test REBL's effectiveness and efficiency. Table 1 lists the different data sets, giving their size, the number of files, and the number of content-defined chunks with the targeted average chunk-size set to 1 KB and 4 KB.

The `Slashdot` and `Yahoo` data sets are web pages that were downloaded and saved, as a system such as the Internet Archive [2] might archive web pages. `Slashdot` represents multiple pages downloaded over a period of about a day, wherein different pages tend to have numerous small changes corresponding mostly to updated counts of user comments. (While the Internet Archive would not currently save pages with such granularity, an archival system might if it could do so efficiently.) As the smallest data set, `Slashdot` is used below in several cases where there are large numbers of experiments with varying parameters to keep the total execution time within reason. `Yahoo` represents a number of different pages downloaded recursively at a single point in time. `Emacs` contains the source trees for GNU Emacs 20.6 and 20.7. The `MH` data set refers to individ-

ual files consisting of entire e-mail messages. Finally, `Users` is the contents of one partition of a shared storage system within IBM, containing data from 33 users totaling nearly 7 GB.

REBL is intended for much larger data sets than the ones presented here. However, the analysis was implemented using in-memory data structures based on the GNU C++ Standard Template Library, and as a result the application's virtual address space places limits on the metadata (particularly matching pairs of fingerprints) kept in memory. The results presented below demonstrate the scalability issues mentioned previously, and lead us to believe that one can extrapolate to larger data sets once out-of-memory data structures are used. Furthermore, one can vary parameters such as block size and the number of super-fingerprints per block to keep the meta-data requirements low. In particular, the `Users` data set is an order of magnitude larger than the next-largest data set, containing many large files, so the REBL analysis of it is done with an average chunk size of 4 KB rather than 1 KB.

4.1 REBL Evaluation

To evaluate REBL, we first read each file in the data set sequentially, break it into content-defined chunks and generate the Rabin fingerprints and SHA hash values for each chunk. The hashes and fingerprints are stored in Berkeley DB format; each chunk has a file name, length, offset, and fingerprints associated with it. In this stage, chunks with the same SHA hash value as earlier chunks require no additional processing, because they are suppressed by the CDC duplicate detection mechanism.

A separate application reads the fingerprints and chunk information to perform REBL or DERD analysis. First, it computes super-fingerprints from the fingerprints, given a specific ratio of fingerprints per super-fingerprint (with a ratio of one, this step would be skipped). At this point, we have the option of *FirstFit* or *BestFit*.

- To do *FirstFit*, we pick a candidate reference chunk and encode all other chunks that share a super-fingerprint with it; an associative array makes pairwise matching efficient. We then iterate over the remaining candidate reference chunks, performing the same operation, ignoring any chunks that have already been encoded or used as a reference. The chunks are analyzed in order of insertion into the system; i.e., the first file read by the system will be broken into one or more chunks, each of which is likely to serve as a reference version for other chunks with many fingerprints in common, and chunks from later files will be increasing likely to have already been encoded.
- To do *BestFit*, we sort the chunks according to the greatest number of matching fingerprints with any other chunk. Each candidate reference chunk is then processed to determine which other potential version chunks have at least a threshold number of super-fingerprints in common with it. The threshold is a specified fraction of the best match found for that chunk (see Section 5.3.2). Again, each chunk is used as either a reference against which one or more version chunks are encoded, or as a version. *BestFit* suffers from quadratic complexity in processing time, as a function of the number of chunks, as well as substantially greater memory usage [3] than *FirstFit*.

Next we compress any remaining chunks that have not already been delta-encoded, including the reference chunks.

Finally, each of the files in the data set is compressed to determine if compressing the entire file is more effective than eliminating duplicate blocks and delta-encoding similar blocks. If so, the WFC size is used instead. We found that CDC in the absence of WFC was much less effective than the combination of the two, but adding WFC to REBL usually made only a small difference because most chunks already benefitted from delta-encoding (see Section 5.2).

One technique we did not explicitly evaluate is duplication detection using fixed-size blocks, like that performed by Venti [20]: we worked under the assumption that CDC would be preferable to fixed-size blocks. Other studies that compare the two techniques head-to-head have found that CDC frequently compresses better, at the cost of increased computation [8, 19].

5 Empirical Results

This section provides an empirical evaluation of several data reduction techniques with an emphasis on REBL. For REBL, we study the effect of parameters such as the average chunk size, the number of super-fingerprints, the similarity threshold above which delta-encoding is applied, and the shingle size. The techniques are compared along primarily two-dimensions: *effectiveness* (space savings) and *efficiency* (run time).

The techniques evaluated in at least one scenario include:

- TGZ
- whole-file compression (WFC)
- per-block compression (PBC)
- CDC (includes PBC)
- CDC with WFC
- SLIDINGBLOCK
- REBL with WFC
- DERD with WFC

In cases where WFC is used in conjunction with another

technique, this means that WFC is used instead of the other technique if it is found to be more effective on a particular file.

Our experiments were performed on an unmodified RedHat Linux 2.4.18-10 kernel running on an IBM eServer xSeries 360 with dual 1.60 GHz Pentium Xeon processors, 6 GB RAM (2×2 GB plus 2×1 GB), and three 36 GB 10k-RPM SCSI disks connected to an IBM Netfinity ServeRAID™ 5 controller. All data sets resided in an untuned ext3 file system on local disks. Although an SMP kernel was used, our tests were not optimized to utilize both processors. All times reported are the sums of user and system time as reported by *getrusage*.

5.1 REBL Hypotheses

This section presents empirical results to support the rationale behind the combination of chunk-level delta-encoding and super-fingerprints.

5.1.1 Chunk Similarity

As discussed in Section 2.1, CDC systems such as LBFS [17] compute SHA hashes of content-defined chunks and use the hash value to detect duplicates. A potential limitation of this approach is that chunks with slight differences get no benefit from the overlap. For REBL to be more effective than CDC, there must be a substantial number of chunks that are similar but not identical.

Figure 1 plots a cumulative distribution of the fraction of chunks that match another chunk in a given number of fingerprints. The graph shows results for the `Slashdot` and `Yahoo` data sets with 84 fingerprints per chunk and shows curves corresponding to average chunk sizes of 1 KB, 4 KB, and whole files. Whole files correspond to an infinitely large average chunk size, which is similar to DERD. All chunks match another chunk in at least 0 fingerprints, so each curve meets the 0 value on the x-axis at $y = 1$. The rightmost points on the graph (depicted as x=85) show the fraction of chunks that are duplicated; smaller chunks lead to greater effectiveness for CDC because they allow duplicate content to be detected with finer granularity. Between these extremes, more of the smallest chunks match other chunks in the greatest number of features. However, any chunks that are not exact duplicates but match many fingerprints are "missed" by CDC, but they are potentially usable by REBL for delta-encoding and result in improved space savings. A good heuristic for expecting improvement from delta-encoding is to match at least half the fingerprints [9].

5.1.2 Benefits of Super-fingerprints

Next we look at the use of a smaller number of super-fingerprints to approximate a larger number of fingerprints. As discussed in Section 3, super-fingerprints are generated by combining fingerprints. For instance, 84 fingerprints can be clustered into groups of six to form 14 super-fingerprints. To generate super-fingerprints, REBL concatenates fingerprints and calculates the corresponding MD5 value of the resulting string. [4]

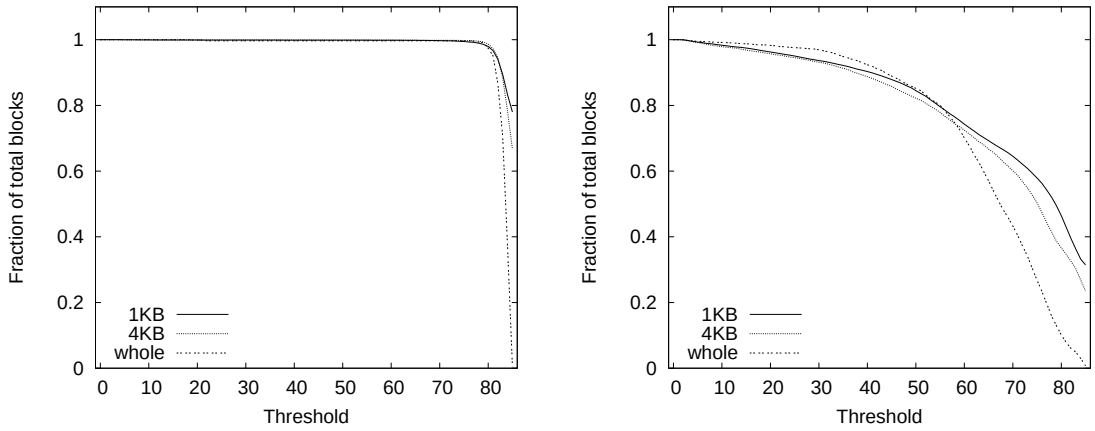
Note that the clustering of fingerprints into super-fingerprints is necessary to make the *FirstFit* variant viable. One could delta-encode any pair of chunks that matched a single fingerprint, but unless the shingle size is quite high, there is little assurance of commonality between the chunks. On the other hand, if two chunks share a specific set of fingerprints, the larger the set, the greater the likelihood of a significant overlap [5].

Figure 2 plots the cumulative distribution of the number of chunks that match another in at least a threshold fraction of fingerprints or super-fingerprints. The data sets used are `Slashdot` and `MH`, with 84 fingerprints, 1-KB average chunks, and 21, 14, 6 and 4 super-fingerprints per chunk. (The curve for 84 fingerprints on the `Slashdot` data set corresponds to the 1-KB curve for `Slashdot` in the preceding figure.) The results indicate that lowering the threshold for similarity between chunks results in more chunks being considered "similar." The results for super-fingerprints follow a similar trend as for regular fingerprints. A useful observation from both data sets is that we can select a threshold value for super-fingerprints that corresponds to a higher number of matching fingerprints. For example, in Figure 2(a), a threshold of one out of four (25%) super-fingerprints is approximately equivalent to a threshold of 73 out of 84 (87%) fingerprints. Using super-fingerprints therefore decreases REBL's execution time by reducing the number of comparisons, as the next subsection describes.

5.1.3 FirstFit and BestFit Variants

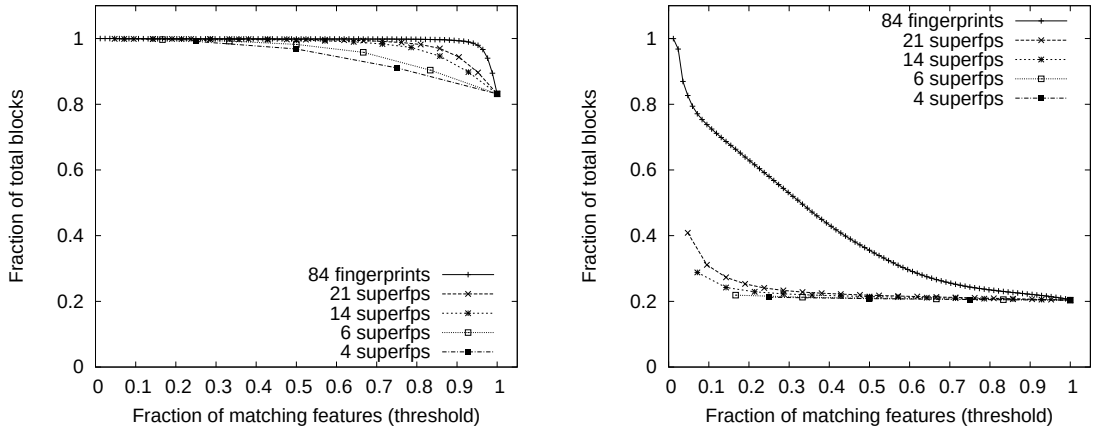
As discussed in Section 4.1, REBL has two variants, *BestFit* and *FirstFit*. In this section, we compare them by contrasting relative effectiveness and efficiency.

Figure 3(a) plots the sizes of the `MH` and `Slashdot` data sets, encoded using the *FirstFit* and *BestFit* variants, and reported relative to the original data set sizes. The experiment uses 84 fingerprints per chunk and two chunking methods, whole files (DERD) and an average chunk size of 1 KB. In this subsection we consider only the 1-KB chunks, discussing later how this compares to other sizes. The figure demonstrates the effect of varying the number of super-fingerprints from three to 84 (with 84 meaning there is no clustering). Both *FirstFit* and *BestFit* have comparable encoding sizes for up



(a) `Slashdot` (b) `Yahoo`

Figure 1: Cumulative distribution of the fraction of similar chunks or files with at least a given number of maximally matching fingerprints. The right-most point in each graph corresponds to identical chunks or files.



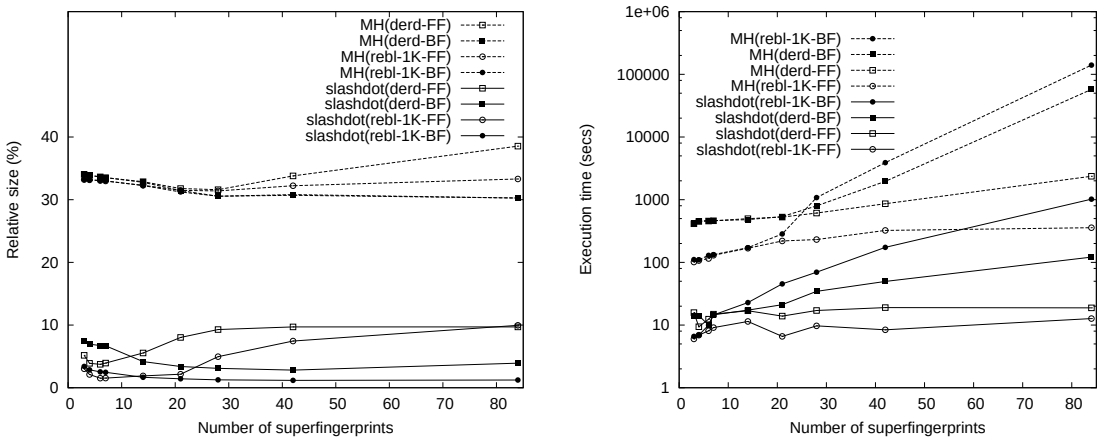
(a) `Slashdot` (b) `MH`

Figure 2: Cumulative distribution of matching fingerprints or super-fingerprints, using 1-KB chunks. The relative shape of the curves demonstrate the much greater similarity in the `Slashdot` data set than the `MH` data set.

to a number of super-fingerprints (21 for these two data sets). After this point, *BestFit* encodes the most effectively because taking the first fit with too few fingerprints per cluster is a poor predictor of a match.

Figure 3(b) plots the corresponding execution times. As we increase the number of super-fingerprints, the number of comparisons for *BestFit* to detect similar chunks increases, leading to dramatically greater execution times. The execution times using *FirstFit* are more or less stable and do not show the same sharp increase. In short, using *FirstFit* allows a little space to be sacrificed in exchange for dramatically lower execution times. For example, with `Slashdot` using *FirstFit*, 1-KB chunks, and six super-fingerprints, REBL produces a relative size of 1.52%; the best *BestFit* number is 1.18% with 42 super-fingerprints. However, the corresponding absolute execution times are 8.1 and 173.5 seconds respectively.

As mentioned in Section 3.1, one interesting parameter that can be modified when using *BestFit* is



(a) `Relative size` (b) `Execution time (logscale)`

Figure 3: Relative size and total execution time as a function of the number of super-fingerprints, for two data sets, using DERD and REBL.

its eagerness to use the most similar reference chunk against which to delta-encode a version chunk. For instance, one might naturally assume that encoding a chunk against one that matches it in 80/84 fingerprints would be preferable to encoding it against another that matches only 70/84. However, consider a case where chunk A matches chunk B in 82 fingerprints, chunk C in 75, and chunk D in 70; C and D resemble each other in 80/84. Encoding A against B and C against D generates two small deltas and two reference chunks, but encoding B, C, and D against A results in slightly larger deltas but only one unencoded chunk. As a result of this eagerness, *FirstFit* often surprisingly encoded better than *BestFit* until we added an approximation metric to *BestFit*, which lets a given chunk be encoded against a specific reference chunk if the latter chunk is within a factor of the best matching chunk. Empirically, allowing matches within 80-90% of the best match improved overall effectiveness, as shown in the sensitivity analysis in Section 5.3.2.

5.1.4 Benefits of Chunking

While REBL applies super-fingerprints to content-defined chunks, super-fingerprints could also be applied to entire files, similar to detecting commonality in web pages [5]. This would amount to a modification of the DERD approach, optimizing the resemblance detection step, but applying them to entire files can potentially reduce the number of files identified as being similar.

Referring again to Figure 3(a), but this time considering the curves for DERD as well as 1-KB chunks, we see that REBL is always at least as effective as DERD for both *FirstFit* and *BestFit*. The curves for 4-KB chunks are omitted in order to keep the graphs readable, but generally follow the 1-KB chunk curves with slightly more space consumed.

The corresponding execution times for DERD and REBL are plotted in Figure 3(b). As expected, the greatest execution time is for REBL with *BestFit*; breaking each file into chunks results in more comparisons. Execution times for *BestFit* increase sharply with increasing numbers of super-fingerprints. The best overall results considering both effectiveness and efficiency are with the *FirstFit* variant of REBL using a small number of super-fingerprints. Using 4-KB chunks (not shown) proves to be moderately faster than 1-KB chunks.

One might ask whether it is sufficient to simply use some number (N) of fingerprints rather than combining a larger number of fingerprints into the same number (N) of super-fingerprints. In fact, with *BestFit*, using as few as 14 fingerprints is nearly as effective as using 84 fingerprints. However, even with only 14 fingerprints, the execution cost of *BestFit* is substantially greater than *FirstFit*. Figure 4 reports relative sizes and execution times for the `Slashdot` data set as a function of the number of fingerprints or super-fingerprints, using an average chunk size of 4 KB. With super-fingerprints and *FirstFit*, relative size increases with more super-fingerprints, while with fingerprints and *BestFit* relative size decreases with more fingerprints. On the other hand, execution time with *BestFit* increases sharply with more fingerprints. The effectiveness with super-fingerprints using *FirstFit* is similar to that using a larger number of fingerprints and *BestFit*.

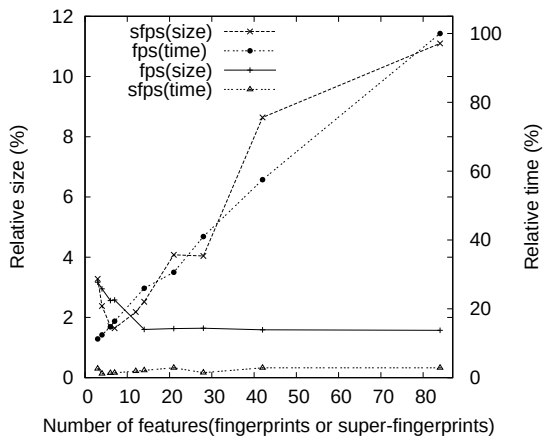


Figure 4: Comparing effect of reduced fingerprints per block and *FirstFit* with super-fingerprints using the `Slashdot` data set.

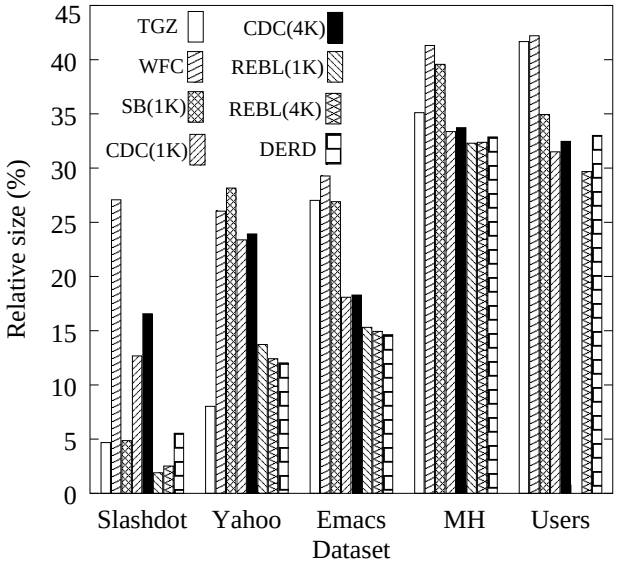


Figure 5: Effectiveness of different encoding techniques based on the relative sizes of the encoded data sets. There is no bar for 1-KB chunks for REBL on the `Users` data set, as this was not computed. SB refers to SLIDINGBLOCK.

To quantify these differences with the example in Figure 4, *FirstFit* with seven super-fingerprints has a relative size of 1.64%; the best effectiveness using fingerprints is 1.57% with 84 fingerprints per chunk. However, the former runs in just 1.4% of the time taken by the latter. The relative execution time with 14 fingerprints and *BestFit*, which gives comparable results to using 84 fingerprints, is 26.0%. Thus, lowering the number of fingerprints per chunk to reduce comparisons (and increase efficiency) may not yield the best encoding size and execution times. In contrast, the use of super-fingerprints and the *FirstFit* variant of REBL is both effective and efficient.

5.2 Comparison of Techniques

In this section, we compare a variety of techniques, focusing on effectiveness and briefly discussing efficiency as indicated by execution times. Table 2 reports sizes compared to the original data set. The relative sizes with CDC are reported for average chunk sizes of 1 KB (with and without WFC) and 4 KB (with WFC). REBL numbers use average chunk sizes of 1 KB (except the 7-GB `Users` set) and 4 KB, and they include both PBC and WFC.

For the experiments using these data sets, we strove for consistency whenever possible. However, there are some cases where varying a parameter or application made a huge difference. In particular, *gzip* produces output somewhat smaller than *vcdiff* for all our data sets except `Slashdot`, for which it is nearly an order of magnitude larger. We report the *vcdiff* number in that case only.

For both REBL and DERD, the table gives numbers for 14 super-fingerprints using *FirstFit*. Full comparisons of regular fingerprints generally gave a smaller encoding, but at a disproportionately high processing cost as the experiments above demonstrated. REBL had the smallest encoding size in three out of the five data sets above. REBL encoded more effectively than CDC, SLIDINGBLOCK, and WFC with all data sets; it was better than TGZ except with the `Yahoo` data set; and it was better than DERD except with the `Yahoo` and `Emacs` data sets.

Figure 5 graphically depicts the data in Table 2, and Figure 6 shows a scatterplot of how the other techniques compare to REBL. For consistency, we compare the encoding sizes of CDC (1-KB chunks) with REBL (1-KB chunks) and CDC (4-KB chunks) with REBL (4-KB chunks). `Users` is compared to REBL with 4-KB chunks throughout. Otherwise, REBL with 1-KB chunks is used as the baseline.

As with REBL using 1-KB chunks, REBL using 4-KB chunks is better than TGZ (except with the `Yahoo` data set), WFC, SLIDINGBLOCK, CDC with either 1-KB and 4-KB chunks and DERD except the `Yahoo` and `Emacs` data sets. The effectiveness of REBL compared to TGZ varied by factors of 0.59-2.46, WFC by 1.28-14.25, SLIDINGBLOCK by 1.18-2.56, CDC by 1.03-6.67 and DERD by 0.88-2.91.

Additionally, we compare the effectiveness of REBL with and without using WFC. The relative sizes of the `Slashdot`, `Yahoo`, `Emacs`, and `MH` data sets using REBL with 1-KB chunks and without WFC are 1.9%, 14.85%, 17.8% and 36.0% respectively. REBL without

Data set	TGZ	WFC	PBC (1 KB)	Sliding Block (1 KB)	CDC (1 KB w/o WFC)	CDC (1 KB w/ WFC)	CDC (4 KB w/ WFC)	REBL (1 KB 14 FF)	REBL (4 KB 14 FF)	DERD (14 FF)
Slashdot	4.68	27.08	44.46	4.86	12.74	12.68	16.56	**1.89**	2.52	5.54
Yahoo	**8.03**	26.02	40.67	28.16	29.18	23.38	23.93	13.72	12.42	12.03
Emacs	27.02	29.27	42.25	26.89	24.95	17.99	18.29	15.31	14.94	**14.64**
MH	35.11	41.30	48.23	39.57	38.10	33.36	33.73	**32.28**	32.38	32.87
Users	41.67	42.19	49.93	34.94	34.49	31.48	32.47	N/A	**29.69**	33.01

Table 2: Data sets and their relative encoding sizes (in percent) as compared to the original size using different encoding techniques. The best encoding for a data set is in **boldface**. TGZ stands for `tar` plus `gzip`, WFC is whole–file compression, and PBC is content-defined block-level compression. REBL uses 14 super-fingerprints and *FirstFit.*

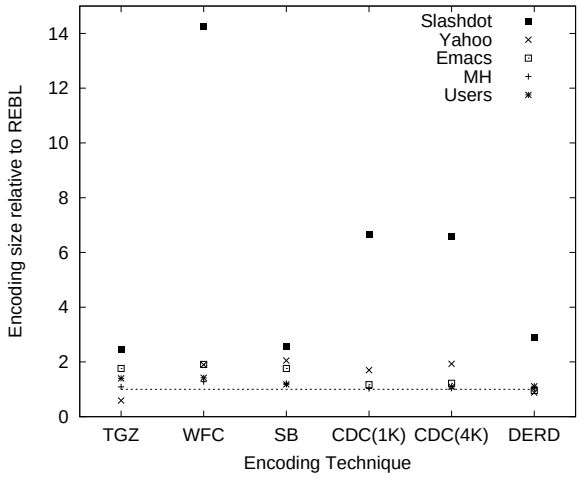


Figure 6: Effectiveness of different encoding techniques, grouped by technique, relative to REBL. REBL uses a 1-KB chunk size, except for `Users` (4-KB chunks in all comparisons) and CDC with 4-KB chunks (compared head-to-head with 4-KB REBL). The horizontal line indicates the break-even point; all points above this line reflect cases in which REBL is more effective. SB refers to SLIDINGBLOCK.

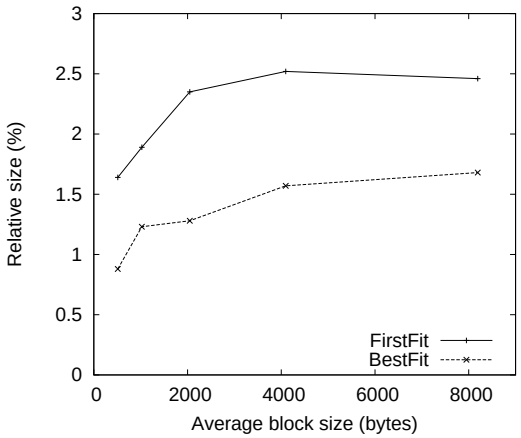


Figure 7: Effect of average block-size on relative size using REBL on the `Slashdot` data set.

WFC with 4-KB chunks using the `Users` data set has a relative size of 30.6%. The corresponding relative sizes of REBL with WFC are reported in Table 2. Without WFC, the relative sizes for two of the data sets (`Slashdot` and `Users`) are within 3% of the sizes that include WFC, but REBL without WFC encodes worse than REBL with WFC for the `Yahoo`, `Emacs`, `MH` and data sets by 7.5%, 16.2%, and 11.5% respectively.

An obvious question is how the execution time of REBL compares to the other approaches we have discussed. We have already compared REBL and DERD in some depth. Since REBL relies on CDC, it is inherently more costly than CDC, which in turn requires substantially more computation than a simple technique such as TGZ. How much more costly REBL is depends on how many deltas are computed and how much computation is performed to select chunks to delta. The *FirstFit* variant requires processing that scales linearly, rather than quadratically, with the input size, just as the CDC processing does. Hence the additional cost is comparable to CDC processing. The additional space savings varies across data sets; for `Slashdot` the additional savings seems easily warranted, while for `MH` it seems unlikely to be worthwhile.

5.3 Additional Considerations

This subsection describes the sensitivity of REBL to various execution parameters that try to optimize its behavior.

5.3.1 Effect of Average Chunk Size

One potentially important parameter for REBL is the average chunk size. As discussed in Section 5.1.1, smaller chunk sizes provide more opportunity to find similar chunks for delta-encoding. Figure 7 reports results of experiments with the `Slashdot` data set, 84 fingerprints per chunk, and varying the average chunk sizes from 512 to 8192 bytes. In the case of `Slashdot`, for

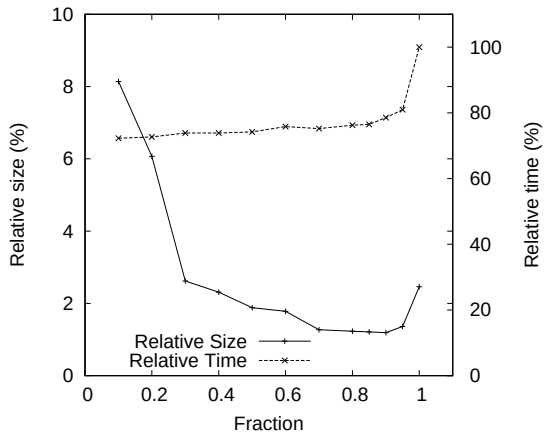


Figure 8: Effect of varying the *BestFit* threshold on relative size using REBL on the `Slashdot` data set.

both *FirstFit* and *BestFit*, increasing the average chunk size results in larger encoding sizes. The smallest relative size is obtained with the 512-byte chunk size in both variants of REBL.

The same experiment was performed with the `MH` data set using the *FirstFit* variant of REBL. In this case too, the smallest relative size (32.3%) was obtained using a chunk size of 512 bytes and other chunk sizes reported slightly larger sizes. However, there was minimal degradation in effectiveness moving to larger sizes (the maximum relative size was 32.4% with a chunk size of 8 KB).

Choosing a smaller chunk size provides more opportunities for delta-encoding and better space savings but at a higher run-time cost. For example, for the `MH` data set, 8-KB chunks save about 20% of the REBL post-CDC processing time, compared to 4-KB chunks, for almost identical encoding effectiveness.

5.3.2 Approximate Matching for *BestFit*

Another parameter we evaluated is the threshold for approximate matching of *BestFit* chunks. Without this threshold, using *BestFit*, a chunk is encoded against another with which it has the most matching fingerprints or super-fingerprints. For a given reference chunk, the "*BestFit* threshold" determines how loose this match can be, permitting the encoding of any chunks within the specified fraction of the best match. Note that in all cases, the system is counting matches and paying the $O(n^2)$ complexity. A very small fraction (low threshold) approximates the *FirstFit* approach in terms of effectiveness, but not efficiency, as it counts the matches but then largely disregards them.

Figure 8 shows the effect of varying the *BestFit* threshold on the `Slashdot` data set using 84 fingerprints per chunk and an average chunk size of 1 KB. The graph indicates that as the threshold increases, effectiveness increases, up to about 90%. A threshold of one corresponds to the most precise match, but it actually misses opportunities for delta-encoding, resulting in increased encoding size. A threshold between 0.7 and 0.9 yields the smallest encoding sizes with regular fingerprints for the `Slashdot` data set; other data sets show similar trends. As expected, the figure also shows increasing relative times as the threshold increases.

5.3.3 Effect of Shingle Size

A shingle specifies the size of a window that slides over the entire file advancing one byte a time, producing a Rabin fingerprint value for each fixed-size set of bytes. The Rabin fingerprints are used to flag content-defined chunk boundaries and to generate features for each chunk that can be used to identify similar ones. We performed experiments varying the shingle size, and using the `Slashdot` and `MH` data sets with 84 fingerprints per chunk, 14 super-fingerprints, an average chunk size of 4 KB, and the *FirstFit* variant.

With the `Slashdot` data set, shingles of four or eight bytes get much less benefit from REBL than larger sizes, but otherwise the analysis is noisy and several disjoint values give similar results. Shingle sizes of 20 and 44 bytes yield similar relative encoding sizes of 2.25% and 2.19% respectively and shingle sizes of 12 and 24 bytes yield relative sizes of 2.52% and 2.53% respectively, whereas a shingle size of 16 bytes results in a relative size of 5.12%. The maximum relative size of 6.84% is obtained with a shingle size of 8 bytes and the minimum of 2.19% with a shingle size of 44 bytes.

In the case of the `MH` data set, we found that varying the size of a shingle did not vary the relative sizes by as much as in the `Slashdot` data set, though they did impact processing time. The minimum relative size was 31.2% with a shingle size of 8 bytes and maximum relative size was 32.5% with a shingle size of 44 bytes. However, the 8-byte shingles took 156 CPU-seconds, while 12-byte shingles took 114 CPU-seconds (27% less) to encode the data set to 31.5% of the original (0.7% more).

We conclude that past work that used four-byte shingles [18] may have found their resemblance detection system to be noisier than necessary, but sizes of twelve bytes or more are probably equally arguable.

6 Conclusions and Future Work

In this paper, we introduced a new encoding scheme for large data sets: those that are too large to encode monolithically. REBL uses techniques from compression, duplicate block suppression, delta-encoding, and super-fingerprints for resemblance detection. We have implemented REBL and tested it on a number of data

sets. The effectiveness of REBL compared to TGZ varied by factors of 0.59-2.46, WFC by 1.28-14.25, SLIDINGBLOCK by 1.18-2.56, CDC by 1.03-6.67 and DERD by 0.88-2.91.

We have compared two variants for similarity detection among blocks, *FirstFit* and *BestFit*, and demonstrated that *FirstFit* with super-fingerprints produces a good combination of space reduction and execution overhead. Super-fingerprints are good approximations of regular fingerprints in all the data sets we experimented with. A low threshold of matching super-fingerprints usually results in similar effectiveness to that obtained using a higher threshold for regular fingerprints, but with a dramatically lower execution cost (orders of magnitude in some cases).

The effectiveness of REBL in our experiments is always better than WFC and CDC. However, this is because it incorporates the technology of compression at the file and block level, and the suppression of duplicate blocks, before adding delta-encoding. In fact, compressing individual chunks (or blocks) in any sort of CDC or SLIDINGBLOCK system seems an essential optimization unless the rate of duplication is substantially higher than we have seen in these data sets. This is consistent with the earlier SLIDINGBLOCK work [8], which found that SLIDINGBLOCK needed to incorporate block-level compression to be competitive with *gzip*. Compressing entire files when none of its pieces are suppressed as a duplicate similarly offers benefits.

In some cases, REBL encodes noticeably better than DERD; in the other cases, REBL is very similar to it. The key difference between REBL and DERD comes from dramatic reductions in execution times. The average block size used to mark content-defined blocks affects the encoding sizes of REBL to a limited extent; the more similarity there is in a data set, such as `Slashdot`, the more effective smaller blocks are.

We are currently working on extending and experimenting with the REBL and LBFS techniques to reduce network usage in communication environments. This will be helpful to determine the applicability of REBL in reducing redundant network traffic, and it can also be compared with other network-oriented mechanisms like *rsync* [23, 29] and link-level fingerprint-based duplicate detection [25]. We would also like to evaluate the effectiveness of these techniques in new environments, such as the Google GMail™ system, which may offer additional opportunities for large amounts of data with subtle variations. Finally, additional detailed comparisons of the wide variety of encoding techniques may offer the opportunity to consider new metrics, such as "bytes saved per cycle," in selecting among the alternatives.

Acknowledgments

We thank Ramesh Agarwal, Andrei Broder, Windsor Hsu, Arun Iyengar, Raymond Jennings, Leo Luan, Sridhar Rajagopalan, Andrew Tridgell, Phong Vo, Jian Yin, and the anonymous referees for comments and discussions. Special thanks go to our shepherd, Scott Kaplan, for his guidance and understanding. We thank David Mazieres and his research group for making the LBFS code available, Windsor Hsu and Tim Denehy for making the SLIDINGBLOCK code available, and Phong Vo and AT&T for making *vcodex* available.

References

[1] M. Ajtai, R. Burns, R. Fagin, D. Long, and L. Stockmeyer. Compactly encoding unstructured input with differential compression. *Journal of the ACM*, 49(3):318–367, May 2002.

[2] The Internet Archive. `http://www.archive.org/`, 2004.

[3] William J. Bolosky, Scott Corbin, David Goebel, and John R. Douceur. Single instance storage in windows 2000. In *Proceedings of the 4th USENIX Windows Systems Symposium*, August 2000.

[4] Andrei Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES'97)*, 1997.

[5] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In R. Giancarlo and D. Sankoff, editors, *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, number 1848 in Lecture Notes in Computer Science, pages 1–10, Montréal, Canada, 2000. Springer-Verlag, Berlin.

[6] Mun Choon Chan and Thomas Y. C. Woo. Cache-based compaction: A new technique for optimizing web transfer. In *Proceedings of Infocom'99*, pages 117–125, 1999.

[7] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, pages 285–298. USENIX, December 2002.

[8] Timothy E. Denehy and Windsor W. Hsu. Reliable and efficient storage of reference data. Technical Report RJ10305, IBM Research, October 2003.

[9] Fred Douglis and Arun Iyengar. Application-specific delta-encoding via resemblance detection. In *Proceedings of 2003 USENIX Technical Conference*, June 2003.

[10] Val Henson. An analysis of compare-by-hash. In *9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, 2003.

[11] David G. Korn and Kiem-Phong Vo. Engineering a differencing and compression data format. In *Proceedings of the 2002 Usenix Conference*. USENIX Association, June 2002.

[12] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Computing, Springer Verlag (Heidelberg, FRG and*

NewYork NY, USA)-Verlag Surveys, ; ACM CR 8902-0069, 19(3), 1987.

[13] U. Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, January 1994.

[14] J. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Goland, A. van Hoff, and D. Hellerstein. *Delta encoding in HTTP*, January 2002. RFC 3229.

[15] Jeffrey Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *Proceedings of ACM SIGCOMM'97 Conference*, pages 181–194, September 1997.

[16] Jeffrey C. Mogul, Yee Man Chan, and Terence Kelly. Design, implementation, and evaluation of duplicate transfer detection in HTTP. In *Proceedings of the First Symposium on Networked Systems Design and Implementation*, March 2004.

[17] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *Symposium on Operating Systems Principles*, pages 174–187, 2001.

[18] Zan Ouyang, Nasir Memon, Torsten Suel, and Dimitre Trendafilov. Cluster-based delta compression of a collection of files. In *International Conference on Web Information Systems Engineering (WISE)*, December 2002.

[19] Calicrates Policroniades and Ian Pratt. Alternatives for detecting redundancy in storage systems data. In *Proceedings of the 2004 Usenix Conference*, June 2004.

[20] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of the First USENIX Conference on File and Storage Technologies*, Monterey,CA, 2002.

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

[22] Sean C. Rhea, Kevin Liang, and Eric Brewer. Value-based web caching. In *Proceedings of the twelfth international conference on World Wide Web*, pages 619–628. ACM Press, 2003.

[23] rsync. `http://rsync.samba.org`, 2004.

[24] rzip. `http://rzip.samba.org/`, 2004.

[25] Neil T. Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of ACM SIGCOMM*, August 2000.

[26] Federal Information Processing Standards. Secure hash standard. FIPS PUB 180-1, April 1995.

[27] Torsten Suel, Patrick Noel, and Dimitre Trendafilov. Improved file synchronization techniques for maintaining large replicated collections over slow networks. In *Proceedings of ICDE 2004*, March 2004. To appear.

[28] W. Tichy. RCS: a system for version control. *Software—Practice & Experience*, 15(7):637–654, July 1985.

[29] Andrew Tridgell. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, Australian National University, 1999.

[30] Andrew Tridgell. Personal communication, December 2003.

Notes

[1] In practice we do not expect the complexity to be this bad, and some heuristics could be used to reduce it [30], but they are beyond the scope of this paper. Even with such optimizations, the techniques described in this paper improve efficiency substantially.

[2] Encoding chains are possible—A against B, B against C, and so on—but decoding such a chain requires first computing B from C to obtain A. We discount this possibility due to its complexity and performance implications.

[3] Our initial implementation stored the relationship of every pair of blocks with at least one matching fingerprint. With this approach, we ran out of address space operating on our larger data sets. We reduced memory usage by storing a information only for blocks matching many fingerprints (a default of $\frac{1}{4}$), but even that approach suffers on extremely large data sets.

[4] Other sophisticated techniques may be used to generate super-fingerprints, but in our case, we needed a hashing function with a low probability of collisions and MD5 satisfied the criteria.